САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СПИСОК-2011

МАТЕРИАЛЫ ВТОРОЙ МЕЖВУЗОВСКОЙ НАУЧНОЙ КОНФЕРЕНЦИИ ПО ПРОБЛЕМАМ ИНФОРМАТИКИ

27–29 апреля 2011 г.,
Санкт-Петербург

Санкт-Петербург
ВВМ
2011

УДК 004(063)
С72

Печатается по рекомендации
кафедры системного программирования
Санкт-Петербургского государственного университета

С72 Список-2011: материалы межвуз. науч. конф. по проблемам информатики. 27–29 апр. 2011 г., Санкт-Петербург. — СПб.: ВВМ, 2011. — 476 с.

ISBN 978-5-9651-0577-9

Тематика сборника затрагивает широкий круг актуальных проблем теоретической и прикладной математики и информатики. Слово «СПИСОК», ставшее названием конференции, — это не только обозначение фундаментальной структуры данных, но и сокращение от названий трех направлений исследований: Системное Программирование, Интеллектуальные Системы, Обеспечение Качества. Результаты исследований в этих областях являются значительной частью того знания, которое в настоящее время можно назвать словом «информатика».

Для студентов и аспирантов естественно-научных специальностей.

ISBN 978-5-9651-0577-9

Наше время быстрого развития прикладных наук и технологий настоятельно требует интенсивного обмена идеями, знаниями, новыми результатами исследований. И конференция «СПИСОК», безусловно, способствует такому обмену, вовлекает в орбиту исследований и разработок талантливую молодежь.

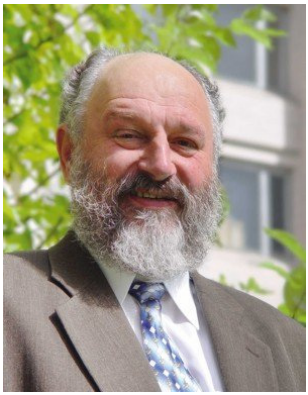
Очень важно, что у истоков этой конференции стояли авторитетные научные коллективы, возглавляющие такими лидерами науки, как профессора А. Н. Терехов и В. Е. Третьяков. Это сразу привлекло внимание многих специалистов, стало основой для формирования традиций будущих конференций.

Желаю участникам конференции «СПИСОК-2011» успешной работы и создания новых актуальных знаний.

Председатель оргкомитета
конференции «СПИСОК-2011»,
д.ф.-м.н., профессор, чл.-корр. РАН,
декан математико-механического факультета
СПбГУ, заведующий кафедрой прикладной
кибернетики СПбГУ

Г. А. Леонов

Системное программирование



Терехов
Андрей Николаевич

председатель программного комитета конференции
д.ф.-м.н., профессор
заведующий кафедрой системного программирования СПбГУ
директор НИИ Информационных технологий СПбГУ
генеральный директор ЗАО Ланит-Терком

USE CASE: ОТЛАДКА РЕАЛИЗАЦИИ RISC ПРОЦЕССОРА ДЛЯ FPGA

О. Медведев (e-mail: Oleg.Medvedev@lanit-tercom.com)

ЗАО «Ланит-Терком»

Abstract: В работе рассмотрен процесс тестирования и отладки реализации RISC процессора для программируемой логической интегральной схемы. Показано, чем он отличается от «обычной» отладки, привычной программистам. Изложен метод, который автору пришлось применить, чтобы выловить плохо воспроизводимые ошибки, из-за которых на процессоре не запускалась ОС Linux.

Введение

Отладка синхронных цифровых интегральных схем (ИС) существенно сложнее отладки обычных последовательных программ такого же размера (если мерять в строчках кода) по причине того, что в ИС обычно содержится множество различных блоков, которые работают параллельно и обмениваются друг с другом информацией, зачастую, на каждом такте. Более того, поскольку каждый блок занимает определенную площадь в кремниевой реализации ИС, то разработчики стараются, чтобы все блоки, по-возможности, все время выполняли какую-то полезную деятельность. Таким образом, в ИС можно наблюдать множество сложных ошибок взаимодействия, которые в принципе не существуют в последовательных программах, а только лишь в параллельных. Например, дедлоки, race conditionы.

Отладка ИС, по моему мнению, сложнее отладки параллельной программы, потому что в параллельной программе несколько потоков обмениваются относительно крупными сообщениями относительно редко (каждая посылка сообщения или синхронизация по доступу к общей памяти может занимать тысячи процессорных тактов). В то же время, блоки, живущие внутри ИС, могут обмениваться маленькими сообщениями (по нескольку бит/десятков бит) на каждом такте работы (при частоте в сотни мегагерц).

Более того, даже если обычную программу по какой-то причине нельзя запускать под отладчиком, в нее можно добавлять довольно серьезные возможности ведения отладочного лога. При этом, чтобы внести изменения в информацию, подаваемую в лог, достаточно перекомпилировать некоторые модули программы (логично ожидать, что это занимает десятки секунд-минуты). В то же время, возможности ведения подобных логов изнутри ИС весьма ограничены, потому что лог надо сохранять в какую-то память. При работе на высокой скорости вывод в лог даже небольшого числа переменных (скажем, двух слов по 32 бита) может переполнить все каналы доступа к любой *внешней* памяти. Значит, чтобы успевать писать лог, надо

существенно замедлять работу ИС. Это сказывается на размере теста, который возможно пропустить за разумное время. Кроме того, изменив описание ИС, прежде чем иметь возможность запустить его на ПЛИС, необходимо пропустить его через несколько средств автоматизированного проектирования, что занимает десятки минут-часы.

В данной работе мы рассматриваем проблему отладки RISC-процессора (клона Xilinx Microblaze). Процессор реализовывался для работы на программируемой логической интегральной схеме (ПЛИС, FPGA). Конечной целью отладки является успешный запуск и работа на процессоре ОС Linux. Приводится описание методов тестирования и отладки — как привычных для разработчиков ПО, так и более специфических.

Процессор с точки зрения программиста

Процессор является клоном Xilinx Microblaze [2] — RISC процессора для встроенных систем. Он реализует 32 32-битовых регистра, целочисленную арифметику (умножения, сдвиги), блок поддержки виртуальной памяти (MMU), защищенный режим работы, поддержку внешнего прерывания, интерфейс с внешней шиной для доступа к памяти и периферии.

С точки зрения программиста можно сказать, что процессор все инструкции исполняет *последовательно* — он считывает через шину ровно одну инструкцию, полностью ее исполняет (сохраняя результат в регистр, либо во внешнюю память), после чего переходит к следующей инструкции.

Такую модель поведения очень легко реализовать в виде программного симулятора процессора, который, во-первых, является полной *исполняемой спецификацией* интерфейса между процессором и ПО, которое на нем исполняется, а, во-вторых, является *эталонной реализацией* процессора.

В нашем случае такой симулятор, будучи запущенным на Intel Core Duo 1.8GHz, исполнял порядка 2 млн. инструкций в секунду, чего было вполне достаточно, чтобы загрузить на нем линукс, а в линуксе — нагрузочный тест. Также не представляла сложности отладка симулятора, ибо он является очень простой *последовательной* программой.

Процессор с точки зрения его разработчика

С точки зрения инженера-разработчика ИС процессор представляет собой сильно распараллеленную версию описанной выше эталонной модели. Корректность реализации означает, что она ведет себя ровно так же, как эталон. Более формально, на каждом такте работы реализации, та часть ее состояния, которая видна программисту (например, содержимое регистров общего назначения или слово состояния), должна совпадать с состоянием эталона до или после исполнения очередной инструкции.

Оптимизация модели процессора основана на следующем простом рассуждении, которое я, тем не менее, изложу явно:

- заметим, что процессор обычно исполняет инструкции подряд, а переходы делает относительно редко;
- заметим, что между несколькими подряд идущими инструкциями часто нет зависимостей по данным (одна не читает регистры, которые записывает другая);
- значит, мы можем загружать инструкции пачками по нескольку штук и исполнять их параллельно (точнее — конвейерно).

В реальности конвейер получается не совсем линейный, а разветвленный, как показано на рис. 1. В отдельную ветвь выносится умножитель, поскольку умножение занимает несколько тактов. То же можно сказать про проверку правил доступа к виртуальной памяти и последующий доступ к шине данных. Кроме того, запись бита переноса происходит на 2 такта раньше записи всех других регистров (чтобы не замедлять работу инструкций, которые идут подряд, но зависят друг от друга только по биту переноса). Последнее требует поддержки отката его значения в случае получения исключения от шины данных позднее.

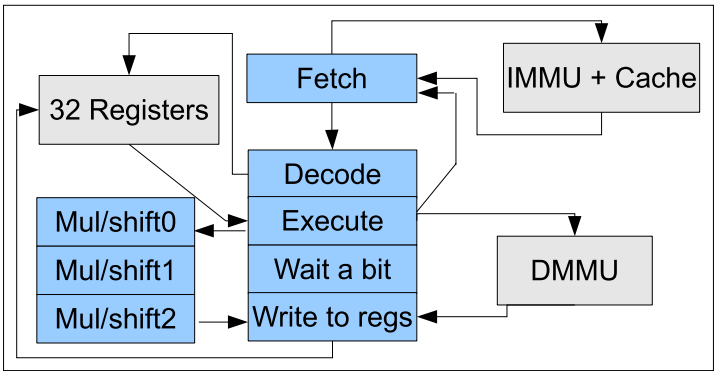


Рис. 1. Примерный вид конвейера, использованного в реализации процессора

Также оптимизация подразумевает кеширование по меньшей мере правил обращения к виртуальной памяти и инструкций.

Тестирование и отладка

В первую очередь тестируется каждый блок процессора отдельно — происходит нечто подобное юнит-тестированию ПО. Например, для блока «стадия конвейера Execute», который получает частично декодированную инструкцию и значения тех регистров, которые она использует, для каждого возможного типа инструкции проверяется, что выходы Execute корректны, то есть

1. инструкция передается далее в стадию «Wait a bit»;
2. если инструкция была переходом с истинным условием (либо безусловным переходом), то высылается сообщение блоку загрузки инструкций о том, что поток управления сменился, а также блоку контроля конвейера сообщается, что результаты исполнения всех уже загруженных инструкций, идущих за переходом, надо будет аннулировать;
3. если инструкция обращается к памяти, то посылается сообщение в DMMU о старте обращения;
4. и т. д. — данный список приведен лишь чтобы показать сложность интерфейса некоторых отдельных блоков.

Подобные тесты хороши тем, что они вычислительно просты и их можно исполнять на тактово-аккуратном симуляторе ИС на инструментальной машине, имея возможность отлаживать ИС и изучать состояние любых внутренних сигналов и регистров.

Проблемой здесь является то, что не очень понятно, что тестируется. Интерфейс многих внутренних блоков (например, того же Execute) оказывается не проще самого блока и допустить ошибку в его описании при помощи набора тестов очень легко (см., например, более общее обсуждение данной проблемы в [4]).

Таким образом, процессор, собранный из протестированных блоков, также приходится отлаживать. Частично в этом помогают тесты на специфические функции процессора, написанные на ассемблере. Например, мне пригодились тесты на

- все классы инструкций;
- виртуальную память отдельно;
- все типы исключений;
- прерывания.

Однако, и такого тестирования на практике оказывается недостаточно — попытки запустить на процессоре линукс оказываются неудачными. В отличие от простых ассемблерных тестов, локализовывать источник ошибки процессора в линуксе практически невозможно, потому что ошибка становится заметной через десятки тысяч инструкций после того, как она произошла. За это время успевают произойти переходы управления через десятки функций, иногда с переключением активного процесса. Чтобы локализовать такую ошибку, надо пройти по этому пути обратно, сверяясь с исходниками ядра и пользовательских приложений, а также с их расположением в памяти в данном конкретном экземпляре неработающего линукса. Это может занять недели на каждую ошибку.

Этому методу есть альтернативы:

- распечатать исходный код процессора и пытаться найти ошибку методом вдумывания в текст, зная, что она есть (не у всех получается на таких объемах исходников);
- не пытаться переходить от простейших тестов сразу к линуксу, а усложнять запускаемую программу постепенно (где взять столько программ?);

- доказывать эквивалентность реализации и эталона формально, с использованием моделчекера (в случае успеха, этот путь найдет все ошибки, но выглядит он довольно специфически, являясь, фактически, полуавтоматическим построением математической теории; не всегда удается уговорить на это начальство);
- запускать линукс синхронно на реализации процессора и на эталонной модели, регулярно сравнивая состояния двух моделей и останавливаясь в случае расхождения.

Последний способ был использован в моем случае. Он будет описан в следующем разделе.

Быстрая локализация ошибок в процессоре при запуске под линуксом

Как было замечено выше, основная проблема поиска ошибок в процессоре, если они проявляются только под линуксом — в их локализации. Если бы можно было остановить работу процессора ровно на той инструкции, которую он исполнил некорректно (или с погрешностью в несколько инструкций), то анализ этих инструкций скорее всего привел бы к пониманию причины ошибки весьма быстро. Более того, имея возможность останавливать работу системы сразу после ошибки, можно реализовать аппаратный лог внутренних сигналов процессора, хранимый в памяти на кристалле. В такой лог можно записывать информацию на очень большой скорости — сотни битов на каждом такте, но в ограниченном объеме (порядка тысячи последних тактов).

Пожелание «точной остановки» легко выполнить, имея корректно работающую эталонную модель процессора. Для этого достаточно запустить аппаратную реализацию процессора и эталонную модель с одинаковым содержимым оперативной памяти (имея в виду, что линукс стартует прямо из памяти) и сравнивать результаты исполнения каждой инструкции. Остановку производить при первом расхождении.

Под результатом исполнения инструкции разумно понимать номер и содержимое записанного регистра для инструкций, которые пишут в регистр. Эта информация имеет ширину 32 + 5 + 1 = 38 бит. Кроме этого, необходимо передавать информацию о том, вместо какой инструкции было возбуждено прерывание (потому что эталонная модель этого знать не может), а также результаты чтения с любых периферийных устройств, которыми пользуется линукс (потому что у эталонной модели есть только своя копия памяти, но не периферийных устройств).

В моем случае связь между отлаживаемым устройством и инструментальной машиной, исполняющей эталонную модель, была организована через канал ethernet, причем скорость передачи оказывалась в районе 11МБит/с. Это означало бы (в худшем случае) передачу информации о примерно 300 ты-

сячах инструкций в секунду, что почти в 10 раз меньше той скорости, на которой может работать эталонная модель (сама реализация работала на частоте 50МГц). По этой причине было решено передавать не все данные целиком, а только их хэш (+ информацию о прерываниях и чтении с периферии). Применялся довольно простой алгоритм вычисления контрольной суммы «Adler 32».

Данная методика позволяла остановить тестирование примерно через 500 тактов после обнаружения расхождения. Лог внутреннего состояния процессора сохранял состояние стадий конвейера и некоторые другие данные за последние 1500 тактов, что позволяло легко найти место расхождения лога процессора с логом эталона и локализовать ошибку за несколько часов.

Заключение

В работе приведено краткое описание клона процессора Xilinx Microblaze, реализованного на технологии FPGA при помощи языка HaSCoL [1, 3] с участием автора и рассмотрена методика отладки процессора на сложном тесте (ОС Linux), позволяющая локализовывать ошибки в процессоре за часы, а не за недели, как могло бы быть при более обычном подходе.

Автор выражает благодарность коллегам из ЗАО «Ланит-Терком», принимавшим участие в обсуждении и реализации процессора.

Литература

[1] *Boulytchev D., Medvedev O.* Hardware Description Language Based on Message Passing and Implicit Pipelining // East-West Design and Test (EWDTS), 2009. Pp. 279–282.

[2] MicroBlaze Processor Reference Guide // http://www.xilinx.com/support/ documentation/sw_manuals/mb_ref_guide.pdf

[3] http://oops.math.spbu.ru/projects/coolkit

[4] *Gerard J. Holzmann.* The Spin Model Checker: Primer and Reference Manual // Гл. 1. С. 3.

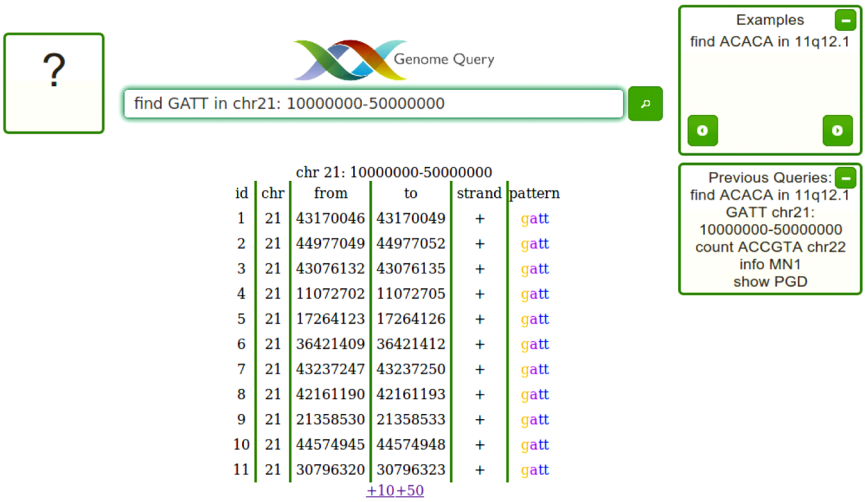
ИНСТРУМЕНТ АНАЛИЗА ГЕНОМА ЧЕЛОВЕКА GENOME QUERY

В. Дудин, А. Кладов, Н. Кочнева, Е. Соса, А. Алеев, А. Пржибельский, В. Рассохин

Научный руководитель:
Н. Вяххи

Genome Query — проект, цель которого — обеспечить удобный и мощный доступ к геномной информации. Проект разрабатывается для биологов и генетиков, которым необходимо выполнять различные поисковые запросы и получать статистические данные о нуклеотидных и белковых последовательностях в геноме. Это необходимо ввиду невозможности ручной обработки больших объёмов информации. К процессу разработки привлекаются биологи в качестве научных консультантов, что помогает создавать функциональное и удобное приложение.

Планируется три варианта реализации проекта: в виде библиотеки, в виде web-сайта и в виде web-API. В настоящее время наиболее функционален web интерфейс. Так выглядит главная страница:



На данный момент реализован поиск и подсчет количества вхождений заданной последовательности нуклеотидов в различных областях генома, демонстрация заданной области. Так же приложение способно выводить биологически значимую информацию об участке генома. Разработаны алгорит-

мы неточного поиска, например, поиск подстроки, отличающейся от образца не более чем на m несовпадений.

Для удобства работы с приложением создан интуитивно понятный и «дружественный» web-интерфейс, позволяющий использовать сервис без дополнительных программных средств.

В целях облегчения работы с геномом, описан и разработан специальный язык запросов, позволяющий четко и лаконично формулировать поставленную задачу.

Алгоритмическая часть приложения реализована с помощью двух различных структур: суффиксные массивы и индекс *k*-меров. В будущем это позволит выполнять поиск максимально быстро, применяя тот или иной алгоритм в зависимости от типа запроса.

Уникальность приложения заключается в наличии метаданных к отдельным участкам генома. Они позволяют осуществлять поиск по определенным генам или другим биологически значимым областям генома.

В будущем планируется добавить возможность работы более чем с одним геномом. Это позволит различным исследовательским институтам и лабораториям осуществлять поисковые запросы по собственным данным.

В настоящее время разрабатываются алгоритмы поиска подстрок, удовлетворяющих заданному регулярному выражению. Также планируется расширить функциональность языка запросов.

Использованные технологии:

- JavaEE
- Javascript
- Git
- Youtrack
- Teamcity

Ключевые слова:

Биоинформатика, геном, данные, поиск.

Ссылки:

- http://confluence.jetbrains.net/display/GQRY
- http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/
- http://genome.ucsc.edu/

МНОГОШТРИХОВЫЕ ЖЕСТЫ МЫШЬЮ В ПРОЕКТЕ QREAL

М. С. Осечкина

Научный руководитель:
Ю. В. Литвинов

Санкт-Петербургский государственный университет

В CASE-системах при добавлении объектов на диаграмму и выполнении элементарных действий над объектами (например, удаление), а также в некоторых редакторах при рукописном вводе текста (например, написание sms-сообщений в последних моделях Samsung) пользователю предоставляется возможность управлять приложением с помощью жестов мышью. Распознавание команды, ассоциированной с жестом происходит по сигналу: отпускание кнопки мыши, нажатие заданной комбинации клавиш, тайм-аут и так далее. Штрих — фигура, получившаяся в результате движения мыши между нажатием кнопки и ее отпусканием. Многоштриховые (multi-stroke) жесты — жесты, состоящие из нескольких штрихов. Жесты мышью — хороший способ сделать взаимодействие с приложениями для пользователя более удобным. Рассмотрим задачу добавления многоштриховых жестов мышью в CASE-систему QReal [1], разработанную на кафедре системного программирования математико-механического факультета СПбГУ.

Изначально предполагалось, что объекты на диаграмму надо добавлять перетаскиванием с помощью операции drag-and-drop из палитры. Чтобы ускорить добавление элементов были реализованы одноштриховые жесты мышью [2]. Но при такой реализации учитывались точка начала и направление движения мыши. Чтобы пользователю было удобнее работать с данной системой, хотелось бы избежать необходимости запоминания подобных деталей. Таким образом возникла потребность замены одноштриховых жестов мышью на многоштриховые, которые независимы от количества штрихов, точек начала и направления движения мыши.

Как правило алгоритм распознавания разбивается на следующие этапы:

1) **Определение пути мыши.** Программа получает сигнал о том, что кнопку мыши нажали или отпустили или мышь двигают, зажав кнопку. У сигнала есть метод, возвращающий координаты курсора мыши. Заносим эти координаты в список точек соответствующего штриха.
2) **Сглаживание пути** [3]. Этот шаг желателен, так как фильтрация помогает минимизировать влияние дрожжания руки на распознаваемость, к тому же различное оборудование дает разную точность при получении позиции мыши.
3) **Построение классификатора**. Сравнение списка точек затруднительно, так как полученный список зависит от оборудования, скорости движе-

ния мыши, масштаба — факторов, которые не обязательно учитывать при распознавании. Вводится множество признаков жеста, по которым производится сравнение. Это множество признаков называется классификатором. На пространстве классификаторов определяется расстояние.

4) **Выбор объекта.** Идеальные жесты — база эталонов, с которыми сравниваются нарисованные пользователем жесты. По объекту генерируется набор штрихов, изображение которых совпадает с графическим представлением объекта. Вычисляется расстояние между классификатором для нарисованного жеста и классификатором для каждого из идеальных жестов. Ищется объект, соответствующий минимальному расстоянию между идеальным жестом и жестом пользователя. Если это расстояние не превосходит максимально возможное расстояние до идеального жеста, генерируется объект.

Для улучшения распознаваемости пользовательских жестов можно использовать обучение. На основе базы жестов пользователя, о каждом из которых известно, к какому классу он относится, можно изменить параметры алгоритма распознавания так, что и другие жесты пользователя, подаваемые на вход, будут распознаваться с большей вероятностью.

Широкое распространение среди алгоритмов обучения получили нейронные сети [4], метод k ближайших соседей и метод Байеса [5].

Для поддержки мышиных жестов в QReal был выбран алгоритм корректировки центра масс класса жестов — алгоритм k-средних [6].

Введем понятие «построенный жест». Построенный жест представим как список ячеек: сначала около жеста пользователя описывается прямоугольник со сторонами, параллельными осям координат, далее этот прямоугольник разбивается на равные ячейки прямыми, параллельными горизонтальным и вертикальным сторонам прямоугольника, затем составляется список ячеек, через которые проходит жест пользователя. Количество ячеек по высоте и по ширине прямоугольника заранее задано.

Мы рассмотрели порядка десяти классификаторов. Приведем два классификатора, показавших наилучший результат.

Матрица расстояний до жеста. Пусть h — высота описанного около жеста прямоугольника, w — ширина прямоугольника, M — матрица $w * h$, элемент матрицы M_{ij} равен расстоянию μ от ячейки с координатами $[i, j]$ ДО ближайшей ячейки построенного жеста (евклидово или максимум модуля разности координат или сумма модулей разности координат). Получили матрицу расстояний до построенного жеста. Расстояние между двумя построенными жестами — это норма разности соответствующих им матриц.

Количество ячеек в прямоугольнике. Следующий алгоритм заключается в подсчете количества ячеек в прямоугольнике $m * n$, где m — меняется от 1 до высоты прямоугольника, n — от 1 до ширины прямоугольника.

Строится матрица $h*w$, где h — высота прямоугольника, w — ширина прямоугольника. Элемент $[i, j]$ равен количеству ячеек жеста, содержащихся в прямоугольнике $i*j$, при этом если жест проходит через одну и ту же ячейку несколько раз, в матрицу заносится только одно прохождение. Расстояние между жестами определяется как расстояние между векторами в R^{h*w} (евклидово или максимум модуля разности координат или сумма модулей разности координат), при этом матрица рассматривается как вектор из этого пространства. Строки матрицы выписываются в одну строку, таким образом получаем вектор.

Первый описанный классификатор показал распознавание 71 % жестов на базе из 1177 пользовательских жестов, второй — 57 %. Однако, второй классификатор давал хорошее распознавание жестов из тех классов, которые плохо распознавались другими классификаторами. Поэтому был построен классификатор, являющийся комбинацией данных двух классификаторов. Соответствующее расстояние задавалось как линейная комбинация с положительными коэффициентами двух вышеприведенных расстояний. Распознавание увеличилось до 76 %, а после обучения на обучающей выборке из 45 жестов (5 жестов в классе, 9 классов) обучение увеличилось до 90 %.

90 % — вполне удовлетворительный результат при распознавании жестов. Сейчас планируется добавить многоштриховые жесты в QReal, предварительно создав обучающую базу жестов для всех объектов.

Литература

1. *А. Н. Терехов, Т. А. Брыксин, Ю. В. Литвинов* и др. Архитектура среды визуального моделирования QReal // Системное программирование. Вып. 4: Сб. статей / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб., 2009. С. 171–196.

2. *М. С. Осечкина, Т. А. Брыксин, Ю. В. Литвинов* и др. Поддержка жестов мышью в мета-CASE-системах // Системное программирование. Вып. 5: Сб. статей / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб., 2010.

3. *Xuejiang Cheng.* Self-adjusting digital filter for smoothing computer mousemovement // Freepatentsonline. URL http://www.freepatentsonline.com/5661502.pdf (Дата обращения: 14.02.2010).

4. *Don Willems, Ralph Niels, Marcel van Gerven, Louis Vuurpijl.* Iconic and multistroke gestures recognition // Radboud University Nijmegen, URL http://www.cs.ru.nl/~marcelge/papers/willems2009.pdf (Дата обращения: 3.02.2011).

5. *Cesar F. Pimentel, Manuel J. da Fonseca, Joaquim A. Jorge.* Experimental evaluation of a trainable scribble recognizer for calligraphic interface // Graphics recognition: algorithms and applications. Eds. by Dorothea Blostein, Young-Bin Kwon. Ontario, Canada, 2001. Pp. 81–85.

6. *MacQueen, J. B.* Some Methods for Classification and Analysis of Multivariate Observations // Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, 1967. Pp. 281–297.

ПОДХОД К РАЗРАБОТКЕ РАСПРЕДЕЛЕННЫХ ГЕТЕРОГЕННЫХ РЕАКТИВНЫХ СИСТЕМ

Н. Е. Соколов (e-mail: Nikolay.Sokolov@lanit-tercom.com),
Д. В. Луцив (e-mail: dluciv@math.spbu.ru)

Санкт-Петербургский государственный университет

При разработке поведения некоторой компоненты программы, подходить к описанию её реакции на внутренние и внешние события можно разными способами. Наиболее распространенный — компонента содержит в себе некоторое количество функций, которые по реакции на событие вызываются в определенном порядке. Таким образом изменяется состояние системы. Этот подход имеет ряд недостатков, таких как возможное нарушение некоторых условий, которым компонента должна удовлетворять в любой момент времени. Или разработчик просто может забыть вызвать определенную функцию или присвоить определенное поле, что впоследствии приведет к ошибке.

В данной работе предлагается альтернативный подход к разработке программных компонент. Каждая компонента в ответ на событие извне сама приводит себя в корректное состояние, изменяя значения полей в соответствии с заданными правилами. Таким образом напрямую указанное изменение полей компоненты должно свестись к минимуму, поскольку каждое поле само следит за своим состоянием, поддерживая корректность состояния компоненты вцелом. Такой подход к созданию систем называется реактивным программированием.

- Реактивное программирование для систем реального времени определяется [1], как система, обладающая следующими свойствпостоянно взаимодействует со своим окружением, причем это взаимодействие может носить асинхронный, непредсказуемый характер;
- имеет свойство прерываемости, т. е. должна быть готовой обрабатывать запросы наивысшего приоритета в то время, когда она занята какой-либо другой работой;
- реакции системы на запросы часто имеют строгие временные ограничения;
- имеет различные сценарии, которые зависят от значения каких-либо данных и от прошлого (истории) системы;
- параллельность.

Подход предлагаемый в данной работе не предполагает своего использования в системах реального времени, что позволяет наложить на нее некоторые ограничения, но это даст определенные плюсы, о которых будет рассказано чуть позже.

Как уже раньше говорилось, одним из преимуществ событийно-ориентированного (а в частности и реактивного) подхода к разработке является гарантия нахождения системы в корректном состоянии. Наличие же свойства постоянного взаимодействия с окружением лишает систему этой особенности, поэтому было принято решение от него отказаться.

Свойство прерываемости оставляется без изменений.

По-умолчанию нет никаких временных ограничений на время обработки компонентой события, но при этом планируется предоставление разработчику возможности задать такое ограничение.

Наличие различных, зависящих от состояния системы сценариев и параллельность работы работы также поддерживается.

Выше были заданы свойства, которым должны удовлетворять реактивные системы, но при этом не было дано определение реактивных систем.

Для иллюстрации приведем пример реактивного связывания двух переменных:

$$x <\sim y + 2;$$

Это значит, что при любом изменении y, которое, как и x, в таком случае называется не переменной, а поведением, будет автоматически вычислено новое значение x. Таким образом для компоненты будет всегда верно, что x на 2 больше y. Правда надо заметить, что это верно только в том случае если нигде нет присваиваний x, т. к. обратного вычисления значения нет.

Реактивно связывать также можно и порты компонент, по-сути реализуя поведение объемлющего модуля, содержащего в себе все компоненты. Причем таких уровней вложенности может быть бесконечно много.

Таким образом реактивная система представляет собой набор взаимосвязанных компонент, которые в свою очередь состоят из таких же взаимосвязанных компонент.

В настоящее время реактивное программирование применяется в основном в системах реально времени [1] и при проектировании интерфейсов пользователя [8, 9].

Для реализации таких систем был предложен язык RL. В данный момент он включает в себя только графическую нотацию. Являясь во многом продолжателем языков REAL [2] и RTST [3] , он перенял часть конструкций них и таких языков, как ROOM [4] и SDL [5]. Помимо перечисленных выше языков, нотация RL обладает общностью с нотацией языка определения потоков данных функционально-реактивной библиотеки Yampa [6] для языка Haskell.

Основной конструкцией языка является компонента. Она может содержать в себе исключительно экземпляры других компонент и поля данных. Ниже приведен пример компоненты.

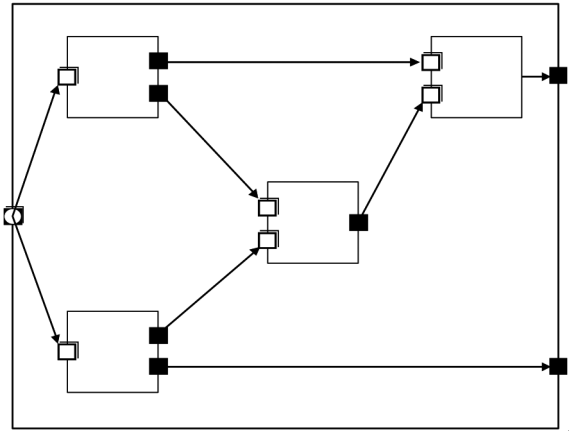


Рис. 1. Пример компоненты

Также у компоненты имеется некоторое количество входных и выходных портов. Через них происходит ее общение с другими компонентами.

Компоненты по их поведению можно разделить на две группы — трансформеры и модули.

Первые, трансформеры, просто принимают на вход некоторое количество данных и дают на выходе полученные только из них результаты. Они являются аналогом чистых функций со многими входами и многими выходами из других языков программирования.

Другой вид компонент, модули, хранят в себе некоторые данные, и вычисляют выходные результаты не только на основе входных, но и на их основе. Ниже представлены изображения трансформеров (слева) и модулей (справа).

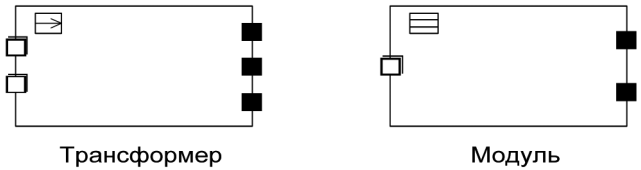


Рис. 2. Компоненты

Компоненты, составляющие систему, не обязательно должны располагаться на одной машине, и не обязательно должны быть написаны на одном языке. Предполагается реализация возможности обертки отдельных модулей, написанных на других языках для придания им поведенческих свойств. Модули при этом будут общаться между собой посредством универсальных сетевых протоколов, например, TCP/IP. При помощи их же модули могут получать события от других нереактивных частей системы.

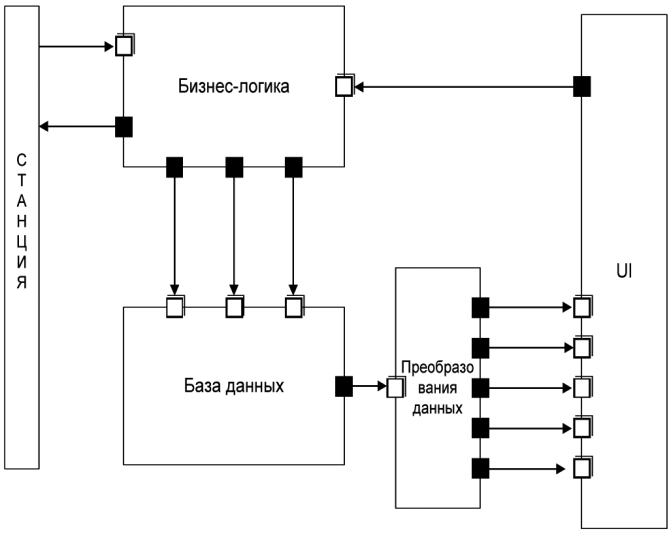


Рис. 3. РМО

Рассмотрим пример описания системы на данном языке. Система — упрощенная версия РМО (Рабочее Место Оператора) для телефонной станции.

Опишем схему работы системы в целом и её составных частей.

От станции могут приходить сообщения об изменении состояния телефонных линий. Также станция может получать сообщения от бизнес-логики, извещающие об изменении состояния конкретной телефонной линии с интерфейса пользователя. Станция является внешней компонентой, общающейся с РМО посредством универсальных сетевых протоколов, например, TCP/IP.

Задача логики — синхронизация данных от событий, полученных от станции и событий от интерфейса пользователя. Логика может инициировать изменения в базе данных. В терминологии языка RL Бизнес-логика в терминах языка является модулем.

В базе данных хранится полная информация, необходимая для работы с телефонными линиями. Помимо реактивного входного порта для логики, база данных имеет выходной порт для модуля, отвечающего за отображение данных на интерфейсе. Поскольку сама база данных не имеет поведенческого свойства, ее придется обернуть кодом на каком-либо другом языке.

Между интерфейсом пользователя и базой данных существует дополнительный модуль, преобразующий данные, получаемые из базы, в данные, необходимые для отображения на интерфейсе. Так, например, информацию об одной линии требуется отобразить в двух местах: списке всех абонентов и в списке входящих вызовов. На вход преобразователь получит из базы данных информацию о линии ровно один раз. На выходе же преобразователь

уже отдаст ее в двойном экземпляре. Причем в каждом из них будет только информация необходимая для отображения соответствующего списка. Преобразователь данных — классический пример трансформера.

При сборке проекта системы, для каждой из компонент будет сгенерирован код на языке данной компоненты. Так, например, если трансформеры будут на некотором языке, отличном языка RL, то будет сгенерирован код для обертки именно на этом языке. Для компонент, написанных на RL будет сгенерирован код на C#. Исключением из этого правила является база данных, обертка над которой будет написана не на ее процедурном языке, а на C#.

Генерация кода по модели (в нашем случае — программе на RL) хорошо зарекомендовала себя в ряде технологических решений, в том числе и телекоммуникационных [10].

На данный момент разработан прототип языка, реализующий минимально-необходимый набор операций и конструкций, необходимых для полноценного программирования распределенных реактивных систем. Рассмотрение применения данного языка в рамках уже реализованного РМО для телефонной станции показало, что такой подход является удачным для программирования распределенных гетерогенных систем.

Помимо продолжения разработки и реализации языка планируется реализация графического редактора при помощи пакета QReal [7].

Литература

1. *D. Harel, M. Politi.* Modeling Reactive Systems with Statecharts: state machine approach. McGraw-Hill. 1998. 258 p.

2. *Кознов Д. В.* Визуальное моделирование компонентного программного обеспечения. Канд. дис. СПбГУ. 2000. 84 с.

3. *Парфенов. В. В.* Проектирование и реализация программного обеспечения встроенных систем с использованием объектно-базированного подхода. Диссертация на соикания степени кандидата ф.-м. наук. СПб.: Изд-во СПбГУ, 1995.

4. *B. Selic, G. Gullekson, P. T. Ward.* Real-Time Object-Oriented Modeling. John Wiley & Sons. Inc. 1994. 525 p.

5. ITU Recommendation Z.100: Specification and Description Language. 1993. 204 p.

6. *A. Courtney, H. Nilsson, J. Peterson.* The Yampa Arcade. Haskell'03 Proceedings of the 2003 ACM SIGPLAN workshop on Haskell. ACM Press 2003. 12 p.

7. *Терехов А. Н., Брыксин Т. А., Литвинов Ю. В.* и др. Архитектура среды визуального моделирования QReal // Системное программирование. Вып. 4. СПб.: Изд-во СПбГУ. 2009, С. 171–196.

8. *C. Elliott* and *P. Hudak.* Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming (ICFP'97). 1997. 12 p.

9. *L. Meyerovich, A.Guha, J. Baskin, G. Cooper, M. Greenberg, A. Bromfield, S. Krishnamurthi.* Flapjax: A Programming Language for Ajax Applications. Best Student Paper. 2009. 20 p.

10. *Терехов А. Н.* Технология программирования: учеб. пособие по специальности «Математ. обеспечение и администрирование информ. систем» — 010503 / Интернет-университет информ. технологий, 2006. 152 с.

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ЗАДАЧ ПРОГНОЗИРОВАНИЯ, УПРАВЛЕНИЯ И ОЦЕНКИ КАЧЕСТВА

А. Торегожин, аспирант СПбГУ, «Ланит-Терком»
(e-mail: arstan.toregozhin@lanit-tercom.com)

Д. Манаев, студент СПбГУ, «Ланит-Терком»
(e-mail: dmitry.manayev@gmail.com)

А. Золотухина, студентка СПбГУ (e-mail: alya.kir@mail.ru)

Ю. Самойлова, студентка СПбГУ (e-mail: yulia.samoylova@gmail.com)

Интеллектуальные системы на основе искусственных нейронных сетей (ИНС) позволяют с успехом решать проблемы распознавания образов, выполнения прогнозов, оптимизации, ассоциативной памяти и управления. Известны и иные, более традиционные подходы к решению этих проблем, однако они не обладают необходимой гибкостью за пределами ограниченных условий. ИНС дают многообещающие альтернативные решения, и многие приложения выигрывают от их использования [1].

На сегодняшний день наиболее мощными и перспективными инструментами, предоставляющими высокоуровневые средства для проектирования искусственных нейронных сетей, являются:

1) NeuroSolutions [2] — универсальный нейропакет фирмы NeuroDimension;
2) Neuroph [3] — проект с открытым исходным кодом, написанный на Java;
3) OpenCV [4] — проект с открытым исходным кодом, написанный на языке C++ и обеспечивающий основные функции нейронных сетей;
4) PWNLIB [5] — продукт отечественной компании «Павлин Технологии», обеспечивающий основные функции работы с нейронными сетями;
5) NNGLULIB [6] — продукт отечественной компании «Павлин Технологии», обеспечивающий ускорение расчета выхода многослойных нейронных сетей прямого распространения сигнала с применением графического процессора;
6) Neural Network Toolbox [7] — пакет дополнений к MATLAB [8] наиболее распространён и удобен для академических целей.

У перечисленных выше платформ существуют свои преимущества и недостатки. Neuroph является наиболее удобным в плане работы с нейронными сетями, так как включает в себя редактор визуального проектирования нейронных сетей, позволяющий проектировать нейронные сети намного быстрее. Преимуществом NeuroSolutions, NNGLUIB и OpenCV [9] является поддержка CUDA [10] (программно-аппаратная архитектура, позволяющая производить вычисления с использованием графических процессоров), что позволяет ускорять вычисления общего характера.

Основным недостатком всех вышеописанных инструментов является то, что они требуют слишком большого участия со стороны человека. От пользователя в большинстве случаев требуется выбрать тип нейронной сети, количество слоёв и нейронов [11], знание определённого языка программирования или специфики используемого инструмента.

Требуется построить механизм автоматизации процесса построения, обучения и настройки ИНС удовлетворяющий следующим нечётким требованиям:

1) автоматизация процесса построения архитектуры ИНС;
2) реализация удобного механизма настройки ИНС;
3) реализация механизма описания ИНС для сохранения полученной нейронной сети;

Прежде чем приступать к проектированию механизма, нужно выбрать платформу, на которой он будет реализован. Это связано с тем, что технологии ограничены в возможностях, и выбор той или иной технологии может накладывать ограничения на архитектуру самого инструмента. Для решения поставленной задачи технология должна удовлетворять следующим требованиям:

1) технология должна поддерживать C-подобные языки программирования. Данное требование обусловлено тем, что C-подобные языки являются самыми распространенными и быстрыми;
2) технология должна предоставлять удобный способ работы с многопоточностью. Данное требование обусловлено тем, что многопоточность крайне важна для ИНС [12];
3) технология должна предоставлять возможность создавать инструменты для мобильных устройств. Данное требование обусловлено тем, что в последнее время мобильные устройства получили широкое распространение, в связи с чем появилась потребность в программных ИНС-инструментах, работающих на мобильных устройствах [13].

Всем вышеописанным требованиям удовлетворяет технология iOS SDK [14]:

1) технология поддерживает языки C, C++, Objective-C;
2) iOS SDK предоставляет удобный высокоуровневый механизм работы с многопоточностью [15];
3) программы, написанные при помощи iOS SDK, работают как на мобильных устройствах, так и на персональных компьютерах.

Для поиска оптимальной архитектуры нейронной сети авторами был разработан механизм генерации нейронных сетей в зависимости от входных данных.

Основные особенности данного механизма:

1) генерация архитектуры нейронной сети;
2) создание и обучение нейронной сети по предоставленной архитектуре;
3) замер качества;

4) сохранение сгенерированной сети.

Алгоритм работы механизма показан на рисунке. На вход подаются три набора тестовых данных:

1) «обучающие» примеры — примеры для обучения нейронной сети;
2) «хорошие» тестовые примеры — примеры для тестирования нейронной сети;
3) «отрицательные» тестовые примеры — примеры, предназначенные для тестового прогона нейронной сети.

Качество каждой сгенерированная нейронная сети вычисляется с использованием описанной ниже метрики.

Сама метрика называется RMSE (Root Mean Squared Error, [16]) — общая ошибка нейронной сети и вычисляется по следующей формуле:

$$RMSE = \frac{\sqrt{\sum_{k=0}^{n}(O_k - P_k)}}{n},$$

где O_k — множество фактических значений нейронной сети, P_k — множество ожидаемых значений нейронной сети, n — количество примеров.

Данная метрика была взята из статистики и является наиболее распространённой для оценки качества нейронных сетей

Архитектура нейронной сети сохраняется в простом xml-формате её описывающем. Данный формат позволяет:

1) легко настраивать;
2) переносить полученную архитектуру вне зависимости от операционной системы;
3) легко внедрить или реализовать полученную нейронную сеть на любой другой технологии.

Разработанный механизм предоставляет большие возможности автоматизации, чем вышеописанные существующие реализации, нужно только предоставить три набора тестовых данных и запустить механизм, а всю остальную работу (построение, тестирование, подбор активационных функций, поиск оптимального числа нейронов в скрытых слоях, сбор статистики и сохранение) механизм позволяет автоматизировать.

В работе предложен высокоуровневый механизм автоматизации проектирования искусственных нейронных сетей со следующими особенностями:

1) автоматизация процесса построения архитектуры ИНС;
2) сохранение и настройка ИНС в xml-формате, что позволяет легко переносить и реализовывать архртектуру на других технологиях.

Сейчас идёт работа по созданию прототипа и проверки предложенного механизма на практике. В будущем для настройки архитектуры полученной ИНС планируется реализовать инструмент визуального редактирования. Результаты работы будут опубликованы.

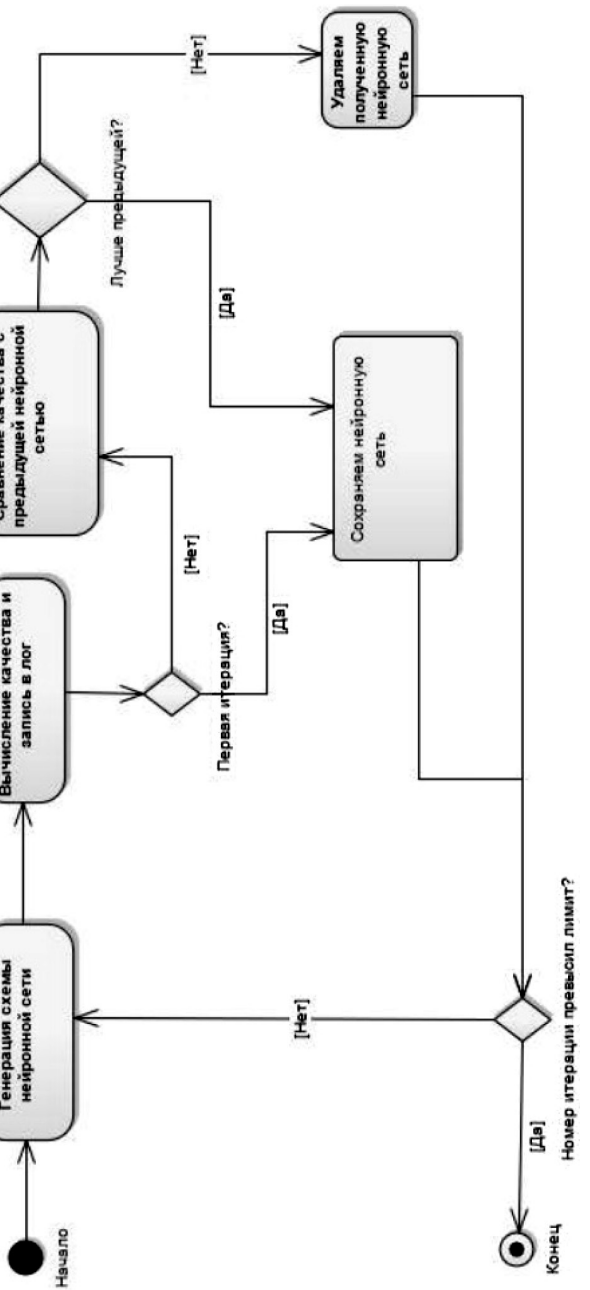


Рис. Схема динамической генерации нейронной сети по входным данным

Литература

1. *Anil K. Jain, Jianchang Mao, K.M. Mohiuddin.* Artificial Neural Networks: A Tutorial, Computer, Vol.29, No.3, March/1996, pp. 31–44.
2. http://www.neurosolutions.com/products/ns
3. http://neuroph.sourceforge.net/
4. http://ru.wikipedia.org/wiki/OpenCV
5. http://www.pawlin.ru/content/view/20/8/
6. http://www.pawlin.ru/content/view/19/8/
7. http://www.mathworks.com/
8. http://www.mathworks.com/help/toolbox/nnet/backpro2.html#34217
9. http://ru.wikipedia.org/wiki/CUDA
10. http://www.nvidia.ru/object/nvidia-for-opencv-press-20100923-ru.html
11. http://www.mathworks.com/help/toolbox/nnet/
12. *А. И. Галушкин.* Нейронные сети: основы теории.
13. *Kathryn Hymes, John Lewin.* OCR for Mobile Phones, Stanford Feb/2009.
14. http://developer.apple.com/devcenter/ios/index.action
15. http://developer.apple.com/library/ios/#documentation/General/Conceptual/ConcurrencyProgrammingGuide/Introduction/Introduction.html
16. http://www.math-interactive.com/

РАЗРАБОТКА СОРЕВНОВАНИЙ КАК ОБРАЗОВАТЕЛЬНЫЙ ПРОЦЕСС

Н. Н. Журавлев, И. В. Зеленчук, Д. В. Корнев
(e-mail: znick@hackerdom.ru, ilya@hackerdom.ru,
theshockwaverider@hackerdom.ru)

Уральский государственный университет им. А. М. Горького

На математико-механическом факультете Уральского государственного университета три раза в год проводятся соревнования по компьютерной безопасности по международным правилам CTF [1]. Это RuCTFE[2] — международные межвузовские соревнования по защите информации, и RuCTF[3] — всероссийские межвузовские соревнований по защите информации, которые проводятся в два этапа.

Для проведения подобных соревнований требуется подготовить два программных комплекса — это, игровой сервер и проверяющая система. Игровой сервер включает в себя несколько, обычно 5–8, игровых сервиса.

Проверяющая система может быть разработана и реализована один раз и незначительно изменяться от одного соревнования к другому. В случае с игровым образом все иначе. Он отдельно готовится на каждые соревнования и не повторяется. Игровой образ представляет из себя виртуальный сервер с несколькими игровыми сервисами. Каждый такой сервис представляет из себя сетевое приложение с определенным функционалом. Минимальный набор требований к такому игровому сервису:

- доступ по сети;
- возможность создания новых пользователей;
- аутентификация пользователей;
- каждый пользователь должен иметь возможность сохранить приватную, для него, информацию с последующим её просмотром.

По желанию, сервис может выполнять дополнительные функции. Примерами таких сервисов могут служить: FTP, SMTP, POP3, различные веб приложения и т. д.

Команда разработчиков, как правило, состоит из следующих людей:

- менеджер проекта (тимлид) — несет ответственность за планирование, подготовку и реализацию игрового образа. Также занимается организацией команды;
- администратор игровой инфраструктуры — планирует и разворачивает игровую сеть;
- администратор проверяющей системы — отвечает за настройку, запуск и мониторинг проверяющей системы;

- администратор игрового образа — собирает игровой образ и убеждается в его корректной работе;
- разработчик сервиса — разрабатывает и реализовывает игровой сервис, пишет документацию и тесты.

В зависимости от продолжительности срока подготовки игрового образа и масштаба соревнований размер команды разработчиков может колебаться от 8 до 20 человек. Дата соревнований объявляется заранее и ее очень сложно изменить. Разработчики соревнований должны планировать ведение разработки таким образом, чтобы она была завершена в срок. Немаловажным фактором является то, что по завершению разработки игровой образ проходит реальные испытания на соревнованиях. За время работы у участников появляются следующие навыки и компетенции:

1) работа в команде, совместное владение кода;
2) умение реализовывать согласованные с другими разработчиками интерфейсы;
3) умение интегрировать результаты работ нескольких разработчиков;
4) итеративная разработка с установкой и тестированием в конце итераций;
5) планирование разработки с жесткими сроками;
6) опыт доведения разработки до конечного продукта;
7) работа с трекером задач [4];
8) работа с системой контроля версий SVN[5];
9) документирование разработки.

Тимлид получает опыт работы в команде в специфических условиях. По ходу разработки он должен решать такие задачи как:

- мотивировать студентов бесплатно сделать качественный продукт;
- выявлять возможные проблемы на ранних стадиях разработки;
- перераспределение задач в случае выхода разработчика из процесса разработки;

Во время разработки и подготовки игрового образа команда разработчиков может опробовать различны подходы к разработке программного обеспечения: итеративная разработка[6], agile[7], парное программирование[8].

После каждого соревнования проводится полный разбор, что позволяет выявить недостатки того или иного подхода разработки, а также улучшить и закрепить удачные моменты.

По мнению команды разработчиков и организаторов подготовка таких соревнований дает ценный опыт для всех участников.

В Уральском государственном университете для подготовки подобных соревнований привлекается сборная команда из студентов разных курсов (2–5), магистрантов, аспирантов и молодых преподавателей. К примеру, на RuCTFE 2010 команда разработчиков состояла из 21 человека: из них 9 студентов, 6 магистрантов, 1 аспирант и 5 молодых преподавателей. В качестве тимлида выступал аспирант Дмитрий Корнев.

Литература

1. http://en.wikipedia.org/wiki/Capture_The_Flag#Computer_security
2. http://ructf.org/e/2010/
3. http://ructf.org/2011/
4. http://en.wikipedia.org/wiki/Bug_tracking_system
5. http://en.wikipedia.org/wiki/Apache_Subversion
6. http://en.wikipedia.org/wiki/Iterative_and_incremental_development
7. http://en.wikipedia.org/wiki/Agile_software_development
8. http://en.wikipedia.org/wiki/Pair_programming

СИМУЛЯТОР КОМПЬЮТЕРНОЙ СЕТИ

В. Самунь

Научный руководитель:
И. В. Зеленчук

Область применения симуляторов компьютерных сетей — обучение студентов основам компьютерных сетей, администрированию, также они могут применяться для апробации различных настроек в крупных сетях и последующего их внедрения на реальную сеть.

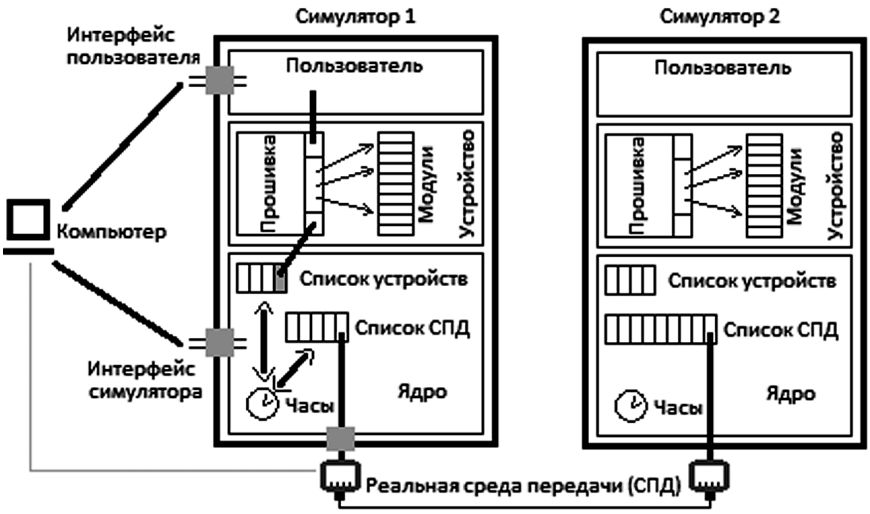
Известные симуляторы, такие как ns-3, GNS, Cisco Packet Tracer, omnet++ и другие имеют некоторые недостатки. Так, в ns-3 топология сети и действия на ней задаются программой на языке C++, топологию сети во время работы изменять нельзя. Симуляторы GNS, Packet Tracer жёстко завязаны на конкретное оборудование.

Мы поставили цель разработать симулятор, лишённый указанных недостатков. Преследуется реализация следующих возможностей:

- Динамическое изменение топологии сети.
- Снятие различной статистики.
- Детализированный просмотр передаваемых данных.
- Простое создание нового оборудования, интерфейсов, сред передачи данных.
- Возможность создания различных пользовательских интерфейсов настройки.
- Организация распределённой системы.

Архитектура симулятора — трёхуровневая. На самом нижнем уровне располагается *ядро* (оперирует сущностями «интерфейс», «среда передачи данных», «часы» и др.), на следующем уровне появляются устройства (хабы, свичи, роутеры, IP-телефоны и т. д.), на самом верхнем уровне располагается пользователь. Каждый уровень может взаимодействовать только с соседним уровнем. Более подробно архитектура изображена на рисунке.

Ядро симулятора — движок, работающий по парадигме часов. Всё время в симуляторе измеряется в тиках, которые генерирует тактовый генератор (*часы*). В процессе разработки ядра было предложено два способа реализации. Первый способ — запуск на каждом экземпляре устройства отдельный поток обработки событий. Второй способ — запуск одного потока («поток ядра»), обрабатывающего все события. Реализация первого подхода осложнена решением проблем синхронизации (несколько обработчиков могут использовать общие данные), также нужно каждый раз производить опрос всех потоков на завершение обработки очередного такта. Во втором подходе этих проблем не возникает, так как выполняется один поток и точно известен момент окончания тика. При реализации был выбран второй подход.



При таком подходе для организации распределённой системы нужно по завершении тика дождаться завершения тиков остальных узлов системы. Это можно реализовать выделением одного узла, который будет считать количество узлов, завершивших обработку тика и далее запускать обработку тика на всех узлах. Второй подход к реализации ядра и применение описанного подхода к распределению позволяет реализовать «гибридный» подход к реализации ядра, включающий преимущества первого и второго подходов.

Реальные устройства представляют собой «коробку» с интерфейсами и нечто, содержащее логику обработки сигналов, приходящих на интерфейс. В симуляторе эта логика реализуется в *модулях*, которые можно подключать в устройство. По сути, модуль реализует какой-то определённый протокол. Межмодульным взаимодействием и настройкой устройства занимается *прошивка*.

Для реализации был выбран язык C#. Основание для такого выбора — простота расширяемости симулятора (без необходимости перекомпиляции). Также, используя фреймворк NModel (nmodel.codeplex.com), можно при помощи введения разметки в код модуля проводить верификацию протокола, реализованного в модуле.

В данный момент реализовано ядро (возможности распределения пока нет), некоторые устройства и интерфейсы настройки. Для реализации различных сущностей имеются интерфейсы `IInterface`, `IBackbone`, `IDevice`, `IDeviceEngine`, `IModule` и их частичные реализации `InterfaceBase`, `BackboneBase`, `Wire`, `DeviceBase`.

Интерфейс настройки устройства состоит из интерфейса настройки прошивки устройства и интерфейса настройки каждого из модулей. При этом

прошивка сама определяет, как агрегировать настройку каждого модуля в свой интерфейс.

Используя перечисленные выше сущности, реализован консольный порт устройства. Для его реализации вводится абстрактный интерфейс «пользователь», который принимает сообщения с устройства ввода и передаёт их на консольный порт, и, принимая данные с консольного порта, выводит их на устройство вывода. Связка абстрактного интерфейса и консольного порта производится, как и в реальности, через консольный провод.

После реализации ядра было проведено нагрузочное тестирование. Для тестирования была сделана в симуляторе топология их трёх компьютеров и хаба. В компьютер была встроена специальная прошивка, которая при получении специального пакета немного модифицирует его и отправляет обратно. Ясно, что в такой сети будет наблюдаться «шторм» пакетов и реальная сеть через некоторое время перестанет работать (то есть, «забьётся» канал и буфера устройств). Тесты проводились на 32-битной архитектуре, процессор Intel Core2Duo T5600, объём памяти 2Gb, ОС Windows 7 Pro. Симулятор перестал работать (произошла ошибка выделения памяти) через 10 минут после запуска, совершив 23 цикла обработки пакетов размером в 25 байт (т. е. всего было обработано около 224 пакетов). Такой симулятор, как Cisco Packet Tracer перестаёт работать через 30 секунд в топологии из 10 маршрутизаторов с настроенным Frame Relay (хотя там «шторма» нет). С помощью несложных расчётов можно получить, что обработка одного пакета в нашем симуляторе заняла примерно 45 мкс, через каждую среду передачи было передано 349 Мб данных. При тестировании специально не обрабатывались коллизии при передаче пакетов, т. к. эта обработка только увеличит время до отказа работы симулятора. В буфере каждого устройства остались ожидать передачи примерно 9 000 000 пакетов.

СИСТЕМА ВИЗУАЛИЗАЦИИ RUCTF

В. Самунь

RuCTF — всероссийское межвузовское соревнование по защите информации. Подробно правила соревнования описаны на официальном сайте — http://ructf.org. Победителем соревнования объявляется команда, набравшая больше всего очков.

Очки складываются из следующих компонент:

1. Поддержка своих сервисов в рабочем состоянии;
2. Защита собственных флагов (флаг — некая приватная информация, хранимая в сервисе);
3. Отправка флагов, захваченных с сервисов команд-соперниц;
4. Отправка уведомлений о найденных уязвимостях;
5. Решение дополнительных заданий (тасков).

За всю работу по начислению баллов (а также проверки доступности сервисов, установки флагов и т. д.) отвечает проверяющая система. Информацию о баллах система хранит в базе данных. Участникам (а также болельщикам) интересно знать во время игры, сколько у какой команды очков по каждой из компонент начисления. Также необходимо как-то получать информацию о том, какие сервисы как работают у каждой из команд — работают корректно, с ошибкой или вообще недоступны.

Для предоставления этой информации проверяющая система время от времени сохраняет всю эту информацию в .xml-файл. Далее, при помощи XSLT-преобразования эта информация может быть показана наглядно. В настоящее время информация показывается в табличном виде — перечислены колонки с баллами, общий балл и информация по сервисам.

Данный способ предоставления информации показывает только то, что происходит в текущее время и не показывает такие интересные вещи, как захват флага. Захват флага пользователь может отследить только по изменению баллов в колонке «атака», но невозможно понять, с какой команды был захвачен флаг.

Чтобы устранить этот недостаток отображения состояния игры, была разработана система визуализации и несколько модифицирована проверяющая система. Новая версия проверяющей системы умеет предоставлять информацию о следующих событиях, происходящих во время игры:

- Начисление баллов;
- Изменение состояния сервиса;
- Сдача захваченного флага.

Предоставление информации реализуется через веб-сервис.

За наглядный показ этой информации отвечает визуализатор. Визуализатор является Silverlight-приложением.

Каждая команда отображается в визуализаторе в виде «цветочка», лепестками которого являются сервисы. Цвет и размер лепестков соответствует состоянию сервиса (работает, некорректно работает, не работает и т. д.). Атаки отображаются линиями, идущими от центра цветка (атакующая команда) к центру лепестка (атакуемая команда). Для наглядности цвет линии изменяется от зелёного к красному. Сервисы, с которых ещё не забрали ни одного флага, имеют чёрную границу. При первой же атаке на них граница убирается. Информация об очках и подробное описание состояний сервисов может быть получена из тултипов «цветка» и его лепестков.

Также отметим, что визуализатор имеет не только realtime-режим, в котором он автоматически забирает последние обновления и их отрисовывает, но и режим плеера, при котором можно просмотреть ход игры с любого момента времени.

При отрисовке есть проблема показа большого количества команд. Команды нужно показывать так, чтобы они не пересекались. Для решения этой проблемы было сделано 3 вида расположения «цветков» — по концентрическим окружностям, по концентрическим прямоугольникам, по заранее установленной сетке. Команды, которые становится невозможно разместить, не показываются. Это ограничение несущественно, т. к. существует возможность скрыть или возобновить показ некоторых команд.

Теперь опишем внутреннюю структуру визуализатора. Все действия визуализатор хранит в виде массива, данные в котором отсортированы по времени. Это позволяет быстро «перематывать» время в режиме плеера. Необходимо найти временну́ю позицию и произвести расчёт состояния сервисов, пробежавшись от найденной позиции назад.

Для получения данных визуализатор время от времени обращается к вебсервису (частота обращений настраивается на сервере, обычно выставляется в 10 секунд), сообщает ему время своего последнего обращения и в ответ получает список событий, произошедших с переданного времени последнего обращения. Далее полученный список присоединяется к уже имеющемуся и происходит отрисовка обновления.

ГРАФИЧЕСКАЯ ПОДСИСТЕМА ОПЕРАЦИОННОЙ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ EMBOX ДЛЯ РОБОТОВ LEGO MINDSTORMS

Д. Дзендзик

Санкт-Петербургский Государственный Университет, математико-механический факультет

Введение

В настоящие время по всему миру активно развивается робототехника. Роботов можно встретить уже в повседневной жизни, причем многие из них обладают хорошо развитыми мультимедийными возможностями, такими как обработка аудио- и видео-информации.

Простейшим примером робота является робот-конструктор LEGO Mindstorms. Несмотря на примитивность он оснащен встроенным экраном для вывода информации и умеет воспроизводить звук.

Embox — модульная операционная система реального времени для встроенных систем, разрабатываемая на математико-механическом факультете СПбГУ. ОС портирована на несколько платформ: SPARC, x86, Microblaze и ARM. Embox успешно запускается на роботах LEGO Mindstorms NXT 2.0.

Моя работа была направлена на создание графической подсистемы для ОСРВ Embox на платформе LEGO.

Постановка задачи

Основной модуль робота оснащён черно-белым экраном, звуковым динамиком и четырьмя кнопками.

Вывод на экран — самый простой и понятный вариант получения обратной связи. Он позволяет определить, что именно происходит в системе в данный момент или узнать о происшедшей ошибке. Это дает возможность, используя экран и кнопки, производить простейшую отладку не подключая робота к компьютеру.

Кроме того экран вместе с кнопками представляет собой эффективный способ управления роботом через простейший графический интерфейс пользователя.

Таким образом, реализация графической подсистемы очень важна. Требуемая система должна уметь выводить на экран как текст, так и графические объекты различной структуры. Следует также отметить, что платформа Lego Mindstorms сильно ограничена по ресурсам, поэтому реализация графической подсистемы не должна быть громоздкой.

Реализация

Описание экрана

Робот имеет дисплей размером 100 × 64 пикселя. Имеется некий буфер, который затем транслируется на экран. Буфер состоит из восьмиста вертикальных ячеек в 8 байт (1 × 8 пикселей). Экран в ячейках имеет размер 100 × 8. Ячейки располагаются сверху вниз и справа налево (рис. 1). В каждой ячейке хранится число от 0 до 255. В двоичном виде — восьмизначное число, «1» соответствует чёрному пикселю, «0» — белому. Числа представляются в шестнадцатеричном виде. Заполнение ячейки идет снизу вверх. Соответственно числу 0x40 (0100 0000 или 64) будет соответствовать одна чёрная точка, расположенная на второй снизу (соответственно 7 сверху) позиции (рис. 2, *а*), а числу 0x0E (1110 0111 или 231) будет соответствовать чёрная полоска с двумя белыми точками посередине (рис. 2, *б*).

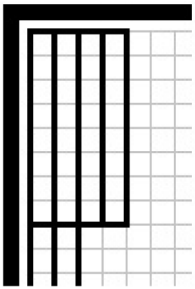


Рис. 1

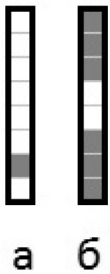


Рис. 2

Наличие буфера позволяет накапливать некоторые изменения и потом единовременно обновить дисплей робота. Возможность считывать значения ячеек из буфера позволяет проводить логические операции с записанными числами (например, инверсию: инверсия нулей и единиц соответствует инверсии цвета) и изменять произвольную область экрана независимо от того, скольким ячейкам или в каком месте ячейки находится эта область.

Шрифт. Вывод строк

Пяти ячеек достаточно, чтобы «нарисовать» букву английского алфавита, таким образом, каждая буква имеет размер 5 × 8 пикселей и представляется в виде массива из пяти чисел. Алфавит из 128 значений таблицы ASCII описан двумерным массивом размера 128 × 5, каждый элемент которого является массивом из пяти чисел (одна буква). Первая координата массива — это номер символа в таблице ASCII, вторая — номер ячейка в одном символе.

Например, английская буква «а» имеет вид: {0x20, 0x54, 0x54, 0x54, 0x78} (рис. 3). Имеется метод

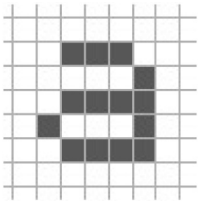


Рис. 3

void display_char(**int** c),

который принимает код ASCII символа и выводит его на экран робота, последовательно записывая в буфер соответствующие значения с проверкой границ экрана. После вывода символа выводится пустая ячейка — разделитель. Экран обновляется, когда символ полностью записан в буфер.

Вывод строки производится как последовательный вывод символов. Эту команду выполняет метод

void display_string(**const char** *str),

принимающий на вход строку. После вывода происходит перевод строки, т.е последующий вывод начнётся с новой строчки.

Также имеется метод, который выводит на экран первые 13 символов строки с отступом от левого края на 8 пикселей:

void tab_display(**const char** *str).

Если запустить его в цикле, то на экране появится список из 8 строк. Этот метод используется для вывода названия приложений в текстовом меню.

Рисование объектов

Для вывода на экран изображения используется метод:

int display_draw(uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint8_t *buff).

В качестве параметров он принимает:

- координаты **x** (в пикселях) и y (в ячейках — 8 пикселей на еденицу) — верхний левый угол (напомним, что координата y растёт сверху вниз);
- размер области — ширина **width** (измеряется в ячейках, т. е. по 8 пикселей на единицу), высота **height** (измеряется в пикселях) — оптимальным является соотношение 1x8 — квадратик 8x8 пикселей;
- **buff** — буфер с числовыми значениями, которые будут записаны в эту область.

Необходимо, чтобы размер буфера совпадал с шириной области. Следует отметить, что все параметры — исключительно целые положительные числа, не превышающие соответствующие им ограничения. Данный метод универсален — на экран можно вывести абсолютно любой «чёрно-белый» рисунок, размеры которого не превосходит размер экрана, но он достаточно трудоёмкий.

Для перекрашивания произвольной области в один цвет используется метод:

int display_fill(uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint8_t q)

В качестве параметров она принимает: координаты начала области — верхний левый угол; ширину и высоту; цвет: 0 — белый, 1 — черный. В этом методе все координаты и размеры измеряются в пикселях и не зависят от ячеек.

Метод

void display_clear_screen(**void**)

записывает нули во весь буфер экрана, т. о. происходит очистка экрана.

Примеры применения. Анализ и оценки эффективности

Метод выведения строк используется для вывода сообщений и любой текстовой информации.

1. Выведение логотипа

В системе Embox присутствует возможность вывести логотип системы на экран. При этом логотип опускается сверху, после чего буквы, входящие в него, поочерёдно «подпрыгивают» на месте. Для реализации анимации необходимо, чтобы между кадрами экран очищался, иначе происходит наслоение одного рисунка на другой и в результате на экране изображено нечто неосмысленное. Выведение логотипа было реализовано тремя способами.

При первом был использован метод рисования букв квадратиками с полным очищением и перерисовыванием экрана в промежутках между кадрами. Это операция, на которую уходит много времени, поэтому даже человеческим глазом визуально были видны задержки и мигания.

Второй метод использует метод перекраски области в один цвет. При этом становится возможно очищать (красить в белый цвет) только тот участок экрана, где происходит движение, оставляя неподвижные части без изменений. В этом случае обновление экрана происходит значительно быстрее.

В третьем случае был реализован вывод на экран логотипа только при помощи заливки области. В этом случае ожидаемо смена кадров происходит еще быстрее.

2. Текстовое меню

В системе Embox есть возможность отобразить на экране текстовое меню, содержащее список всех доступных для запуска тестовых примеров. Для отображения меню на экране был применен вывод текста с отступом. В этом случае отступ необходим для положения визуального указателя на текущее приложение. Визуальный указатель имеет вид большой точки и выво-

дится на экран с помощью метода рисования внутри квадрата. Эта же самая функция применяется для изменения положения указателя — удаление с его предыдущего положения.

Когда указатель смещается за пределы экрана происходит смещение списка отображаемых приложений в меню. Переход реализован двумя способами:

Первый просто перезаписывает строки, затирая название предыдущего теста названием нового. Данная операция проходит последовательно, поэтому заметны неприятные человеческому глазу мильтишения.

Второй способ очищает экран перед новой записью. В этом случае мильтишения не происходит, но процесс занимает некоторое время и визуально наблюдаем.

3. З м е й к а

Традиционным для системы Embox тестовым приложением для демонстрации работы на различных архитектурах является запуск классической игры «Змейка». При появлении поддержки дисплея на Lego Mindstorms мы смогли поиграть в эту игру прямо на роботе.

Возникшие проблемы

На данном этапе, метод заливки не работает в том случае, когда внутри одной ячейки есть некоторая небольшая область и мы хотим внутри этой области перекрасить область ещё меньшего размера. В этом случае перекрашивается полностью ячейка, включая точки, вне той области внутри которой мы красим (рис. 4. Есть *а*. Хотим *б*, получаем *в*). Описанная проблема решается путём считывания из буфера.

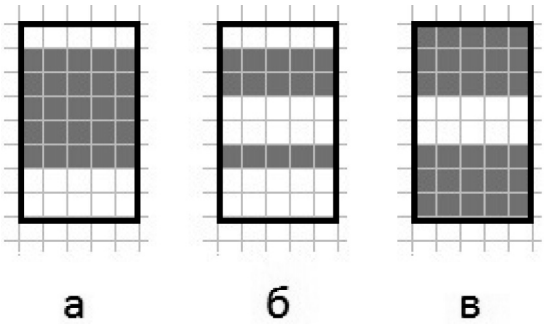


Рис. 4

Результаты

На сегодняшний день есть следующие результаты:

- Рассмотрена текущая реализация графической подсистемы LEGO.
- Реализован драйвер монитора для ОС Embox.
- Реализовано минимальное API для вывода текстовой и графической информации.
- Проведён сравнительный анализ способов вывода.
- Выявлены недостатки и способы их устранения.
- Реализованно визуальное текстовое меню.

Развитие графического интерфейса позволило визуализировать работу системы, упростить процесс отладки и тестирования ОСРВ Embox на платформе LEGO Mindstorms NXT 2.0.

Дальнейшее развитие

В ближайшее время планируется расширять API для вывода на экран: линий по координатам начала и конца и толщине, по толщине, углу наклона и координатам начала, окружность по координатам центра и радиуса. А также разбор файла в формате BMP и вывод его на экран.

Библиография

1. LEGO MINDSTORMS NXT Hardware Developer Kit.
2. LEGO MINDSTORMS NXT Software Developer Kit: http://mindstorms.lego.com/en-us/support/files/default.aspx
3. Классификация алгоритмов компьютерной графики: http://www.codenet.ru/progr/alg/alg.php

ОБЗОР ОПЕРАЦИОННЫХ СИСТЕМ ДЛЯ ПОСТРОЕНИЯ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

А. Бондарев

Аспирант кафедры системного программирования СПбГУ

Операционная система — комплекс управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между устройствами вычислительной системы и прикладными программами, а с другой стороны предназначены для управления устройствами, вычислительными процессами, эффективного распределения вычислительных ресурсов между вычислительными процессами и организации надёжных вычислений. Это определение применимо к большинству современных ОС общего назначения.

В ядро операционной системы обычно включают:

- планировщик;
- диспетчер памяти;
- драйверы устройств;
- сетевая подсистема;
- файловая система.

История операционных систем началась в 60-е гг. XX в., когда фирма IBM решила создать операционную систему для своих вычислительных комплексов. До этого момента каждая вычислительная машина по сути снабжалась собственной ОС. Но из-за трудностей и задержек в разработке, которые описаны в книге Брукса «Мифический человеко-месяц, или Как создаются программные системы», была создана не одна ОС, а целый их набор:

- OS/MFT для систем среднего класса. Она имела одного преемника, систему OS/VSI, развитие которой продолжалось до 1980-х;
- OS/MVT для крупных машин. Она была сходна с OS/MFT (программы могли переноситься между ними без перекомпилирования), но имела более продвинутое управление памятью и систему разделения времени, TSO. MVT имела несколько наследников, включая z/OS;
- DOS/360 для низших моделей System/360 имела несколько преемников, включая z/VSE, используемую до настоящего времени. Она значительно отличалась от OS/MFT и OS/MVT.

Таким образом, даже у одной фирмы возникли сложности при создании универсальной ОС, поскольку цели и ресурсы у различных вычислительных систем очень разнятся.

Начало созданию операционных систем для микроЭВМ положила ОС CP/M. Она была разработана в 1974 году, после чего была установлена на многих 8-разрядных машинах.

Успех системы в значительной степени был обусловлен ее предельной простотой и компактностью, возможностью быстрой настройки на различные конфигурации ПЭВМ. В рамках этой операционной системы было создано программное обеспечение значительного объема, включающее транслято-ры, тестовые редакторы, СУБД и так далее. Первая версия системы занимала всего 4 К.

Особую роль в истории создания операционных систем безусловно занимают различные UNIX системы.

Начальные версии операционной системы UNIX были разработаны в AT&T Bell Laboratories в конце 1960-х. Будучи абсолютно бесплатной в первых версиях и легко модифицируемой, эта система завоевала большую популярность. Так как UNIX была написана на языке высокого уровня Си, её можно легко было перенести на новую аппаратную архитектуру. Эта переносимость позволила ей стать основной системой для второго поколения миникомпьютеров и первого поколения рабочих станций.

Коллектив, создавший ОС UNIX, развил концепцию унификации объектов ОС, включив в исходную концепцию UNIX «устройство — это тоже файл» также и процессы, и любые другие системные, сетевые и прикладные сервисы, создав новую концепцию: «что угодно — это файл». Эта концепция стала одним из основных принципов ОС Plan9 (название было позаимствовано из фантастического триллера «План 9 из открытого космоса» Эдварда Вуда-младшего), призванной преодолеть принципиальные недостатки дизайна UNIX.

К 1990 году в рамках проекта GNU, основанного Ричардом Столлманом, был разработан набор программ, составивших инструментарий для разработки программ на языке Си: текстовый редактор Emacs, компилятор языка Си gcc, отладчик программ gdb, командная оболочка bash, библиотека стандартных вызовов. Все эти программы были написаны для POSIX-операционных систем и сейчас активно используются и развиваются.

А в 1991 году финский студент Линус Торвальдс загорелся идеей создания свободной UNIX подобной операционной системы. Он написал об этом в сети usenet на форуме MINIX и предложил разработать подобную ОС. В 1992 году версия ядра Linux достигла 0.95, а в 1994 году вышла версия 1.0. Сейчас на 90% суперкомпьютеров работают под одной из версий ОС Линукс.

О операционной системе MINIX необходимо сказать не только потому, что ее идеями увлекся упомянутый финский студент, а еще и потому, что изначально она применялась для демонстрации принципов построения надежных ядер операционных систем, описанных в книге Таненбаум Э., Вудхалл А. «Операционные системы: разработка и реализация». Таненбаум даже не принимал патчи для развития MINIX. Но начиная с третьей версии, анонсированной 24 октября 2005 года, MINIX был переработан и доведён до годности к использованию в качестве надёжной операционной системы для микроконтроллеров и других встраиваемых систем. Надежность ядра

в MINIX достигается за счет использования микроядерной архитектуры в отличие от монолитной, которая используется во всех вышеперечисленных ОС в том числе и Linux.

При разработке встроенных систем зачастую выдвигаются особые требования, в первую очередь это повышенная надежность. Еще одной особенностью для многих встроенных систем является работа в режиме реального времени. То есть выполнение заданного действия за строго определенный период времени. Необходимость таких систем была осознана еще в 60-е гг. XX в. И приблизительно во время создания первых версий UNIX, Digital Equipment Corporation создала простую операционную систему RT-11 для серии 16-битных машин PDP-11 (RT от Real time (в режиме реального времени)) — небольшую однопользовательскую операционную систему реального времени.

Особенности систем реального времени

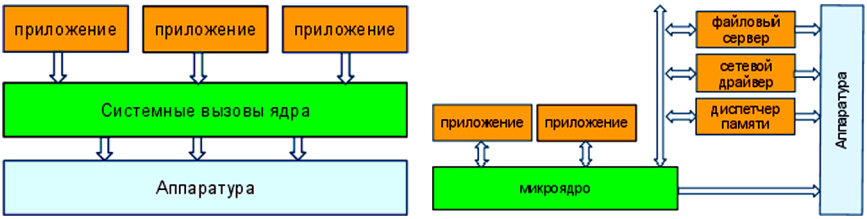
В качестве основного требования к системам реального времени выдвигается требование обеспечения **предсказуемости** или **детерминированности** поведения системы в наихудших внешних условиях, что резко отличается от требований к производительности и быстродействию и, следовательно, делает затруднительным применение в них универсальных ОС. Хорошая ОСРВ имеет предсказуемое поведение при всех сценариях системной загрузки (одновременные прерывания и выполнение потоков).

Мартин Тиммерман сформулировал следующие необходимые требования для ОСРВ:

- ОС должна быть многозадачной и допускающей вытеснение (preemptable);
- ОС должна обладать понятием приоритета для потоков;
- ОС должна поддерживать предсказуемые механизмы синхронизации;
- ОС должна обеспечивать механизм наследования приоритетов;
- поведение ОС должно быть известным и предсказуемым (задержки обработки прерываний, задержки переключения задач, задержки драйверов и т. д.); это значит, что во всех сценариях рабочей нагрузки системы должно быть определено максимальное время отклика.

Как показано выше операционные системы развиваются уже не один десяток лет, пройдя путь от монолитной архитектуры к клиент-серверной Последнюю принято называть микроядерной, и она чаще всего используется в современных системах реального времени. Поскольку традиционный монолитный (в том числе и модульный) подход имеет ряд недостатков:

- большой объем кода, следовательно трудно избежать ошибок и проверить систему на детерминируемость;
- невозможность распределить систему по различным процессорам;
- невозможно обращаться сквозь различные уровни защиты (например нужно напрямую к аппаратуре).



Построение систем реального времени на основе ОС общего назначения

Обычно в системе реального времени все же не нужно, чтобы все процессы обладали детерминированным временем реакции. Но необходимость наличия даже одного такого процесса вынуждает разработчиков этой системы продумать все варианты работы этого процесса в зависимости от внешних состояний и воздействий, а ОС на базе которой подобная система будет построена накладывает дополнительные требования на предоставляемые услуги.

Как уже было замечено, услуги должны быть следующими:

- предсказуемость времени переключения процессов;
- предсказуемость времени выделения памяти (в том числе и виртуальной);
- предсказуемым временем обработки прерываний.

В современных системах есть ряд дополнительных трудностей — это наличие кеша и наличие виртуальной памяти. С кешем все просто: поскольку необходимо рассчитывать максимальное время обработки, то кеш считается выключенным на момент работы задачи реального времени. С виртуальной памятью все несколько сложнее. Программист самостоятельно должен обеспечить наличие всех страниц, необходимых процессу реального времени, в TLB (table lookup buffer), — это по сути дела кеш для таблиц виртуальной памяти. Но в большинстве систем общего назначения, например, в ОС Linux, для этого необходимо модифицировать само ядро ОС. Что является совсем нетривиальной задачей.

То же самое касается времени переключения процессов и диспетчера динамической памяти. Для ядра ОС Linux есть специальные патчи, которые позволяют использовать планировщик реального времени, но это опять-таки модификация монолитного ядра. Что же касается динамически выделяемой памяти, то ее просто стараются не использовать, а реализуют собственные диспетчеры, заранее выделив память под максимальную нагрузку, если этого конечно не предоставляет само ядро.

Исходя из вышеизложенного, можно сказать что построить на ОС общего назначения, например, ОС Linux систему реального времени можно,

но это потребует гораздо больших затрат, чем например взять за основу какую нибудь ОСРВ. При этом наиболее дешевый способ создания задачи реального времени, заключается в расположении этой задачи в пространстве ядра, в виде обработчика прерываний. Но даже в этом случае необходимо заботится о состоянии таблиц виртуальной памяти и динамически распределяемой памяти для необходимых объектов.

Стандарты ОСРВ

Как уже отмечалось к встроенным и реалтайм системам предъявляются повышенные требования к безопасности. К тому же, поскольку во встроенных системах применяется большое количество разнообразных платформ (процессоров, микроконтроллеров), то остро стоит вопрос о стандартизации и сертификации ОСРВ.

Первым и самым старым стандартом для ОС был POSIX, в нем для систем реального времени определено расширение 1003.1b (Realtime extension). В нем определяются следующие требования для ОСВР:

- сигналы реального времени;
- планирование выполнения;
- таймеры, синхронный и асинхронный;
- ввод/вывод, ввод/вывод с приоритетами;
- синхронизация файлов;
- блокировка памяти;
- разделяемая память;
- передача сообщений;
- семафоры.

На данный момент данный стандарт сильно устарел и редко используется.

Ставшая фактическим международным стандартом ARINC 653 (Avionics Application Software Standard Interface), созданная группой авиапроизводителей, требует, чтобы ОСРВ обеспечивали многоуровневое разделение (виртуализацию) вычислительного времени, памяти и периферийных ресурсов между приложениями при работе на однопроцессорной плате.

Стандарт DO-178B радиотехнической комиссии по аэронавтике при Федеральной авиационной администрации США, который также планирует одобрить Пентагон, определяет последствия программных сбоев. Как минимум, сертифицированная система должна работать так, чтобы сбой в ней или в прикладном ПО не приводил к отказу всего комплекса.

Стандарт OSEK/VDX является комбинацией стандартов, которые изначально разрабатывались в двух отдельных консорциумах, впоследствии слившихся. OSEK берет свое название от немецкого акронима консорциума, в состав которого входили ведущие немецкие производители автомобилей: BMW, Bosch, Daimler Benz (теперь — Daimler Chrysler), Opel, Siemens и Volkswagen,

а также университет в Карлсруэ (Германия). Проект VDX (Vehicle Distributed eXecutive) развивался совместными усилиями французских компаний PSA и Renault. Команды OSEK и VDX слились в 1994 г.

В связи со стандартами для ОСРВ стоит отметить широко известный стандарт критериев оценки пригодности компьютерных систем (Trusted Computer System Evaluation Criteria — TCSEC) [DoD85]. Этот стандарт разработан Министерством обороны США и известен также под названием «Оранжевая книга» (Orange Book — из-за цвета обложки). Что касается Общих критериев, то в них введены похожие требования обеспечения безопасности в виде оценочных уровней (Evaluation Assurance Levels — EAL)

Стандарт EAL агентства национальной безопасности США задает стойкость алгоритмов шифрования информации и возможность хранения на одном ПК открытых и секретных данных. EAL делится на семь категорий (первая — решение полностью уязвимо, седьмая — максимальная защита). К ОСРВ обычно предъявляются наивысшие требования — EAL-7. Этот уровень предполагает формальную верификацию модели объекта.

Embox — набор для построения встроенных систем

Изначально проект Embox направлен на создание операционной системы для применения во встроенных системах. Он разрабатывался на кафедре системного программирования СПбГУ как проект с открытым исходным кодом. И применялся в том числе для обучения студентов в сфере разработки операционных систем и встроенных решений.

Ориентация на встроенные системы сказалась на архитектуре проекта в целом. Хотя ОС Embox и содержал обычный планировщик или диспетчер памяти, но уже на первых этапах проектирования разработчики задумались об ограниченных ресурсах встроенных систем, а также о необходимости стратифицировать и формально верифицировать код ОС и системы в целом. Поэтому в систему сборки была включена возможность тонкой настройки ОС под конкретную систему, модули ОС (в том числе и службы ядра) были изолированны друг от друга, и все статические зависимости обрабатывались на этапе сборки проекта. Такой подход позволил создавать конечные образы ОС порядка шести килобайт, иметь возможность отказываться от многозадачности, сетевой и файловых подсистем и так далее. В будущем этот же подход позволил описывать необходимые службы ядра (API) с помощью интерфейсов и даже задавать конкретную реализацию интерфейсов в системе.

На втором этапе проект был ориентирован на адаптацию наработок на платформу Lego Mindstorms. Это было сделано в сотрудничестве с кафедрой теоретической кибернетики СПбГУ. Перенос был осуществлен успешно, но в систему потребовалось добавить свойства работы в реальном времени. Поскольку робототехника — одна из тех областей, где необходимо решать задачи реального времени.

Таким образом, на сегодняшний день Embox представляет из себя конфигурируемую и кросс-платформенную ОСРВ

В ее состав входит ядро ОС, обеспечивающее несколько вариантов планирования задач, различные диспетчеры динамической памяти, первичные обработчики прерываний, в том числе и вложенных. Сетевую и файловые подсистемы. Набор служебных утилит и библиотек.

Все это позволяет строить на основе Embox не только задачи для встроенных систем на основе обычных POSIX вызовов, но и создавать в процессы реального времени на основе расширения API ядра, таких как посылка событий, легких потоков, пулов объектов и так далее.

Вывод

В статье рассмотрены различные ОС и показаны причины того, что операционные системы продолжают свое развитие. Особенно это заметно в сфере встроенных систем, которые обычно обладают ограниченными ресурсами и зачастую выполняют критические по времени задачи, то есть должны обладать предсказуемым временем реакции системы, что является трудно достижимым с использованием систем общего назначения.

В статье также рассмотрена ОСРВ Embox, разрабатываемая на кафедре системного программирования для различных задач реального времени, в том числе и задач робототехники. Данная ОС в силу своей конфигурируемости позволяет добиться заданных требований к ОС по временным характеристикам, требуемым ресурсам и пользовательским свойствам (наличие стека протоколов файловой системы и так далее).

Литература

1. *Бурдонов И. Б., Косачев А. С., Пономаренко В. Н.* Операционные системы реального времени // ИСП РАН. Препринт 14. М., 2006.

2. *Таненбаум Э.* Современные операционные системы. 2-е изд. СПб.: Питер, 2007. 1038 с.

3. *Таненбаум Э., Вудхалл А.* Операционные системы: разработка и реализация. СПб.: Питер, 2007. 704 с.

FLASH ФАЙЛОВАЯ СИСТЕМА ОСРВ EMBOX

А. Батюков

Санкт-Петербургский государственный университет

Введение

Разработка встроенных систем (embedded systems) — особая область системного программирования, направленная на разработку программно-аппаратных комплексов разного уровня. Она активно развивается в последнее время в связи с ростом применения микрокомпьютеров/микроконтроллеров в различных устройствах.

Одной из ключевых особенностей этой области является направленность на эффективное решение конкретных задач, зачастую реального времени, в условиях ограниченного числа аппаратных ресурсов.

Эффективного решения задачи разработки встраиваемых систем можно добиться лишь при акивном сотрудничестве и взаимодействии как программистов, так и инженеров, ищущих пути решения задачи на стыке программирования и проектирования аппаратуры.

Таким образом, с точки зрения системного программиста разработка встраиваемых систем сегодня представляет собой совершенно особую область, в которой зачастую бывают неприменимы классические подходы решения тех или иных задач.

Flash-память

Системное ПО, созданное программистом загружается на аппаратуру и хранится там постоянно. Оно представляет собой вообще говоря код системного загрузчика и (возможно) операционной системы, в которой реализованы управлюящие алгоритмы Кроме того оно должно оставлять возможность конфигурирования под конкретный вид используемой аппаратуры.

Код загрузчика, ОС и конфигурационные файлы хранятся прямо на аппаратуре в энергонезависимой памяти. Стандартом де-факто во встраиваемых системах стало использоваие flash-памяти на основе NAND или NOR ячеек. Эта память обладает определенными достоинствами и недостатками, которые перечислены ниже:

- энергонезависимость;
- механическая стойкость;
- высокая скорость чтения;
- малый физический размер;
- дешевизна;

- недолговечность;
- невысокая скорость перезаписи.

Следует отметить, что flash-память используется особым образом. В готовом изделии она нужна в основном для считывания информации, в то время как во время разработки она активно используется для загрузки и тестирования различных прошивок и конфигураций устройства.

Обзор используемых файловых систем

Все вышесказанное позволяет задуматься о том, что для эффективной разработки системному программисту был бы удобно использовать некие высокоуровневые механизмы работы с памятью. В классическом программировании такой высокоуровневый и удобный для прикладного программиста интерфейс работы с памятью представляют файловые системы. Они могут сильно отличаться между собой, быть в разной мере приспособлены для решения различных задач. Давайте подробней познакомимся с некоторыми из них.

Интерфейс разделов жесткого диска

Раздел — часть долговременной памяти жесткого диска, выделенная для удобства работы и состоящая из смежных блоков.[2] На других носителях информации выделение разделов не предусмотрено и почти не практикуется.

Достоинства:

- на одном жестком диске можно хранить информацию о разных файловых системах;
- можно отделить информацию пользователя от информации ОС;
- можно установить несколько ОС.

Информация о размещении разделов на жестком диске хранится в таблице разделов, которая является частью главной загрузочной записи (MBR), располагающейся в первом физическом секторе жесткого диска.

Интерфейс разделов жесткого диска может быть перенесен на flash-накопители, но не предназначен изначально для работы с ними, что создает ряд трудностей, таких как быстрый износ flash-ячеек. Кроме того он в достаточной степени низкоуровневый.

FAT16/FAT32

Иерархическая файловая система, каталоги могут вкладываться друг в друга на произвольную глубину. Запись и чтение происходят блоками, информация о блоках принадлежащих файлу хранится в специальной табли-

це размещения файлов, часть из которой находится в оперативной памяти, а часть на диске.[2]

Достоинства:

- повсеместная поддержка;
- сравнительно быстрая работа с flash-памятью [5].

Файловая система семейства FAT может быть использована для flash-накопителей, но приводит к сильному износу ячеек flash-памяти, отведенных под таблицу размещения разделов. Это ограничивает область ее применения только теми flash-накопителями, у которых контроллер аппаратно поддерживает алгоритм циклической перезаписи блоков.

Linux: виртуальная файловая система

Виртуальная файловая система ОС Linux — высокоуровневая абстракция, представляющая единый интерфейс доступа ко всем совместимым файловым системам более низкого уровня.[4]

Виртуальная ФС очень удобна в использовании и является необходимой частью современной ФС.

Linux: ext2/ext3/ext4

Основная файловая система ядра ОС Linux, подключается к виртуальной файловой системе. Отсутствие журналируемости в ext2 позволяет использовать ее для работы с flash- накопителями. Не оптимизирована для работы с flash-памятью. Запись и чтение происходят блоками, информация о блоках хранится в индексной таблице. Ведется резервирование важных частей ФС.[3, 9]

Достоинства:

1. очень высокая скорость работы для жестких дисков;
2. надежность хранения данных;
3. открытость.

Файловые системы семейства ext при определенной профилировке могут быть использованы для flash-накопителей, но являются универсальным, а значит слишком тяжеловесным, решением, неприменимым для встраиваемых систем.

Linux: jffs/jffs2

Современная flash-ориентированная (NOR и NAND) файловая система, подключается к виртуальной файловой системе. Реализует алгоритм цикли-

ческой записи блоков, что увеличивает скорость работы и уменьшает износ ячеек flash памяти. Поддерживает журналирование операций с файловой системой. [6]

Достоинства:

- flash-ориентированная;
- журналируемая.

Часто используется для загрузки различных конфигурационных файлов в ОС Linux, когда эта ОС используется во встраиваемых системах некритичных ко времени. Пожалуй, являются лучшим решением для традиционного использования flash-накопителей. Не ориентированы на разработку встраиваемых систем. По некоторым исследованиям слишком требовательны к объему оперативной памяти. [8]

Contiki: coffee filesystem

В рамках проекта contiki идет разработка ОС для встраиваемых систем. В нем реализована flash-ориентированная файловая система coffee. Она достаточно проста в использовании и относительно невелика по размеру [7].

Достоинства:

- flash-ориентированная;
- интуитивно понятный интерфейс.

Легковесность этой файловой системы позволяет использовать ее во встраиваемых устройствах сильно ограниченных по ресурсам. ФС coffee реализует только необходимый минимум функциональности работы с файлами. Практически не расширяема, так как архитектура системы максимально упрощена в угоду минимизации использования ресурсов.

UFFS

Появилась как результат разработки ультра легкой flash-ориентированной ФС для встраиваемых систем (An ultra lowcost flash file system for embedded system).

Достоинства:

- экстремальная легковесность;
- быстрое монтирование ФС;
- статическое распределение всей памяти при загрузке;
- может работать без ОС.

Особенности этой файловой системы, по заявлениям разработчиков, позволяют использовать ее даже в задачах, требующих «мягкого» реального

времени. [8] Как и coffee FS практически нерасширяема из-за минимализтичности. Кроме того представляет чересчур низкоуровневый интерфейс «страниц» вместо привычного интерфейса файлов.

Кроме рассмотренных существует еще большое количество как файловых систем общего назначения (таких как NTFS, btrfs, raiserfs/raiser4, ISO 9660) [9], так и flash-ориентированных файловых систем (например, FFS/FFS2, ETFS, ExtremeFFS, YAFFS) [10]. Они ничем не хуже, а в чем-то даже лучше рассмотренных. Ограниченный объем статьи не позволяет рассмотреть их все. Выбирая, какие же из них описать, я руководствовался желаниями описать базовые принципы и дать как можно более широкий обзор тенденций развития ФС. Кроме того предпочтение отводилось описанию ФС, которые оказали влияние на написание ФС ОСРВ Embox.

Постановка задачи

Задача создания собственной файловой системы возникла перед нами в рамках развития проекта создания ОСРВ Embox.[1] Ключевой особенностью этой системы является модульность и конфигурируемость, что позволяет применять ее для широчайшего класса встраиваемых устройств. Кроме того исторически сложилось, что одним из направлений развития системы является создание на ее основе инструментального средства для разработки встраиваемых систем.

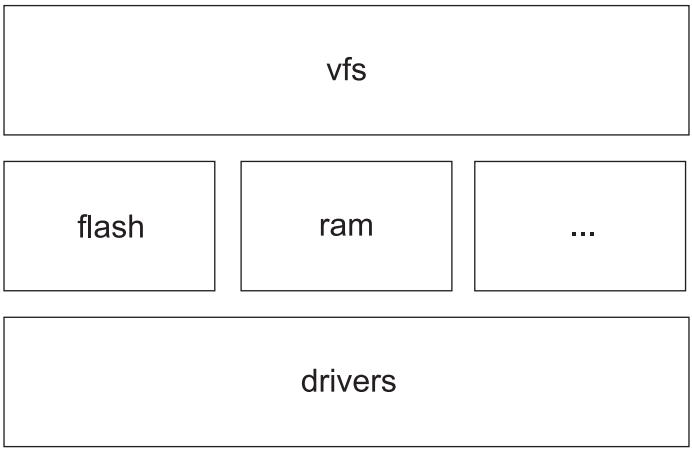
Ни одна из рассмотренных выше файловых систем не подходила в полной мере, так как не соответствовала выдвигаемым к системе требованиям:

- универсальный интерфейс (по возможности POSIX) доступа к файлам;
- ориентированность на flash-память;
- возможность работы с ram памятью и возможность дальнейшего расширения системы;
- нетяжеловесность и достаточная скорость работы;
- удобство для применения на этапе разработки встраиваемых систем.

Теоретическая часть

Общая структура файловой системы

Файловая система организована в три уровня. На верхнем уровне находится виртуальная файловая система, в которой реализован общий интерфейс работы с файлами. Уровнем ниже располагаются драйверы файловых систем, реализующие функции интерфейса верхнего уровня в зависимости от особенности той или иной конкретной файловой системы. На самом нижнем слое находится реализация аппаратно-зависимых драйверов.



Такая организация файловой системы с одной стороны предоставляет универсальный интерфейс доступа к файлам, находящимся на различных носителях, а с другой стороны не громоздка, а значит может быть использоваться для встраиваемых систем.

Алгоритмы уменьшения износа flash-ячеек

Для уменьшения износа ячеек flash-памяти используется алгоритм трансляции реальных «физических» адресов в доступные через стандартизованный интерфейс «логические» адреса. С одной стороны, такое преобразование позволяет работать с накопителем в рамках распространенных стандартов без применения специализированного ПО. С другой стороны, оно призвано обеспечить прозрачный для пользователя дефект-менеджмент и выравнивание «износа» ячеек памяти, обладающих ограниченным ресурсом: при необходимости перезаписи ячейки информация из нее переносится в свободную ячейку памяти с наименьшим износом.

Алгоритмы перепрограммирования flash-памяти

На этапе разработки встраиваемых систем существует необходимость частой перезаписи во flash-память тестируемого образа системы. Для того чтобы это было возможно без использования дополнительных перепрограмматоров специфичные функции драйверов, отвечающие непосредственно за запись во flash-память, помещаются в отдельную релацируемую секцию. Это позволяет, загрузив код образа в оперативную память через ethernet кабель по tftp протоколу, прошить его во flash-память.

Realtime возможности

В системах реального времени традиционно отсутствует понятие файловой системы, потому что работа с файлами на диске невозможна в realtime режиме. Работа в системах реального времени ведется в оперативной памяти, за исключением того, что код системы может исполняться из flash-памяти напрямую. Это возможно за счет того, что время чтения из flash-памяти не зависит от физического расположения ячейки.

Но в оперативной памяти системы реального времени было бы удобно иметь интерфейс файлов. Реализован драйвер файловой системы ram, подключаемый к виртуальной файловой системе. С одной стороны доступ к памяти через интерфейс файловой системы не влияет на realtime, а с другой стороны делает доступ к файлам единообразным, не зависящим от их расположения во flash или оперативной памяти.

Практическая часть

Виртуальная (логическая) файловая система

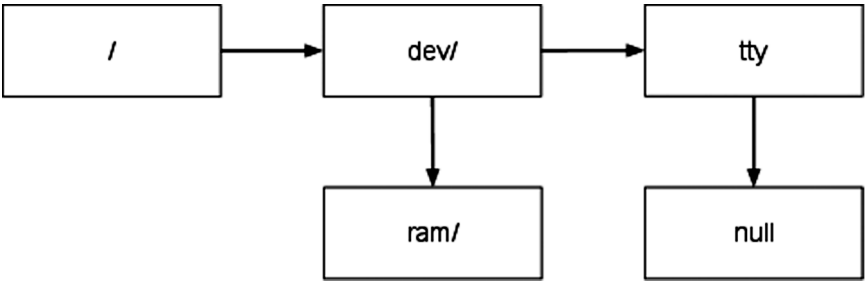
Каждый файл в системе представляет собой узел (node) виртуальной файловой системы. Все узлы организованы в иерархию, общая структура которой — дерево, в корне которого лежит узел /. Узел описывается специальной структурой

```
typedef struct node {
        const char          name[CONFIG_MAX_LENGTH_FILE_NAME];
        void                                    *file_info;
        file_system_driver_t        *fs_type;
        struct list_head            neighbors;
        struct list_head            leaves;
} node_t;
```

Элементы структуры имеют следующее предназначение:

- *name* — имя узла в системе;
- *file_info* — некая внутренняя информация о файле;
- *fs_type* — указатель на драйвер файловой системы, обслуживающей этот узел;
- *neighbors* — структура, содержащая информацию о «соседях» того же уровня;
- *leaves* — структура, содержащая информацию о «детях» узла.

Пример организации файловой системы



Драйвер файловой системы описывается структурой:

```
typedef struct file_system_driver {
        const char                  *name;
        const file_operations_t     *file_op;
        const fsop_desc_t           *fsop;
} file_system_driver_t;
```

Элементы структуры:

- name — имя драйвера файловой системы;
- file_op — указатель на структуру, описывающую операции, которые могут быть произведены с файлом;
- fsop — указатель на структуру, описывающую операции, которые могут быть произведены с файловой системой.

Для того, чтобы добавить поддержку нового драйвера файловой системы, необходимо реализовать все необходимые функции, после чего зарегистрировать драйвер в виртуальной файловой системе с помощью макроса DECLARE_FILE_SYSTEM_DRIVER, который добавит описание драйвера в специальную секцию. Таким образом, мы статически можем задать все необходимые параметры драйвера файловой системы.

Подключаемые драйвера файловых систем

В файловой системе проекта Embox реализованы два драйвера файловых систем: flashfs и ramfs. Каждый из них реализует необходимые функции интерфейса виртуальной файловой системы vfs.

Особенности реализации flashfs

Представляет собой реализацию операций с файлами, находящимися на блочном устройстве. Каждый блок представляет собой структуру со следующими основными полями:

- логический адрес блока;
- физический адрес блока;
- степень износа блока;
- указатели на следующий и предыдущий логические блоки.

Для отслеживания занятости блока и его степени износа используется карта физических адресов блоков. Для идентификации файлов в системе и получения информации о их расположении во flash-памяти используется карта файлов с указателями на логический адрес начального блока каждого из них.

Особенности реализации ramfs

Представляет собой реализацию операций с файлами, находящимися в линейной адресном пространстве.

В момент создания узла виртуальной файловой системы в описывающую его структуру вносится информация о том, драйвер какой файловой системы используется для операций с этим узлом.

Драйвера устройств

На нижнем уровне файловой системы располагаются драйвера устройств flash памяти. На данный момент в проекте Embox поддерживаются NOR микросхемы flash-памяти Intel серии P30.

Заключение

Описанная файловая система удовлетворяет выдвинутым требованиям. Она предоставляет универсальный интерфейс доступа к файлам, поддерживает работу с flash и ram памятью. Простота принципов построения делает ее небольшой по объему и в то же время приемлемой для использования. В данный момент ведутся работы по ее улучшению, в первую очередь в области быстродействия и соответствия стандарту POSIX.

В соответствие с принципами ОСРВ Embox все части файловой системы организованы в виде отдельных независимых модулей, что позволяет сконфигурировать ее в минимально необходимой в каждом конкретном случае варианте.

Файловая система ОСРВ Embox нашла свое применение в проектах компании ЗАО «Ланит-Терком» по разработке встраиваемых систем, в том числе решающих realtime задачи робототехники.

Литература

1. http://code.google.com/p/embox/wiki/FileSystemDescription
2. *Э. Таненбаум* «Современные операционные системы». 2-е изд. СПб.: Питер, 2007. 1038 с.: ил.
3. http://en.wikipedia.org
4. http://www.ibm.com/developerworks/ru/library/l-virtual-filesystem-switch/index.html
5. http://www.testfreaks.com/blog/information/usb-flash-drive-comparison-part-2-fat32-vs-ntfs-vs-exfat
6. http://sourceware.org/jffs2/jffs2-html/
7. http://www.sics.se/contiki/wiki/index.php/Coffee_Filesystem_Guide
8. http://sites.google.com/site/gouffs/home
9. http://www.ibm.com/developerworks/ru/library/l-linux-filesystem
10. http://www.ibm.com/developerworks/ru/library/l-flash-filesystems

УДАЛЕННОЕ УПРАВЛЕНИЕ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Г. Д. Ефимов, Р. С. Одеров, И. Л. Киряновский

Санкт-Петербургский государственный университет

Введение

Сегодня на рынке множество устройств, которые должны управляться в режиме реального времени, например роботы. Пользователи хотели бы иметь возможность управлять ими удалено. Для этого уже существуют различные протоколы, начиная от Bluetooth и заканчивая GSM, UMTS, CDMA, а в будущем LTE связью. Но подобные протоколы не ориентированы на передачу данных в реальном режиме времени, то есть с гарантированной задержкой. С другой стороны ширина и качество каналов позволяют обеспечить гарантированную доставку данных к устройству и обратно (хотя бы в пределах одного сегмента сети). Таким образом для реализации механизма управления устройством в реальном времени необходимо реализовать стек протоколов, который бы обеспечил минимальное время доставки определенного сорта пакетов. В TCP/IP подобную проблему пытаются решать с помощью введения качества обслуживания QoS. Стандартным методом добавления качества обслуживания является вставка в пакет на физическом уровне соответствующих тегов, показывающих в том числе и тип обслуживания. Но к сожалению не все производители сетевого оборудования и разработчики стеков поддерживаю данный метод.

Наша команда занимается разработкой операционной системы Embox, которая в том числе может применяться в роботах на платформе Lego NXT. При работе с данной платформой мы столкнулись с необходимостью управления роботизированными устройствами в режиме реального времени.

Проблема

Рассмотрим возникающие при разработке проблемы на примерах:

1. ОСРВ установлена на роботе Lego, и пользователь захотел управлять им на расстоянии, используя коммуникатор (планшет и т. п.). Для этого, например, установим связь Bluetooth между роботом и мобильным устройством. Используя кнопки направлений или гироскоп, можно управлять роботом, однако возникает проблема: ОСРВ робота должна быстро реагировать на приходящие от пользователя команды. В системах реального времени не допускаются задержки реакций системы, так как это может привести к:

- утрате актуальности результатов;
- материальным потерям;
- авариям и катастрофам.

2. Допустим мы устроили гоночные соревнования роботов. У нас есть сеть Wi-Fi, к которой подключаются многочисленные участники заездов. Но управлять роботом вслепую не удастся, поэтому на нем имеется видеокамера, передающая получаемое изображение мобильному устройству. Первая опасность заключается в том, что главный сервер должен пропускать большие объемы информации (в данном случае — потоковое необработанное видео в приемлемом для нас качестве). В добавление к этому здесь ясно обозначается проблема, связанная с обратной связью: большие объёмы потокового видео должны быстро обрабатываться, упаковываться и отправляться по каналу связи на нетбук/коммуникатор пользователя. И при всем этом необходимо уложиться в максимально наименьшее время.

Таким образом в роботах, роботизированных и других встроенных системах, требующих гарантированное время реакции, которые мы рассматриваем, необходимо иметь встроенный стек, обладающий предсказуемыми характеристиками доставки пакетов, как детерминированное время, и по возможности оно должно быть наименьшим.

Обзор существующих стеков

Нами был произведен обзор существующих решений данной проблемы. В итоге было выделено несколько реализаций сетевых стеков, в частности стек Linux и стек lightweight IP (lwIP). Тщательно проанализировав их структуру, можно выделить для нас некоторые наиболее важные особенности:

Стек BSD (Linux)

Плюсы:

1. Обработка пакетов производится не в контексте прерывания;
2. При разборе пакеты не копируются, а всего лишь происходит передача ссылки на пакет;
3. Поддержка большого числа протоколов.

Минусы:

1. Большой объем кода, трудно проверить на ошибки;
2. Монолитный, протоколы включаются все сразу;
3. Сложный и многослойный классификатор пакетов, пакеты передаются через весь стек протоколов последовательно, и невозможно передать пакет напрямую к пользователю.

4. Не детерминированное время доставки: в зависимости от уровня протокола и загрузки сокета оно будет меняться.

Стек lwIP

Плюсы:

1. Простой и маленький код
2. Возможность работы с небольшими устройствами
3. Простой классификатор пакетов

Минусы:

1. Входящие пакеты обрабатываются сразу при получении, что не дает возможности без ожидания переходить к работе со следующим пакетом.

Решение

Таким образом, в качестве решения разумно было бы совместить плюсы стеков Linux и lwIP, избежав описанных выше недостатков. В качестве решения мы можем предложить создание собственного сетевого стека, при реализации которого можно достигнуть наименьших задержек. Основные концепции нашей архитектуры будут следующие:

- Стек с минимальной задержкой: сначала поставить пакет в очередь, а потом разобрать его отдельно. Таким образом снижается время на обработку;
- Пакет не копируется, а происходит передача ссылок на него. Соответственно избегается громоздкая процедура копирования пакета. (Например, если мы хотим передавать изображение с камеры робота на сервер даже размером 240x320 нам понадобится 106 пакета размером 15 байт, итого на кадр около 160 килобайт и к тому же для нормального управления роботом необходимо по-крайней мере 14 кадров в секунду. В итоге получаем примерно 2,2 мегабайта в секунду с камеры, что приведет к значительным задержкам при постоянном дублировании пакетов);
- Маленький классификатор пакетов: при передаче данных исполнять поддержку исключительно нужных протоколов. (Например, при управлении роботом с помощью коммуникатора передаем только самые нужные и, по возможности, наименьшие по объему данные, чтобы соблюдать real-time режим);
- Конфигурируемый классификатор пакетов: возможность прокидывать необходимые пакеты напрямую к пользователю, минуя стек, и, следовательно, экономя время на обработку пакета;
- Возможно создание собственных протоколов, максимально подходящих для наших задач. Тогда также имеет смысл присвоить нашему пакету приоритет выше, чем у остальных, тем самым он будет практически сразу проскакивать стек и уходить на обработку. Это значительно увеличит скорость обработки интересующих нас пакетов.

На данный момент в проекте Embox реализация стека протоколов TCP/IP поддерживает следующие необходимые части:

- конфигурируемый классификатор пакетов, можно выбирать из уже реализованных протоколов (ARP, ICMP, UDP...);
- обработка (классификация) пакетов происходит не в контексте прерываний;
- выделение памяти под пакет может происходить с помощью аллокатора реального времени;
- осуществлена поддержка bluetooth стека.

Выводы

В статье рассмотрены проблемы, возникающие при необходимости удаленного управления устройствами в реальном времени. Проведен обзор существующих реализаций стеков, выявлены их недостатки и преимущества. В настоящее время реализация стека Embox не до конца подходит для решения данной задачи, необходимо добавить следующие части:

- избежать громоздкой процедуры копирования при обработке пакета;
- возможность прокидывать пакеты напрямую, минуя стек;
- создать в итоге свой протокол передачи данных для систем реального времен.

Развитие данного стека является достаточно перспективной, и в результате он может стать очень эффективным для класса задач, где требуется удаленное управление системами в реальном времени.

Литература

1. http://www.tml.tkk.fi/Opinnot/Tik-110.551/1999/papers/08IEEE802.1QosInMAC/qos.html#chap3.1

2. http://www.connect.ru/article.asp?id=2563

3. http://code.google.com/p/embox/

СИСТЕМА СБОРКИ ОСРВ EMBOX

Э. Абусалимов

студент 5-го курса кафедры Системного программирования
Математико-механического факультета
Санкт-Петербургского государственного университета

Аннотация

Рассмотрены основные подходы к автоматизации сборки приложения из исходных кодов, сформулированы требования при разработке встроенных систем, проведен обзор существующих средств. Описаны возможности разработанной нами системы сборки Embuild.

Введение

Одним из неизбежных этапов разработки программного обеспечения является сборка результирующего продукта, готового к эксплуатации или отладке, из исходных кодов. Сборка также может включать в себя такие этапы как:

- Выполнение тестов.
- Подготовка сопроводительной документации.
- Создание отчетов о результатах сборки.

Как правило, используются специальные средства, в той или иной степени автоматизирующие этот процесс, — от простых скриптов, вызывающих команды в нужном порядке, до сложных систем, способных самостоятельно выявлять необходимые шаги и порядок сборки.

Особенности при разработке встроенных систем

Разработка встроенной системы в первую очередь означает узкую специфику решаемых задач. Программное обеспечение должно обеспечивать лишь необходимую функциональность, требуемую от системы в целом. Это обусловлено несколькими факторами:

- Зачастую аппаратное обеспечение устройства обладает ограниченными ресурсами, что сказывается на разработке программного обеспечения и ведет к требованию «ничего лишнего».
- Помимо аппаратных ограничений, минималистичность исходных кодов требуется также и при прохождении сертификации.
- И наконец, с ростом объема кода, растет и количество ошибок в нем.

Разумеется, можно для каждого нового устройства создавать программное обеспечение «с нуля». Но во-первых, такой подход ведет к чрезмерному удорожанию стоимости разработки, а во-вторых, в новом коде неизбежно появление новых ошибок.

Гораздо более привлекательно выглядит разработка универсальной системы, содержащей базовые блоки и способной с высокой точностью конфигурироваться для решения конкретной задачи. Таким образом, к системе сборки предъявляется ряд дополнительных требований.

- Возможность построения приложения из взаимосвязанных компонент.
- Поддержка множества конфигураций приложения.
- Проверка непротиворечивости построенной конфигурации.
- Функция получения «среза» лишь тех исходных кодов, которые действительно участвуют в сборке для конкретной конфигурации.

Существующие средства

GNU make

Система сборки make [1] является стандартом де-факто в мире Unix-подобных систем. make автоматизирует процесс сборки на основе информации, предоставленной разработчиков в т. н. Makefile'ах. В Makefile'ах описываются шаги сборки на специальном логическом языке, позволяющем задавать зависимости файлов друг от друга (цели и реквизиты), а также правила удовлетворения этих зависимостей (рецепты).

К примеру, Makefile для сборки приложения *baz*, состоящего из двух исходных файлов *foo.c* и *bar.c*, можно записать следующим образом:

```
baz : foo.o bar.o
      ld -o $@ $^

foo.o bar.o : %.o : %.c
      cc -o $@ -c $<
```

Этот Makefile говорит бувально следующее: приложение *baz* — это результат компоновки утилитой `ld` объектных файлов *foo.o* и *bar.o*, которые в свою очередь получаются при компиляции *foo.c* и *bar.c* с помощью `cc` соответственно.

Граф зависимостей для описанной сборки выглядит следующим образом:

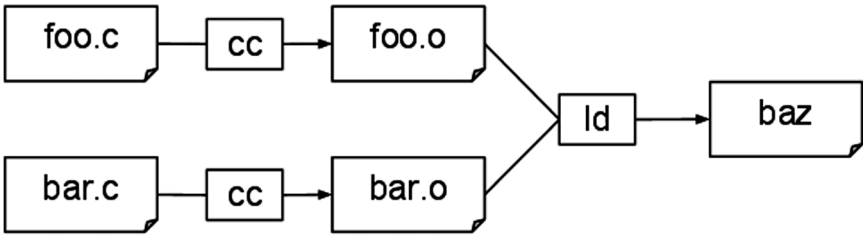


Рис. 1. Граф зависимостей, используемый системой сборки make

В момент сборки make анализирует граф зависимостей и на основе информации о времени последнего изменения каждого файла определяет и запускает необходимые программы. Таким образом, make обеспечивает простой механизм инкрементальной сборки — при изменении нескольких реквизитов выполняется лишь минимальный набор рецептов, необходимый для получения цели.

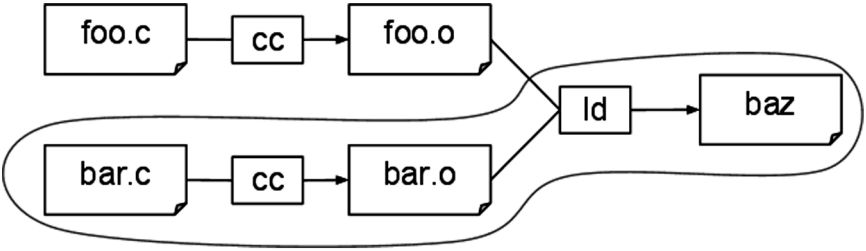


Рис. 2. Механизм инкрементальной сборки make. При изменении лишь файла *bar.c* производится только компиляция *bar.o* и компоновка цели *baz*

Использование этого механизма существенно сокращает время сборки при активной разработке, когда часто изменяется небольшой набор файлов исходных кодов, избавляя от необходимости каждый раз совершать полную сборку. Особенно это актуально для крупных проектов, для которых время полной сборки может достигать нескольких часов.

Помимо базовой функциональности, большинство реализаций make (в том числе GNU make) поддерживает возможность параллельной сборки на многопроцессорных компьютерах, что также актуально для больших приложений.

Таким образом, make — это универсальное средство, позволяющее описывать шаги сборки на гибком скриптовом языке Makefile'ов.

Однако такая гибкость влечет за собой некоторые недостатки. Основной из них заключается в том факте, что Makefile'ы целиком пишутся самими разработчиками. Помимо отсутствия общепринятого стиля кодирования, это также неизбежно приводит к ошибкам в скриптах. Поскольку средства отладки make достаточно бедны, обнаружить такие ошибки как правило весьма трудно. Кроме того, с ростом и развитием проекта необходимо постоянно поддерживать Makefile'ы в актуальном состоянии, что также лежит на плечах разработчика.

С учетом этих недостатков, в чистом виде make используется лишь для малых проектов или только на ранних стадиях развития. В большинстве же случаев используются либо специальные средства генерации Makefile'ов (такие как GNU Automake [2]), либо сложные скрипты make, обеспечивающие готовую инфраструктуру для сборки, (например, Kbuild [3]). И тот и другой способ дает возможность создавать описания на языке более высокого уровня.

К недостаткам этих систем следует отнести:

- Большинство из них требуют установки на ПК разработчика дополнительного ПО, что иногда влечет проблемы, когда для разработки используются ПК под управлением разных операционных систем.
- Сложность управления конфигурациями. Зачастую эти системы ограничиваются возможностью установки констант вида CONFIG_XXX, задающих набор файлов исходных кодов и правила условной компиляции. Все это приводит к т. н. «#ifdef nightmare» и влечет ухудшение читабельности исходного кода, создавая проблемы при сертификации.

Проблема

При создании программного обеспечения с высокой степенью конфигурируемости необходимо учитывать семантическую связь между различным частями системы, которую не отслеживает Make. Например, одна часть системы может не функционировать без другой.

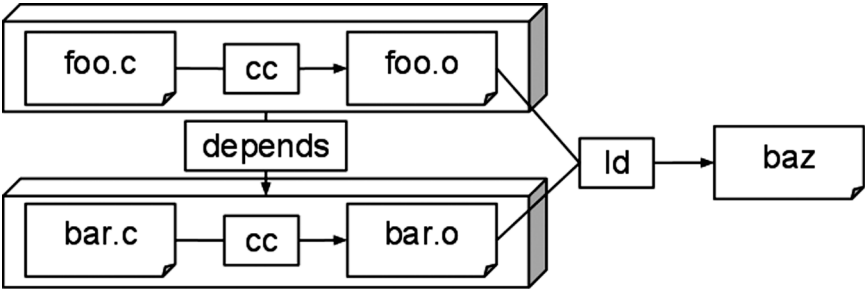


Рис. 3. Модель, учитывающая семантические зависимости между компонентами приложения. Модуль *foo* не может функционировать без модуля *bar*

Кроме того, при использовании классического подхода возникают следующие проблемы:

- Сложно составить вручную такую конфигурацию системы, которая бы обладала непротиворечивостью, обеспечивая успешную компиляцию и корректное функционирование приложения
- Поскольку Makefile'ы создаются разработчиком вручную, необходимо поддерживать их в актуальном состоянии при добавлении новых исходников или изменении существующих

Предлагаемое решение

В рамках одного из проектов ЗАО «Ланит-Терком» нами была спроектирована и реализована система сборки Embuild, которая нашла свое применение в проекте с открытым кодом по разработке ОСРВ Embox [4]. Этот проект

разрабатывается силами сотрудников ЗАО «Ланит-Терком» и позиционирует себя как высококонфигурируемая инструментальная система для применения на всех этапах разработки встраиваемых систем.

Embuild обладает следующими преимуществами.

Кросс-платформенность

Embuild реализован на языке GNU make — наиболее распространенном диалекте make. Это означает, что от разработчика, использующего систему сборки Embuild при создании приложения, равно как и от пользователя этого приложения не требуется установки дополнительного ПО, поскольку GNU make по умолчанию входит в поставку большинства систем семейства GNU/Linux и в Unix-окружения разработки для Microsoft Windows (такие как Cygwin и MinGW).

Высокий уровень описаний

Вся система, использующая инфраструктуру Embuild, рассматривается как набор взаимосвязанных модулей, что дает возможность строить систему «из кубиков».

```
define module foo
brief = The main application module
sources = foo.c
depends = bar
endef
```

Каждый модуль является самостоятельной сущностью со своим набором файлов исходных кодов, документацией и зависимостями от других модулей. Поскольку при конфигурировании конечной системы разработчик мыслит в терминах целостных модулей, а не отдельных файлов, это упрощает понимание всей системы и снижает порог вхождения.

Ориентированность на модульную структуру

Модульная структура позволяет собирать систему, нацеленную на решение конкретной задачи, будь то система для низкоуровневого тестирования оборудования или прикладное ПО с поддержкой плагинов. Особенно это актуально в сфере встроенных систем, где большое значение имеют аппаратные ограничения.

Как уже говорилось, в проектировании подобных систем важную роль имеют семантические межмодульные зависимости. Система сборки позволяет автоматически разрешать эти зависимости:

(1) На этапе компиляции при включении в сборку одного модуля вместе с ним подключается весь подграф его зависимостей, гарантируя успешную компоновку итогового образа.
(2) Во время исполнения гарантируется правильный порядок запуска и останова модулей приложения опять же за счет отслеживания зависимостей между ними.

Верифицируемость образа

В процессе сборки образа строится граф зависимостей и проверяются ограничения для каждого из модулей, это позволяет создавать верифицируемое ПО, которое гарантирует включение нужных модулей в систему с требуемыми версиями этих модулей и заданными параметрами. Для упрощения процесса граф зависимостей может быть представлен в графическом виде с помощью утилиты Graphviz.

Облегчение сертификации

Система сборки позволяет получить дистрибутив (срез), содержащий только те исходники, которые участвуют в сборке. Наличие такого дистрибутива является необходимым условием для прохождения сертификации.

Заключение

Разработанные нами средства дают возможность создавать гибкие системы, надежные во всех возможных конфигурациях, готовые к сертификации исходного кода. Кроме того, описанные технологии позволяют извлекать и структурировать знания о приложении, тем самым облегчая понимание сложных систем.

Литература

1. GNU ‘make’ manual, http://www.gnu.org/software/make/manual/make.html
2. GNU Automake manual, http://sources.redhat.com/automake/automake.html
3. The Evolution of the Linux Build System — Bram Adams, Kris De Schutter, Herman Tromp, Wolfgang De Meuter, http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/115/119
4. Embox — Essential toolbox for embedded development, http://code.google.com/p/embox

ДИНАМИЧЕСКОЕ ВЫДЕЛЕНИЕ ПАМЯТИ В ОПЕРАЦИОННЫХ СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Д. А. Зубаревич

Санкт-Петербургский государственный университет

Память — один из важнейших ресурсов, требующий рационального управления. Основная цель части операционной системы, отвечающей за распределение динамической памяти, применять и удачно комбинировать наиболее эффективные алгоритмы управления памятью. Давать им оценку необходимо по следующим основным критериям:

1. Область применения алгоритма, т. е. рассмотрение как тех ситуаций, для которых алгоритм будет наиболее удачен, так и тех, для которых алгоритм плохо подходит;
2. Время на выделение подходящего блока памяти.
3. Время освобождения ненужного более блока памяти.
4. Фрагментарность памяти и ее влияние на скорость работы.

В последние время широкое распространение получают **операционные системы реального времени** (ОСРВ) , например, в промышленном производстве, робототехнике , военной и других областях. Основным критерием для таких ОС служит, прежде всего, скорость работы, которая позволила бы обеспечить выполнение операций без каких либо задержек.

Касаясь вопроса динамического распределения памяти в ОСРВ, становится очевидным, что для ОС такого типа важнейшим критерием эффективности алгоритмов управления памятью служит время, затрачиваемое на выделение и освобождение.

Задумываясь о том, как ускорить процесс выделения и освобождения памяти, стоит вспомнить о том, что может появиться необходимость выделения памяти для большого количества однотипных объектов. Примером тому могут послужить сетевые пакеты. Это наблюдение приводит к идее **менеджера объектов**.

Под объекты выделяется массив памяти из максимального количества объектов данного типа, и содержащиеся в нем объекты связываются некоторой структурой данных, например списком свободных объектов. Тогда выделение объекта — это всего лишь возвращение ссылки на голову списка, а освобождение объекта — это помещение ссылки на него в конец очереди свободных объектов. Очевидно, это как нельзя лучше подходит для ОСРВ, так как позволяет обойти временные затраты на поиск наиболее подходящего куска памяти.

В качестве доказательства достаточно привести временные результаты следующего теста. В первом случае происходит выделение с помощью си-

стемной функции malloc множества однотипных объектов. Во втором случае однократно выделяется пул для всех объектов сразу, и создается список этих объектов. После этого в программе объекты выделяются из этого списка. Во втором случае затраченное время будет существенно ниже.

Естественно, что писать для каждого типа объектов собственный менеджер было бы весьма накладно. Эта проблема решается с помощью создания некоторого общего интерфейса для выделения и освобождения **различных** однотипных объектов. Для каждого типа этих объектов создается свой пул, а в системе создается кеш подобных пулов для различных зарегистрированных объектов. Такой метод управления памятью называется **slab allocator**.

Таким образом применение метода **slab allocator для** динамического распределения памяти соответствует основной концепции ОСРВ, позволяя ускорить выделение и освобождение памяти для большого количества однотипных объектов.

Литература

1. *Таненбаум Э.* Современные операционные системы. 3-е изд.
2. http://code.google.com/p/embox/

ПРИОРИТЕТНАЯ МОДЕЛЬ ПЛАНИРОВАНИЯ И НАСЛЕДОВАНИЕ ПРИОРИТЕТОВ ПОТОКОВ В ОСРВ EMBOX

А. Крамар

Санкт-Петербургский Государственный Университет

Введение

В последнее время наблюдается ужесточение требований к операционным системам реального времени. Это связано с тем, что для существующих задач реакция системы должна быть более быстрой и детерминированной. Более того, нередки случаи, когда перед ОСРВ встает сразу несколько задач, выполнить которые надо должным образом, то есть в жестко заданный промежуток времени. В связи с этим многопоточность — одно из основных требований к операционным системам в целом.

Задача — это набор инструкций (поток), в операционных системах реального времени каждый из которых должен обладать приоритетом. Тем самым, задавая различный приоритет, мы можем определить нити, процессорное время которым следует отдавать в первую очередь, а которым при отсутствии более важных задач. Исходя из этого, в современных ОСРВ планировщики используют приоритетную модель.

Например, в ОСРВ Embox эта модель реализована следующим образом:

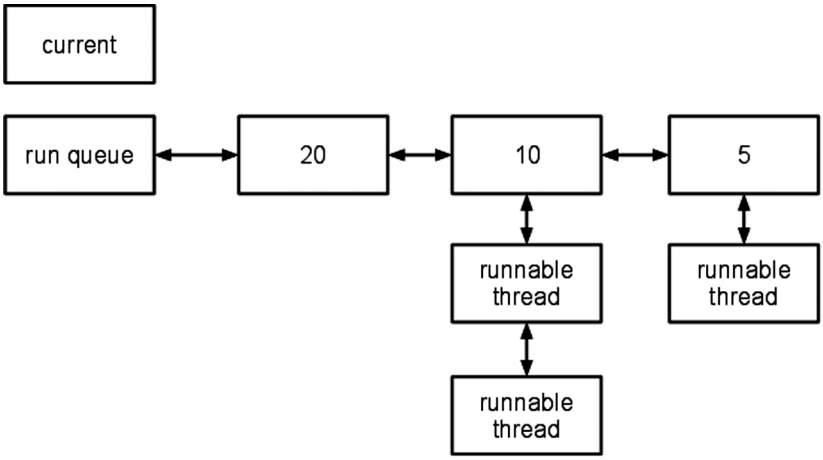


Рис. 1

В планировщике хранятся только готовые к исполнению потоки. Потоков, находящихся в ожидании или по тем или иным причинам не запущенных там нет. Сразу после старта, поток добавляется в очередь (*run queue*,

см. рис. 1). Исполняемый в данный момент поток (*current*, см. рис. 1) в этой очереди не хранится, но в ней хранится ссылка на список потоков с тем же приоритетом. Например, если поток с приоритетом текущего больше нет, то список пуст. Очередь — это двусвязный список списков потоков с одинаковым приоритетом (тоже своего рода очередь, где потоки упорядочены по времени добавления в планировщик). Она упорядочена по убыванию приоритета. Тем самым поиск наиболее приоритетного из готовых к исполнению происходит очень быстро (взять первый поток их первого списка). Удаление из планировщика тоже не занимает много времени. Более наглядно изображено на рис. 1. Тут исполняется поток с приоритетом 20. Соответствующий ему список в очереди пуст, так как он один с таким приоритетом и все остальные потоки хранятся упорядоченно.

Таким образом ОСРВ Embox поддерживает многопоточность и операции планировщика происходят за конечное и предсказуемое время.

В многопоточной среде часто возникают проблемы, связанные с использованием различными потоками одних и тех же ресурсов (данных или устройств). Для решения подобных проблем используются такие методы взаимодействия потоков, как взаимоисключения (мьютексы), семафоры, критические секции и события. Их использование может породить некоторые неудобства, ошибки и т. п.

Возникновение инверсии

Использование мьютексов может существенно замедлить исполнение потока, так как тот ресурс, который он защищает в данный момент может использоваться только одним потоком. Потоки, желающие заполучить мьютекс образуют очереди ожидающий его освобождения. Это может отложить исполнение на неопределенное время, что неприемлемо для ОСРВ.

Ситуация, когда по тем или иным причинам выполняются потоки с приоритетом ниже ожидаемого или требуемого называется инверсией приоритетов. В настоящее время планировщики достаточно умны, чтобы не допускать таких ошибок. Сейчас чаще всего причиной возникновения инверсии приоритетов является принятие наиболее значимой нити неготовой к исполнению в тот момент, когда разработчик ожидает исполнения этого потока. Иногда причиной объявление потока неготовым к исполнению становиться невозможность захватить мьютекс. Иными словами, если мьютекс был занят другим потоком до того, как он понадобился более приоритетному потоку, а мьютекс еще не освобожден.

К примеру, это может произойти, когда первостепенной задачей является обработка сообщений, находящихся в определенной очереди. И если сообщение было передано другому потоку (с более низким приоритетом), то исполнение первостепенного потока может быть отложено. Приоритетный поток в таком случае объявляется неготовым к исполнению из-за отсутствия

сообщения для обработки. В этой ситуации разработчик будет недоволен тем, что данное сообщение попало другому обработчику. Это простой пример инверсии приоритетов.

Классическим сценарием развития инверсии приоритетов с использованием мьютексов является пример, показанный на рис. 2.

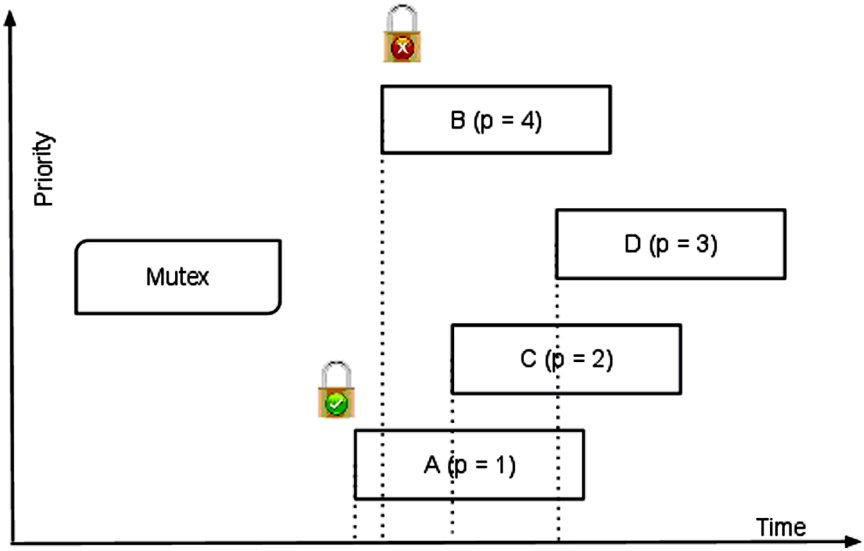


Рис. 2

Здесь мьютекс защищает некий ресурс от одновременного использования потоками *A* и *B*. Если один поток захватил мьютекс, то другой засыпает до момента освобождения.

В этом примере изначально готова к исполнению нить с низким ($p=1$) приоритетом (*A*). В той или иной точке ей потребовался ресурс, защищенный мьютексом. Так как других задач пока нет, и мьютекс свободен, то *A* получает ресурс и продолжает выполняться. Чуть позже нить с большим ($p=4$) приоритетом (*B*) становится готовой к исполнению, начинает выполняться и доходит до момента, когда ей требуются те же ресурсы, что и первой нити. Так как в данный момент мьютекс занят, поток *B* засыпает в ожидании освобождения ресурса потоком *A*. После этого нить *C*(*B*) со средним ($p=2$) приоритетом по независимым причинам становится готовой к запуску. Поток *C* не заинтересован в ресурсе, поэтому он ничего не делает для получения мьютекса. Но так как приоритет *C* выше приоритета исполняемого (*A*), то планировщик запустит ее сразу. Получаем ситуацию, что нить *C* работает, хоть имеет приоритет ниже, чем у потока *B*. Это и называется инверсией приоритетов. Поток *B* не может зайти в мьютекс, потому что ждет, пока завершится поток

A, которого вытеснил поток *C*. Ситуация может ухудшится путем неограниченного вытеснения потоков (к примеру нить *D* с приоритетом, чуть большим ($p=3$) чем у потока *C*, вытеснит этот поток и т. д.) Это называется цепочкой блокировок. Исполнение *B* может отложиться на непредсказуемое время.

Рассмотрим на рис. 3 более подробно, что происходит с мьютексом (снизу), и как распределяется процессорное время между потоками на примере трех нитей (*A*, *B* и *C* с приоритетами из предыдущего примера). Мы видим, что наиболее приоритетный поток исполняется в последнюю очередь из-за ожидания освобождения мьютекса наименее приоритетным потоком, вытесненным в свою очередь потоком со средним приоритетом.

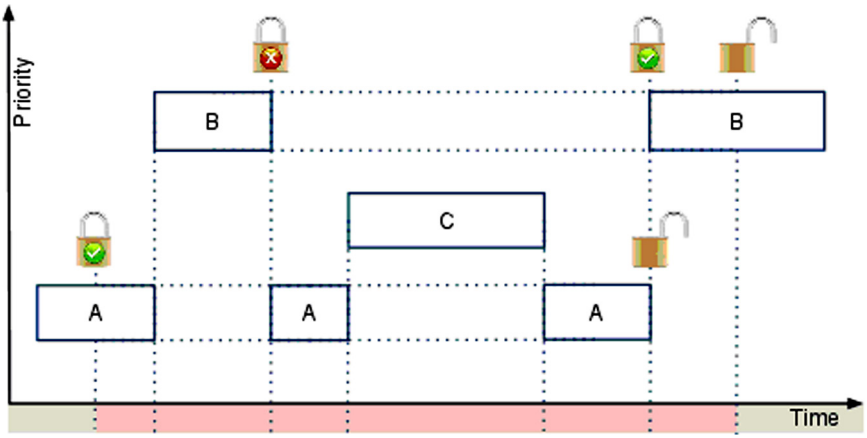


Рис. 3

Вероятность происхождения этого явления настолько мала, что некоторые разработчики, зная о такой проблеме умышленно ее не решают. Но в случае происхождения инверсии, последствия могут быть непредсказуемыми.

Явление инверсии приоритетов может иметь серьезные последствия в критическом режиме реального времени для встроенных систем. В частности, приоритетная задача может быть отложена для выполнения на неожиданно долгий срок. Эти задержки могут стать причиной несоблюдения временный сроков (что недопустимо для ОСРВ), пробелы в сборе данных и т. п.

Существующие решения проблемы

1. Игнорирование проблемы

Игнорирование проблемы инверсии приоритетов может привезти к пагубным последствиям, злоумышленной инверсии (искусственное поднятие разработчиком ранга низкоприоритетных задач), но если система не является

СРВ, то это действенный метод, так как нет затрат ресурсов для решения проблемы и переключение потоков идет естественным путем.

2. «Потолок» приоритета для мьютекса

Одним из решений задач является выставление для мьютекса максимального приоритета, так называемого «потолка». Он определяется как максимум из приоритетов существующих потоков. При таком решении, если поток заблокируется и станет исполняться поток в низшим приоритетом, ему искусственно поднимут приоритет до приоритета-«потолка» и он завершится, так как его никто не вытеснит. Это действенно, если не станет готовым к иполнению поток с приоритетом, большим «потолка». Тогда поток с искусственно поднятым приоритетом будет вытеснен и исполнение передастся новому. Таким образом, в случае возникновение более приоритетной задачи этот метод не всегда будет работать.

3. Наследование приоритетов

Еще одним методом является наследование приоритетов. В этом случае потоку, завладевшему ресурсом присваивается приоритет равный максимальному из приоритетов потоков, этот ресурс ожидающих.

Основной идеей метода является игнорирование изначально заданных приоритетов во время выполнения критических секций. Когда поток завладевает ресурсом, он исполняется на самом высоком приоритетном уровне. В результате этого, ни один поток, имеющий приоритет ниже максимума не сможет вытеснить поток, захвативший критическую секцию. После завершения потока приоритеты возвращаются к исходным и планирование идет по прежнему сценарию. Таким образом поток, завладевший ресурсом, всегда имеет наивысший приоритет из потоков, ждущих этот ресурс, тем самым решается не только проблема неограниченной инверсии, но и проблема взаимных блокировок.

Более подробно на рис. 4.

Здесь рассматривается случай, показанный на рис. 2, но с использованием наследования приоритетов. Сначала происходит то же, что и в предыдущем примере (см. рис. 2). Но в момент попытки Потока *B* захватить мьютекс, система понимает, что приоритет потока, занявшего этот мьютекс меньше приоритета ожидающего. Поэтому потоку *A* присваивается приоритет потока *B*, так как он на данный момент максимальный. Поток *A* освободит мьютекс раньше, так как не будет вытесненным никаким другим потоком. После освобождения потоку *A* возвращается его изначальный приоритет. После освобождения мьютекса происходит перепланирование. Потоку *B* разрешен захват мьютекса и он продолжает выполяться. Тем самым мы получили ожидаемое: приоритетная задача исполнится раньше.

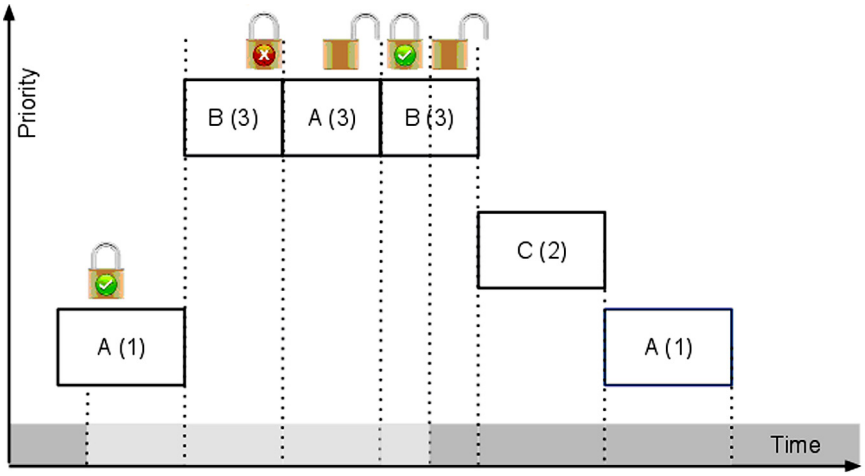


Рис. 4

Характеристика механизма наследования приоритетов

Так как операционная система реального времени должна обладать понятием приоритета для потоков и обеспечивать механизм наследования приоритетов, то в ОСРВ Embox следует реализовать третий способ решения проблемы инверсии приоритетов.

Но так же, как и у любого решения у него есть свои недостатки. Одним из таких недостатков является возможность заведения системы в тупик круговыми зависимостями. Так же не исключается возможность злоумышленного тупика, когда разработчик формирует цепи наследования таким образом, чтобы отложить исполнение приоритетной задачи на долгий, хоть и конечный, срок.

Преимуществами наследования приоритетов является относительная простота реализации (в отличие от второго метода) и в то же время решение проблемы (в отличие от игнорирования).

Выводы

Благодаря введению механизма наследования приоритетов в ОСРВ становятся редкими явления инверсии приоритетов и взаимных блокировок. Тем самым предсказать поведение потоков и системы в целом становится проще. Система может гарантировать выполнение наиболее приоритетного потока из готовых к исполнению в ожидаемое конечное время. Более того, так как на данный момент ОСРВ Embox не поддерживает POSIX потоки,

то введение наследования приоритетов позволит их применять в задачах реального времени.

Литература

1. *Бурдонов И. Б., Косачев А. С., Пономаренко В. Н.* Операционные системы реального времени // ИСП РАН. Препринт 14. М., 2006.

2. *D. Kalinsky.* “Mutexes Battle Priority Inversions”. http://www.kalinskyassociates.com/Wpaper2.html

СИСТЕМА СОБЫТИЙ В ОС РЕАЛЬНОГО ВРЕМЕНИ ДЛЯ ВСТРОЕННЫХ СИСТЕМ

А. Козлов

Санкт-Петербургский государственный университет

Введение

Определения

Операционная система — набор средств для управления и абстрагирования аппаратных ресурсов и предоставления окружения для выполнения различных приложений.

ОС реального времени — ОС, где время от возникновения события до отклика системы укладывается в некоторые определенные рамки.

ОС для встроенных систем — ОС предназначенная для работы в условиях с сильно ограниченными аппаратными ресурсами. Далее будут рассмотрены ОСРВ для встроенных систем.

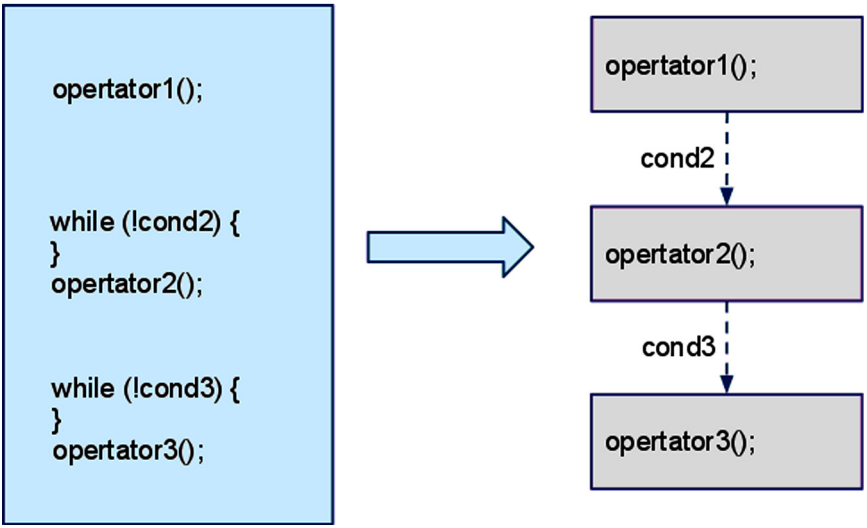
Предметная область

Одним из способов выполнения свойств РВ является использование специальных алгоритмов, особых структур данных. Такая методика не всегда применима, так как это повлечет за собой дополнительные сервисные расходы, что является заведомо плохой практикой при ограниченных ресурсах. Альтернативный способ заключается в изменении концепции приложений так, что бы они удовлетворяли некоторым требованиям, при выполнении которых вся система будет обладать свойством РВ. Для примера, существует практика отказа от механизма виртуальной памяти для критических участков программ.

Основной особенностью ОСРВ является детерминированный алгоритм переключения задач (нитей или процессов). Для встроенных систем нити используются более широко по сравнению с процессами из-за ограничений по памяти. Существуют платформы, вычислительной мощности которых хватает для выполнения ряда приложений, но с существенно малым количеством оперативной памяти, что делает непригодным использование классических нитей. Каждая нить должна иметь как минимум собственный стек достаточного размера, равный максимальной оценке заполнения стека. Существует возможность динамического изменения размера стека, но оценка её реализации амортизировано пропорциональна количеству элементов в стеке, одиночное помещение в стек может потребовать количество операций, пропорциональное размеру стека. Таким образом, принцип организации приложений в нити не всегда подходит для ОСРВ ВС.

Механизм сообщений

Предлагается реализация переключения, где каждое приложение разбивается на конечное множество состояний, переходы между которыми выполняются по сообщениям извне.

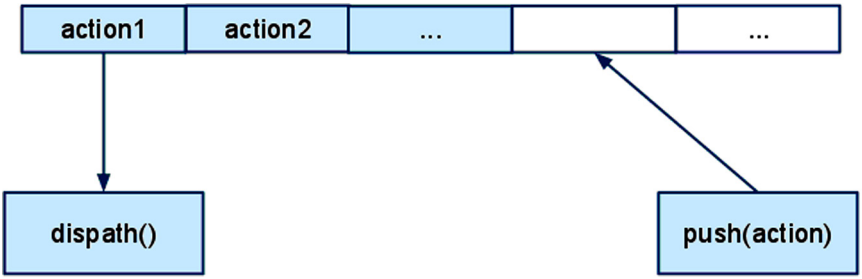


Более детальное описание: система разбивается на 2 подсистемы.

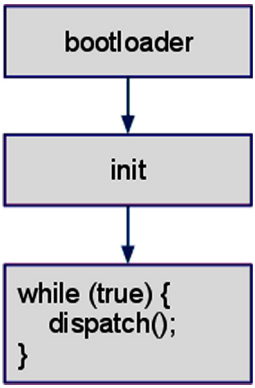
Очередь действий

- Поместить сообщение в систему (аргументы: действие — структура вида адрес функции, аргументы функции).
- Диспетчеризовать (нет аргументов).

Организуется очередь действий. Приложения и драйверы устройств помещают действия в очередь. Диспетчер изымает действие из очереди, вызывает указанную функцию с аргументами, после её завершения последовательность действий повторяется.



Цикл выполняется как единственное приложение в классическом понимании термина.

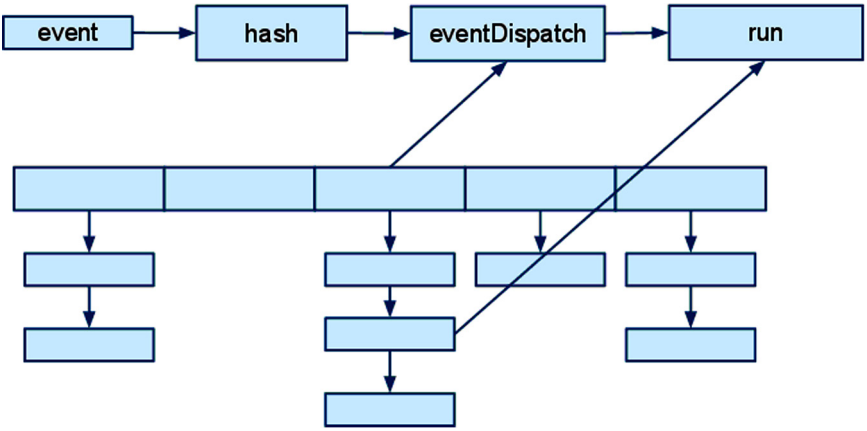


Подсистема рассылки

Вторая подсистема — подсистема рассылки:

- Подписаться на событие (аргументы: событие — элемент натурального ряда; действие)
- Оповестить о событии (аргументы: событие)

Вводится массив списков. Массив нумеруется хеш-функцией от событий. При подписке на событие в конец соответствующего списка заносится информация о событии и действии. При оповещении о событии вычисляется хеш-функция от события, выполняется полный (этим структура отличается от классической хеш-таблицы) проход по соответствующему списку: если произошедшее событие и событие в элементе списка совпадают, выполняется действие.



Следует заметить, что функция оповещения о событии не вызывается напрямую, а только через систему сообщений. Использование двух подсистем совместно даёт абстрактную (события) и эффективную (сообщения) систему для выполнения приложений.

Усовершенствование и применение

Предложенный механизм легко может быть обобщён для поддержки приоритетов. Это достигается запуском нескольких обработчиком действий, каждый в отдельной нити. Таким образом, количество нитей соответствует числу приоритетов, что в общем случае существенно меньше числа приложений.

Аргументы в защиту разделения на две подсистемы. Первая подсистема предназначена в первую очередь для драйверов. Такой приём позволяет существенно снизить время обработки прерывания с помощью вынесения основной логики работы драйвера в обработчик действия. Таким образом, пропадает необходимость в более сложных обработчиках прерывания, поддерживающих наложение прерываний (*interrupt nesting*). В то же время, логику драйвера не следует размещать в пространстве событий; иначе происходит наложение собственно обработчиков событий и драйверов, появляются «нежелательные» для использования приложениями события, разработчику ОС приходится проявлять осторожность в выборе событий для драйверов, снижается общая инкапсулированность ОС (приложение может вызвать логику драйвера, которая, естественно, предполагает, что произошло прерывание, аппаратура изменила своё состояние).

Заключение

Таким образом, построенный механизм рассчитан на приложения управления, большую часть времени выполнения ожидающие внешнего события.

Преимущества:

- Приложения используют единый стек.
- Быстрая обработка событий.
- Упрощение обработчиков прерываний.

Недостатки:

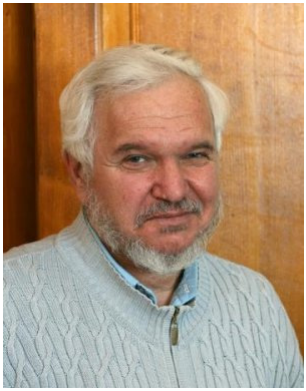
- Приложение должно быть написано особым образом.
- Не подходит для больших вычислительных последовательных приложений.

Список литературы

1. *Бурдонов И. Б., Косачев А. С., Пономаренко В. Н.* Операционные системы реального времени // ИСП РАН, препринт 14, М., 2006.

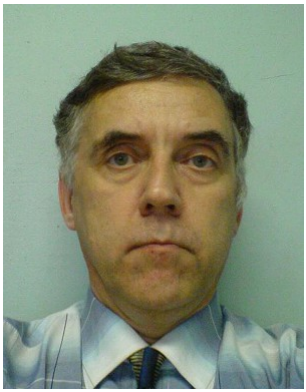
2. *Таненбаум Э.* Операционные системы, разработка и реализация // СПб.: Питер, 2006.

Фундаментальная информатика



Косовский
Николай Кириллович

д.ф.-м.н., профессор
заведующий кафедрой информатики СПбГУ



Герасимов
Михаил Александрович

к.ф.-м.н., доцент кафедры информатики СПбГУ

ТЕХНОЛОГИЯ МИГРАЦИИ ПРОЕКТОВ СО SPRING FRAMEWORK В ENTERPRISE JAVA BEANS

Д. С. Шульгин

Основная задача

Работа современной компании трудно представима без использования специализированной информационной системы. В широком смысле «информационная система» имеется практически на любом предприятии. Преимущества от внедрения такой информационной системы касаются различных аспектов работы компании: оптимизации организационной структуры, хранения и обработки данных, развития информационной базы для подготовки самых разнообразных решений, развития персонала и повышение эффективности использования рабочего времени как персонала, так и руководителя, возможности контролировать бизнес-процессы и повышать эффективность работы.

Подобная система была разработана для Городской справочной службы по недвижимости «Квартирный Вопрос», входящий в состав Группы Компаний «Бюллетень Недвижимости».

Система является распределенным приложением, реализованным на языке программирования Java, с использованием технологии Google Web Toolkit (GWT), позволяющей компилировать код на Java в JavaScript. Для реализации объектов был выбран пакет EclipseLink, для управления этими объектами и работой с базой данных Firebird — Spring Framework.

Основную задачу данной работы можно описать следующим образом :

Допустим, есть система, реализованная с помощью Spring Framework, и необходимо перейти к использованию Enterprise JavaBeans. При выполнении данной работы хотелось бы знать, с какими трудностями можно столкнуться, сколько ресурсов для этого необходимо и будет ли переход оправдан с точки зрения эффективности. Для ответа на эти вопросы рассмотрим информационную систему, реализованную на практике. На модели этой информационной системы изучим весь процесс миграции, исследуем эффективность системы в том и другом случае, а на основе выводов сформулируем некую методологию в общем виде, с помощью которой впоследствии можно будет оценить идею перехода для различных систем.

Для решения поставленной задачи будет построена упрощенная модель информационной системы по двум причинам. Во-первых, политика конфиденциальности компании не позволяет показать исходные коды и данные, во-вторых, на упрощенной модели будут отражены только нужные для задачи структуры данных, логические связи и схемы взаимодействия.

Этапы решения поставленной задачи:

- Построение упрощенной модели.
- Построение упрощенной модели.

- Проведение исследования эффективности, нагрузки на систему, ее быстродействие в стандартных для нее ситуациях Исследования проводились на основе дипломной работы студента 5-го курса Математико-Механического факультета СПБГУ Ильи Данилова «Имитационное моделирование контейнера Enterprise JavaBeans».
- Реализация созданной модели с использованием Enterprise JavaBeans.
- Исследования модели.
- Сравнение результатов исследований.
- Составление руководства по миграции проектов со Spring Framework. на Enterprise JavaBeans.

Используемые технологии

Spring Framework — пакет с открытыми исходными кодами для Java-платформы. С его простотой и гибкостью был призван справиться с недостатками Java Enterprise Edition. Однако сравнивать Spring Framework с Java EE не совсем корректно. В то время как Java EE определяет платформу реализуемую сервером приложений, а бизнес логика должна быть реализована в Enterprise JavaBeans, Spring Framework определяет модель программирования, которая может быть реализована как на Java EE платформе, так и на других.

Spring — это мощный фреймворк, который решает множество обычных проблем в Enterprise Java приложениях. Множество свойств Spring также полезны в большом разнообразии Java окружений, вне стандартного Java EE. Spring предоставляет непротиворечивый способ управления бизнес-объектами и поощряет хорошую практику, такую как программирование на уровне интерфейсов, а не классов. Этот фреймворк достаточно мощный, чтобы позволить многим приложениям освободиться от сложности EJB, в то же время наслаждаясь основными функциями, обычно ассоциируемыми с EJB.

В то же время, Enterprise JavaBeans — больше, чем просто технологическая подложка. Ее использование подразумевает еще и технологию (процесс) создания распределенного приложения — навязывает определенную архитектуру приложения, а также определяет стандартные роли для участников разработки. EJB формируют промежуточный слой, который обеспечивает изоляцию клиентских приложений от деталей внутренней организации данных, предоставляя им некоторый прикладной программный интерфейс, ориентированный на решение определенного круга задач для конкретного клиента. Enterprise JavaBeans — это существенный шаг к стандартизации модели распределенных объектов в Java.

В последней фразе заключено главное преимущество EJB и главный недостаток Spring. Об этом говорится в спецификации JSR 244: JavaTM Platform, Enterprise Edition 5 (Java EE 5) комитет JCP постановил, что Spring Framework не является реализацией JSR спецификации и, поэтому, не включен в список стандартных JEE технологий. Spring Framework использу-

ется для разработки Java/Java EE приложений, таким образом он становится частью Java EE приложения, но не является Java EE технологией. При использовании Spring Framework неизбежно исправление множества ошибок при миграции, в то время как, используя реализацию стандартной Java EE технологии Enterprise JavaBeans, можно практически без ошибок перенести приложение, например, на другой сервер приложений.

Ожидаемые результаты

На основе полученных данных при смене технологии будет составлено руководство по переходу с учетом всех совершенных ошибок, встреченных трудностей и путей их решения.

Стоит отметить, что ситуация, когда есть реальная необходимость перевода проекта, происходит нечасто. Программисты, которые свободны в выборе средств разработки, с большой долей вероятности выберут Spring Framework, несмотря на то, что он не является стандартом Java. Разработчики, выбор которых ограничен рамками стандартов, скорей всего будут использовать Enterprise JavaBeans. В связи с этим, полученная в результате методология в большей степени будет полезна тем, кто стоит перед выбором, какую из двух технологий выбрать для своего будущего проекта.

Помимо этого, используя полученные при исследовании данные, можно будет сравнить быстродействие информационной системы с разными реализациями. Учитывая, что система запущена на сервере приложений Oracle Glassfish Server 3, то ожидается, что реализация на Enterprise Java Beans будет эффективнее, нежели реализация на Spring Framework, как раз за счет стандартов JavaEE.

Степень готовности

Оценивая степень выполнения задачи, можно оценить ее в 60%.

Реализовано:

- Построение упрощенной модели.
- Построение упрощенной модели.
- Проведение исследования эффективности, нагрузки на систему, ее быстродействие в стандартных для нее ситуациях.

В разработке:

- Реализация созданной модели с использованием Enterprise JavaBeans.

Еще не реализовано:

- Исследования модели.
- Сравнение результатов исследований.
- Составление руководства по миграции проектов со Spring Framework на Enterprise JavaBeans.

ПРЕОБРАЗОВАНИЯ ПРОГРАММ ДЛЯ РАБОТЫ С РАЗРЕЖЕННЫМИ МАТРИЦАМИ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТАЛЬНОЙ СИСТЕМЫ SPARSEASSIST

М. Л. Симуни

Санкт-Петербургский Государственный Университет

В докладе описывается инструментальная система SparseAssist, предназначенная для поддержки процесса разработки программ, обрабатывающих разреженные матрицы.

Разреженные матрицы (матрицы, в которых большая часть элементов равна нулю [1]) широко используются в различных приложениях от задач математического моделирования [2, 3] до поисковых систем. Рассмотрим некоторые особенности задач обработки таких матриц:

- Очень высокие требования к быстродействию и обьему обрабатываемых данных.
- Для большинства алгоритмов существуют различные варианты реализации. В частности, как правило, матрицу можно обрабатывать по строкам, или по столбцам, или разбить матрицу на блоки и обрабатывать по блокам и т. д.
- Скорость работы зависит от большого числа факторов: архитектуры процессора, организации кеша, структуры матрицы. Для выбора оптимального варианта реализации необходимо, как правило, экспериментальное сравнение различных вариантов.
- Как правило, программы должны работать с матрицами различных типов (целые, вещественные, разной точности).
- При промышленном программирование задач такого типа как правило не использует конструкции высокого уровня (в частности, объекты). Как следствие, такие программы обычно трудны для понимания и поддержки.

Пример применения SparseAssist

Рассмотрим небольшой фрагмент реального кода (библиотека CSparse [3]):

```
int p, j, n, *Lp, *Li;
double *Lx;
n = L->n; Lp = L->p; Li = L->i; Lx = L->x;
for (j = 0; j < n; j++) {
x [j] /= Lx [Lp [j]];
for (p = Lp [j]+1; p < Lp [j+1]; p++)
x [Li [p]] -= Lx [p] * x [j];
}
```

Это код функции, решающей треугольное матричное уравнение $Lx = b$, где L — разреженная треугольная матрица, X и b — обычные (плотные) вектора. Матрица представлена в формате CCS (сжатое представление по столбцам) [1]. Данный код имеет недостатки с точки зрения простоты его понимания и удобства поддержки:

- Код выглядит достаточно запутанным. Для того, чтобы понять, что он делает, необходимо знание деталей формата представления данных.
- Допустим, перед нами стоит задача переработать эту программу для сжатого представления по строкам (формат CRS). Эта работа требует понимания обоих форматов и переработки алгоритма, при этом существует риск возникновения ошибок.

Рассмотрим, как можно преобразовать этот код с использованием инструментальной системы SparseAssist.

1. Инкапсуляция доступа к матрице при помощи FOREACH.

Пользователь может выделить фрагмент кода и применить к нему реализованную в SparseAssist операцию «Инкапсулировать доступ к матрице». Необходимо также указать, какой формат хранения используется (CCS). Будет сгенерирован следующий код:

```
FOREACH_COLUMN(L, j, L_j, Ljj) {
x[j] /= Ljj;
FOREACH_ELEM_BELOW_DIAGONAL(L_j, i, Lij)
x[i] -= Lij * x[j];
}
```

Что произошло при этом преобразовании? SparseAssist проанализировал программу и инкапсулировал логику доступа к матрице при помощи специализированных заголовков циклов FOREACH. Специализированные заголовки циклов реализованы, как макросы препроцессора и по своей функциональности аналогичны макросу FOREACH библиотеки Boost. В отличии от Boost, эти макросы задают несколько переменных цикла (например, значение очередного элемента и его индекс).

2. Изменение порядка обхода матрицы.

Пользователь может выделить фрагмент кода и применить к нему операцию *«Изменить порядок обхода»*, указав, что необходимо обойти матрицу по строкам. Будет сгенерирован следующий код:

```
FOREACH_ROW(L, i, Li_, Lii) {
FOREACH_ELEM_BEFORE_DIAGONAL(Li_, j, Lij)
x[i] -= Lij * x[j];
x[i] /= Lii;
}
```

SparseAssist генерирует новый вариант программы, соответствующий порядку обхода матрицы по строкам. При этом проверяется, что новый порядок обхода не нарушает зависимости между операторами, то-есть не меняет результат вычисления. При необходимости меняется порядок вычисления операторов.

3. Переход к шаблону.

Пользователь может применить к фрагменту кода операцию *«Создать шаблон»*. Будет сгенерирован шаблон, обеспечивающий возможность работы с матрицами различных типов:

```
template <class Matrix, class Vector>
void lsolve(const Matrix& L, Vector& x)
{
FOREACH_ROW(L, i, Li_, Lii) {
FOREACH_ELEM_BEFORE_DIAGONAL(Li_, j, Lij)
x[i] -= Lij * x[j];
x[i] /= Lii;
}
}
```

В общем случае задача генерации шаблона для данного фрагмента кода является достаточно сложной [4]. Однако для данной предметной области эта функциональность может быть реализована надежно и эффективно, поскольку взаимосвязь используемых типов данных и операции над ними известны заранее.

Представление информации о форматах хранения матриц

Описание преобразований для различных схем хранения матриц. Для каждого формата, поддерживаемого системой, задаются правила преобразования низкоуровневой программы в программу с использованием FOREACH макросов. Формат задается правилами. Правила определены в текстовом файле и могут быть легко дополнены описаниями новых форматов.

Описание семантики схем хранения матриц. Для каждого формата хранения матрицы задается ее семантика. А именно, для каждого элемента схемы хранения указывается, что ему соответствовало бы в плотной матрице. Эта информация используется для преобразования программ при переходе к другому формату хранения. Кроме этого, ее можно использовать для автоматического добавления комментариев.

Существующие подходы к созданию инструментальных средств для программ обработки разреженных матриц

В работе [5] описана система генерации программ для обработки разреженных матриц из программ для обработки плотных матриц. Система предназначена только для обработки матриц, представленных в формате CRS. В работе [6] были предложены способы автоматической генерации программ для обработки разреженных матриц из высокоуровневых спецификаций с учетом информации о формате хранения данных. В работе [7] описан подход, при котором программа. автоматически подбирает оптимального варианта реализации на основе заданного пользователем набора возможных оптимизаций.

Литература

1. *Писсанецки С.* Технология разреженных матриц. М.: Мир, 1988. 416 с.

2. *Дубицкий С. Д.* ELCUT 5.1 — платформа разработки приложений анализа полей // Exponenta Pro. Математика в приложениях. 2004, Н. 1 (5). С. 20–26.

3. *Davis T. A.* 2006. Direct Methods for Sparse Linear Systems. SIAM, Philadelphia, PA.

4. *Siff M.* and *Reps T.* 1996. Program generalization for software reuse: from C to C++. SIGSOFT Softw. Eng. Notes 21, 6 (October 1996). Pp. 135–146.

5. *Bik A. J. C., Brinkhaus P. J. H., Knijnenburg P. M. W., Wijshoff H. A. G.* The Automatic Generation of Sparse Primitives // ACM Transactions on Mathematical Software. 1998. Vol. 24. P. 190–225.

6. *Ahmed N., Mateev N.*, and *Pingali K.* 2000. A framework for sparse matrix code synthesis from high-level specifications. In Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM) (Supercomputing ‘00). IEEE Computer Society, Washington, DC, USA, Article 58.

7. *Vuduc R.* 2003. Automatic Performance Tuning of Sparse Matrix Kernels. Doctoral Thesis. University of California, Berkeley.

ПОЧТИ ЛИНЕЙНЫЙ ПО ВРЕМЕНИ ВЫПОЛНЕНИЯ ЖАДНЫЙ АЛГОРИТМ ДЛЯ ПРИБЛИЖЕННОГО РЕШЕНИЯ NP-ПОЛНОЙ ЗАДАЧИ О РАЗБИЕНИИ

М. А. Герасимов

Санкт-Петербургский Государственный Университет

Аннотация. Рассматривается жадный алгоритм, позволяющий приближенно решать NP-полную задачу о разбиении множества целых чисел на два подмножества одинакового веса за почти линейное время на одноленточной, одноголовочной машине Тьюринга с входной и выходной лентами. Рассматриваются некоторые свойства этого алгоритма, даются оценки точности разбиения и скорости работы при различных входных условиях.

Детерминированная машина Тьюринга

Для оценки скорости работы рассматриваемого алгоритма будем использовать детерминированную одноленточную машину Тьюринга [5] с входом и выходом следующего вида. Входные данные рассматриваемой машины Тьюринга записаны на входной ленте, обрабатываются на рабочей ленте, а результат получается на выходной ленте. Машина имеет некоторое конечное число состояний $Q = \{q_1, \ldots, q_N\}$. При работе машины Тьюринга используется алфавит, состоящий из четырех символов {#, b, 0, 1}. Результатом работы алгоритма считается битовая последовательность, кодирующая найденное разбиение исходного множества целых чисел.

Если входные данные (исходное множество целых чисел) записаны в виде битовой последовательности на входной ленте между двумя маркерами «#», то считывание второго маркера будет означать конец цепочки входных данных. Входная лента позволяет считывать входные данные произвольное количество раз.

Выходная лента выбранной модели позволяет только записывать результат вычисления в виде последовательности символов рабочего алфавита. Каждый символ выходной цепочки записывается только один раз и больше не изменяется.

Скоростью работы алгоритма на такой машине Тьюринга будем считать число шагов (как функцию от длины входных данных), необходимое для корректного завершения алгоритма и получения результирующей цепочки данных.

Приближенное решение задачи о разбиении

Задача разбиения множества целых чисел решается в следующей стандартной постановке [2]. Пусть дано множество целых чисел $S = \{n_1, n_2, \ldots, n_k\}$.

Требуется найти такие два подмножества S_1 и S_2, что $S_1 \cup S_2 = S$ и суммарные веса множеств S_1 и S_2 равны. Приближенное решение данной задачи с относительной точностью $1 > e > 0$ — это такие два множества S_1' и S_2', что $S_1' \cup S_2' = S$ и $|w(S_1') - w(S_2')|/(w(S)) < e$. Если задача о разбиении решается точно, то e может быть сколь угодно малым, но в этом случае неизвестен алгоритм, способный найти такое решение за полиномиальное число шагов. Если допустить, что $e > 0$, то возможно найти соответствующее решение за почти линейное число шагов, более точно за $O(n \log(n))$ шагов, где n — суммарная длина входных данных алгоритма.

Если рассматривать множество алгоритмов, решающих задачу о разбиении приближенно, то можно так сформулировать задачу оптимизации разбиения: найденный алгоритм будет оптимальным, если при заданных ресурсах, может быть найдено решение с минимальным е. В частности, при $e = 0$ получается точное решение задачи о разбиении, которое может быть найдено при ограниченных ресурсах (например, за полиномиальное время от длины входных данных).

Жадный алгоритм быстрого разбиения

Рассмотрим следующий достаточно простой жадный алгоритм, приближенно решающий задачу о разбиении.

Пусть на первом этапе работы машина Тьюринга сортирует входные натуральные числа, закодированные битовыми цепочками в порядке убывания их значение (весов). При реализации этого этапа с использованием алгоритма быстрой сортировки [3] это может быть сделано за время $O(n \log(n))$, где n — общая длина входных данных.

На втором этапе решения приближенной задачи о разбиении алгоритм помещает отсортированный набор чисел в два множества по следующему правилу: в каждый момент времени очередное число помещается в то множество, вес которого минимален.

По завершении работы алгоритма получается два подмножества натуральных чисел из входного набора данных, веса которых «почти равны». В наиболее благоприятном случае эти веса будут равны точно. В качестве результата работы алгоритма можно выписать битовое представление полученных множеств натуральных чисел.

Достаточно очевидно, что рассматриваемый алгоритм является «жадным» и в любом случае позволяет получать приближенное решение задачи. Другое дело, что это решение, как правило, является не оптимальным в том смысле, что возможно существование другого разбиения данного входного множества, с меньше относительной погрешностью e.

С другой стороны, из описания алгоритма видно, что наибольшая временная сложность его работы принадлежит первому этапу (сортировке). Таким образом, общая временная сложность определяется именно эффектив-

ностью этого этапа. Как уже было замечено, эта оценка равна $O(n\log(n))$, что будет верхней оценкой сложности всего рассматриваемого жадного алгоритма быстрого разбиения.

Специальные случаи задачи о разбиении

В том случае, когда входной поток данных обладает некоторыми специальными свойствами, можно получить оценку точности получаемого решения в зависимости от характеристик обрабатываемого потока. В частности, если известно, что для любых n_i и n_j из входного потока, таких, что $n_i > n_j$ справедливо соотношение $n_j/n_i > ½$ (исходные натуральные числа не слишком сильно различаются), то рассматриваемый жадный алгоритм всегда находит приближенное решение с погрешностью не большей чем $1/k$, где k — количество целых чисел во входном потоке. Данное требование можно даже ослабить, а именно, достаточно, чтобы после сортировки натуральных чисел входного потока по убыванию их весов, существовала некоторая геометрическая последовательность с основанием $1 < t < 2$, такая, что $a_{i+1} > n_i > a_i$, $a_i = b \cdot t^i$, $b > 0$ для всех элементов n_i отсортированного входного набора данных. Данный результат остается справедливым и в том случае, когда входной набор данных удается разбить на фиксированное конечное число наборов, каждый из которых обладает требуемым свойством.

Остается случай, когда входной поток данных, даже в отсортированном виде не удается разбить на фиксированное конечное число наборов с указанным свойством. Это возможно в двух случаях. Либо существует некоторая геометрическая последовательность с основанием $t > 2$, такая, что $a_i = b \cdot t^i$, $a_{i+1} > n_i > a_i$, где $b > 0$ (тогда точная задача о разбиении не имеет решения). Либо для любой геометрической последовательности с основанием $t > 1$, начиная с некоторого номера $i \geq 1$, все элементы отсортированного входного потока с номерами большими i, становятся меньше a_i. В этом случае, начиная с этого номера i, справедливо неравенство $n_{i+1}/n_i < 1/t$. Понятно, что тогда задача также может быть решена с точность $1/k$, где k — количество чисел во входном наборе данных.

Поскольку рассматриваемый жадный алгоритм работает после сортировки линейное число шагов от суммарной длины отсортированного входного потока данных, то основная временная сложность алгоритма определяется скоростью сортировки входных чисел. Если применить методы быстрой сортировки, то временная сложность сортировки входного набора чисел суммарной длиной n, может быть сделана меньше $Cn\log(n)$, где C — некоторая константа. Таким образом, рассматриваемый жадный алгоритм имеет временную оценку сложности для детерминированной одноленточной машины Тьюринга с входной и выходной лентой — $O(n\log(n))$, при гарантированной оценке относительной точности приближенного решения $1/k$ (если точное решение существует), где k — количество чисел входного набора данных.

Литература

1. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. М., 1979.

2. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

3. *Кнут Д.* Искусство программирования. Т. 1: Основные алгоритмы. 3-е изд. М.: Издат. дом «Вильямс», 2000.

4. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ. 2-е изд. М.: Издат. дом «Вильямс», 2007.

5. *Минский М.* Вычисления и автоматы. М., 1971.

САМООБУЧАЮЩИЕСЯ СИСТЕМЫ РАСПОЗНАВАНИЯ ОБРАЗОВ

А. П. Бельтюков (e-mail: belt@uni.udm.ru),
А. Н. Тетерин (e-mail: v777@izh.com)

Предлагается новая теория восприятия, хранения и воспроизведения информации на базе детерминистских теорий классификации (без учителя), кодирования и иерархического синтаксического анализа на основе определения языка как дерева объектов. Это так называемое «дерево описания данных», в листьях которого находятся терминальные символы, а в узлах грамматика и атрибуты объекта, обозначаемого нетерминальным символом. Главное правило таких описаний: нельзя определять объект через самого себя. Терминальные символы (иероглифы) и их атрибуты, составляющие входную *n*-мерную матрицу, определяются с использованием геометрического подхода к классификации. В процессе ее анализа строится и оптимально кодируется эвристическими правилами другое дерево объектов — дерево данных с числовыми атрибутами, «дерево разбора». Это дерево разбора — сжатое описание входной последовательности, которая может быть воспроизведена с небольшими искажениями, с исправлением допущенных ошибок или с добавлением ассоциаций.

Восприятие — определяет сложный процесс приёма и преобразования и распознавания информации, получаемой при помощи аудио видео устройств ввода, датчиков, формирующих целостный образ объекта, свободный от ошибок (приема, преобразования, распознавания реальной нетривиальной ситуации). Различение отдельных признаков в объекте, выделение в нём информативного содержания, адекватного цели действия, ее сохранение с последующим воспроизведением в рамках текущей ассоциации, либо ассоциаций связанных с ним, которые в свою очередь порождают другие ассоциации и т.д. (формирование, проецирование «чувственного образа»).

Распознавание образов обычно связано с идеальными условиями идентификации личности с помощью ЭВМ по биометрическим параметрам (отпечатки пальцев, радужка глаза, лицо, голос, письменный, клавиатурный почерк) или вводом в ЭВМ речи, различных видов текстовой графической информации. Идеальные условия подразумевают ввод биометрических параметров с максимально близкого расстояния, когда распознаваемый объект занимает большую часть площади сканирования или речи одного диктора, не говоря уже об отсутствии атмосферных осадков, грозы, молнии, разнообразных шумов т. д. Решение таких проблем подразумевает последовательное решение следующих задач:

1. Оцифровка аналоговой информации.
2. Цифровая обработка сигнала, сжатие информации.

3. Распознавание объектов с исправлением ошибок этапов 1, 2.
   а) Классификация, предсказание.
      - Неправильное предсказание является естественным и говорит о происшедших изменениях, приводит к увеличению сохраняемых данных.
   б) Кодирование.
   в) Семантический анализ, предсказание.
      - Неправильное предсказание говорит об ошибках в структуре и необходимости ее изменения (самообучение).
4. Решение обратной задачи: проецирование воспринятой информации на «отдельные участки мозга» (память).
5. Сравнение с данными этапа 1 и в случае неудачи:
   а) Последовательная смена параметров 1 и 2 этапов.
   б) Исправление структурных ошибок этапа 3 (самообучение**).**
6. Хранение воспринятой информации.
7. Воспроизведение ассоциаций с воспринятой информацией.

Первая задача имеет значение для аудио информации, телеметрических датчиков, видеоинформация воспринимается ПЗС структурами, имеющими цифровой выход. На этом этапе можно менять качество оцифровки (количество разрядов, разрешение).

Объемы данных получаемых после первого этапа достаточно велики, и необходимо использовать известные методы сжатия данных и выделения первичных признаков, имеющих числовые характеристики.

Третья задача связана с классификацией, кодированием в результате которых мы получаем вектор кодовых слов (аудиоданные) с числовыми параметрами матрицу (видео) или нечто большей размерности (в зависимости от количества признаков), меняющейся во времени. Или другими словами n-мерная символьная матрица, каждый элемент которой имеет некоторое количество атрибутов. Обрабатывать эти структуры можно последовательно, параллельно, а сам процесс выделения объектов представляет некоторую иерархию, отражающую его структуру. В последнем случае мы переходим к иерархическому синтаксическому анализу, в результате которого определяется семантическое значение объекта и дерево разбора.

Деревья разбора связаны со временем и местом (контекст), хранятся в иерархической базе данных. Теперь настало время оценить качество решенных задач. Для этого достаточно воспроизвести информацию, хранящуюся в дереве разбора (5) и сравнить с цифровой информацией на входе. В случае неудовлетворительного результата можно для начала поменять разрешение при вводе видео или увеличить количество разрядов для оцифровки аудио. Можно использовать несколько стратегий смены разрешений для точки направления взгляда и траектории его движения и изображения в целом (это выходит за рамки статьи).

В случае если результаты смены разрешений не изменили ситуации, необходимо скорректировать параметры метода сжатия, используемого на втором этапе, либо заменить его. В конечном итоге можно полностью отказаться от него, используя упрощенную модель решения задачи 3.

В случае обнаружения структурных ошибок, связанных с неправильным распознаванием объектов необходимо корректировать правила, дерево данных, числовые параметры метода 3.

Когда все ошибки устранены дерево разбора остается для хранения в базе данных и него устанавливаются обязательные, факультативные и необязательные связи. Примером обязательных связей — время, место в общей иерархии, объекты, не попавшие в поле зрения робота, но находящиеся справа, слева, снизу, сверху в общей иерархии. Факультативные связи — новое место (на что похоже), старое (что изменилось). Необязательное — отношение к месту хорошее, плохое (почему?). По этим связям можно порождать ассоциации — деревья разбора накопленные ранее. После воспроизведения последних их можно распознавать при полном отсутствии ошибок (относительно быстро) и они будут приводить к другим ассоциациям.

Пункты 3, 5 и 6 необходимо повторить для одной или всех ассоциаций столько, сколько нужно в зависимости от цели действия.

При решении задач 1 и 2 как во входных данных возможны ошибки, которые устраняются выполнением 3. Для решения 3 предлагается использовать геометрический подход к классификации с самообучающимися алгоритмами [2] «интеллектуальные» эвристические правила [3], иерархический синтаксический анализ[4], возможно вместе с геометрическим подходом для обработки числовых атрибутов. Для решения 6 — иерархические базы данных [5]. Для воспроизведения 4 используются деревья разбора, в полученные в результате решения 3, взятые из иерархической базы данных и процедуры вывода.

Литература

1. *Тетерин А. Н.* Геометрический подход к классификации — новая модель работы нейрона // ЖВМ и МФ. 1992. Т. 31. № 12. С. 1972–1980.

2. *Teterin A. N.*. Classification on Bounded Sets // Pattern Recognition and Image Analysis. Pleiades Publishing, 2010. Vol. 20. No. 4. Pp. 564–572.

3. *Beltyukov A. P., Teterin A. N.* On The Amount of Information // Pattern Recognition and Image Analysis. Pleiades Publishing, 2011. Vol. 21. No. 1. Pp. 66–70.

4. *Бельтюков А. П. Тетерин А. Н.* Иерархический синтаксический анализ // Научная сессия МИФИ-2010.

5. *Бельтюков А. П. Тетерин А. Н.* Иерархические базы данных для хранения и визуализации 3D объектов // Труды Второй международной конференции «Трехмерная визуализация научной технической и социальной реальности. Технологии высокополигонального моделирования». 24–26 ноября 2010. г. Ижевск.

ЭФФЕКТИВНОЕ ПО ВРЕМЕНИ И ПО ПАМЯТИ ВЫЧИСЛЕНИЕ ЭКСПОНЕНЦИАЛЬНОЙ ФУНКЦИИ КОМПЛЕКСНОГО АРГУМЕНТА НА МАШИНЕ ШЁНХАГЕ

С. В. Яхонтов

Санкт-Петербургский Гос. Университет,
математико-механический факультет
E-mail: sergey_home_mail@inbox.ru

В данной работе вводится мера емкостной сложности вычислений на оракульной машине Шёнхаге [1] и дается верхняя оценка временной и емкостной сложности предлагаемого алгоритма вычисления экспоненциальной функции комплексного аргумента в каждом круге на машине Шёнхаге.

Основные сведения о двоично-рациональных числах и конструктивных числах и функциях можно почерпнуть в [2]. Посредством **Sch(FQLINTIME//LINSPACE)** будем обозначать класс алгоритмов, квазилинейных по времени и линейных по памяти при вычислении на машине Шёнхаге.

Известно [3, 4], что многие элементарные функции вычислимы с помощью алгоритмов, квазилинейных по времени и квазилинейных по памяти. В данной работе показывается, что комплексная экспоненциальная функция и некоторые другие функции вычислимы алгоритмами, также квазилинейными по времени, но линейными по памяти.

Определение 1. *Емкостную вычислительную сложность алгоритма при расчете на оракульной машине Шёнхаге определим как сумму длин используемой памяти по всем массивам плюс максимум длины памяти, занятой стеком.*

Будем говорить, что последовательность $\phi : \mathbf{N} \to \mathbf{D} \times \mathbf{D}$, $\phi(n) = (\phi_x(n), \phi_y(n))$, $\mathbf{D}$ — множество двоично-рациональных чисел, двоично-рационально сходится к комплексному числу z, если для любого $n \in \mathbf{N}$ выполняется $prec(\phi_x(n)) = n + 2$, $prec(\phi_y(n)) = n + 2$ и $|z(n) - z| \leq 2^{-n}$, где $z(n) = \phi_x(n) + i\phi_y(n)$. Множество всех функций ϕ, двоично-рационально сходящихся к числу z, будем обозначать CF_z. Комплексное число z будем называть CF конструктивным, если CF_z содержит вычислимую функцию ϕ.

Определение 2. *Число $z \in \mathbf{C}$ будем называть Sch(FQLINTIME//LINSPACE) конструктивным комплексным числом, если существует функция $\phi \in CF_z$, вычислимая в пределах класса сложности Sch(FQLINTIME//LINSPACE).*

Пусть f — функция $f(z) : A \to \mathbf{C}$, где $A = \{z \in \mathbf{C} : |z| \leq R\}$ — круг в комплексной плоскости.

Определение 3. *Функция $f(z)$ называется Sch(FQLINTIME//LINSPACE) конструктивной комплексной функцией в круге A, если для любого z из*

этого круга существует функция ψ из $CF_{f(z)}$, вычислимая в пределах класса вычислительной сложности Sch(FQLINTIME//LINSPACE).

Пусть величина p — натуральное число, $p \geq 0$. Функция $\exp(z)$ будем рассматривать с точностью 2^{-n} в круге $|z| \leq 2^p$. Вычисление комплексной экспоненциальной функции с помощью простых преобразований сводится к расчету вещественной экспоненциальной функции. Далее приближенные значения вещественной экспоненциальной функции рассчитываются на основе разложения в ряд Тейлора с помощью модифицированного алгоритма быстрого вычисления экспоненты *LinSpaceFEE* (классический метод быстрого вычисления экспоненты описан в [3]), который, в свою очередь, использует модифицированный алгоритм двоичного деления *LinSpaceBinSplit* для гипергеометрических рядов (классический метод двоичного деления описан в [4]). Для алгоритмов *LinSpaceBinSplit*, *LinSpaceFEE* и основного алгоритма *LinSpaceExpValue* показывается квазилинейность по времени и линейность по памяти; верхняя оценка временной вычислительной сложности алгоритма *LinSpaceExpValue* — $O(M(n)\log(n)^3)$ ($M(n)$ — сложность умножения целых чисел), верхняя оценка емкостной сложности данного алгоритма — $O(n)$.

Теорема. *Комплексная функция $\exp(z)$ является вычислимой в пределах класса Sch(FQLINTIME//LINSPACE) в любом круге $|z| \leq 2^p$.*

Утверждение. *Комплексные функции $\sin(z)$, $\cos(z)$, $\mathrm{sh}(z)$, $\mathrm{ch}(z)$ являются Sch(FQLINTIME//LINSPACE) конструктивными в любом круге $|z| \leq 2^p$.*

Алгоритм *LinSpaceExpValue* вычисления комплексной экспоненциальной функции можно применять в информатике как основу **Sch(FQLINTIME//LINSPACE)** конструктивных комплексных функций $\exp(z)$, $\sin(z)$, $\cos(z)$, $\mathrm{sh}(z)$, $\mathrm{ch}(z)$, заданных на множестве **Sch(FQLINTIME//LINSPACE)** конструктивных комплексных чисел.

Если для умножения целых чисел использовать алгоритм Шёнхаге—Штрассена со сложностью $O(n\log(n)\log\log(n))$, то временная сложность алгоритма *LinSpaceExpValue* будет ограничена сверху $O(n\log(n)^4\log\log(n))$; при использовании для умножения простого рекурсивного метода с временной сложностью $O(n^{\log(3)})$ временная сложность алгоритма *LinSpaceExpValue* будет $O(n\log(n)^3 n^{\log(3)})$.

Литература

1. *Schonhage A., Grotefeld A. F. W, Vetter E.* Fast algorithms. A multitape Turing machine implementation // Germany: Brockhaus AG, 1994. 298 p. — ISBN 3-411-16891-9.

2. *Ko K.* Complexity theory of real functions // Boston: Birkhauser, 1991. 310 p. — ISBN 0-8176-3586-6.

3. *Карацуба Е. А.* Быстрые вычисления трансцендентных функций // Проблемы передачи информации. Т. 27, Вып. 4, 1991. С. 76–99. — ISSN 0555-2923.

4. *Haible B., Papanikolaou T.* Fast multiple-presicion evaluation of series of rational numbers // Proc. of the Third Intern. Symposium on Algorithmic Number Theory. June 21–25, 1998. P. 338–350. — ISBN 3540646574.

Технологии
и инструментальные средства
разработки программ

Сафонов
Владимир Олегович

д.т.н.
профессор кафедры информатики СПбГУ
член-корреспондент РАЕН

РЕАЛИЗАЦИЯ ТЕХНОЛОГИИ DESIGN-BY-CONTRACT СРЕДСТВАМИ АСПЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В СИСТЕМЕ ASPECT.NET

А. Р. Когай, В. О. Сафонов

Санкт-Петербургский Государственный Университет

Технология контрактного проектирования (Design-by-Contract) [1] — процесс проектирования, предполагающий разработку формальных и верифицируемых спецификаций (контрактов) для каждого программного элемента. Контрактная спецификация представляет собой набор утверждений, которые четко описывают, что должен и не должен делать каждый конкретный метод, определяя, таким образом, взаимоотношения программных компонентов. Технология была предложена Бертраном Мейером еще в 1986 году в контексте разработки языка программирования Eiffel, однако сегодня его идея приобретает совершенно новое звучание — программирование с использованием так называемых *«надежных компонентов» (trusted components)*, в корректности которых не должно быть никаких сомнений.

Если некоторый компонент предоставляет окружению свою функциональность, он может наложить *предусловие (precondition)* на ее использование. Предусловия выражают ограничения, выполнение которых необходимо для корректной работы программы. Корректная система никогда не вызовет функциональность, если не выполняется ее предусловие. В свою очередь компонент может гарантировать выполнение некоторого действия с помощью *постусловия (postcondition)*. Постусловие определяет состояние, завершающее выполнение программы. *Инвариант (invariant)* класса — это утверждение, выражающее фундаментальные соотношения, характерные для данного класса. Инвариант применяется к классу как целому, и этим отличается от предусловий и постусловий, характеризующих отдельные методы. Инвариант должен выполняться перед вызовом и после вызова каждого метода данного класса.

Язык Бертрана Мейера Eiffel, реализующий концепцию контрактного проектирования, не является общеупотребимым, что препятствует распространению этой концепции на практике. В то же время в традиционных языках программирования при реализации контрактных пред- и постусловий разработчик сталкивается со следующими сложностями:

- Во-первых, код проверки пред- и постусловий перемешивается с основным кодом компонента. Это снижает ясность кода и, что более важно, снижается способность программного компонента к повторному использованию. Например, если в другой системе будут более жесткие требования к производительности, может возникнуть необходимость отключить часть проверок, что неизбежно ведет к модификации кода.

- Во-вторых, код проверки пред- и постусловий рассредоточивается по всей системе. Если возникнет необходимость изменить какое-либо из условий, то придется произвести изменения во всех модулях, на которые оно распространяется. При этом достаточно тяжело поддерживать систему в согласованном состоянии.

Проверка утверждений контракта — типичный пример сквозной функциональности, которая присутствует во многих программных модулях. Классический объектно-ориентированный подход (ООП) предоставляет удобные средства для выделения логики программы в отдельные компоненты. Но в рамках данного подхода не существует возможности локализовать в отдельные модули функциональность, которая пронизывает всю систему.

Решением данной проблемы может быть аспектно-ориентированное программирование (АОП) [2]. Это новая стремительно развивающаяся парадигма, которая предлагает средства для модуляризации сквозной функциональности и ее автоматизированного добавления в целевую программу. АОП расширяет традиционную концепцию модуля по Г. Майерсу [3] понятием *аспекта (aspect)*. Аспект — это специальный модуль, содержащий фрагменты кода сквозной функциональности, активируемые в заданных точках целевой программы.

Aspect.NET [4] — языково-независимая среда разработки АОП-приложений для платформы Microsoft.NET. Aspect.NET является расширением *(add-in)* популярной интегрированной среды разработки Microsoft Visual Studio. Это означает, что разработчики могут использовать Aspect.NET как часть Visual Studio, с ее многочисленными возможностями для разработки программного обеспечения (сборка, отладка, профилирование и др.). Аспект в системе Aspect.NET определяется как исходный код класса, аннотированный конструкциями метаязыка Aspect.NET.ML. Структура метаязыка Aspect.NET.ML и его семантика не привязаны к конкретному языку реализации, что позволяет использовать одни и те же спецификации аспектов на Aspect.NET.ML в сочетании с различными языками их реализации. Подобная языковая независимость метаязыка АОП — важный принцип Aspect.NET. Кроме того, в отличие от многих других инструментов АОП, Aspect.NET предоставляет возможность *управляемого внедрения (user-controlled weaving)*. Графический пользовательский интерфейс Aspect.NET Framework GUI позволяет выбрать, внедрять или не внедрять конкретный аспект в ту или иную точку целевого приложения.

В данной работе предлагается реализовать контрактное проектирование с помощью аспектно-ориентированного программирования в системе Aspect.NET [5]. В рамках такого подхода основная бизнес-логика компонента не требует внесения изменений. Контрактная спецификация (т. е. предусловия, постусловия и инварианты) реализуется в аспекте. После чего происходит автоматическое внедрение аспектов в заданные точки программы на уровне бинарных файлов и сборок.

Рассмотрим пример:

```
public static void LoadInnerEntities(IEnumerable<Tender> tenders)
{
    // possible ArgumentNullException, ArgumentException
    foreach (Material material in materials)
        dicMaterials.Add(material.Id, material);
    foreach (Supplier supplier in suppliers)
        dicSuppliers.Add(supplier.Id, supplier);
    foreach (Price price in prices)
        dicPrices.Add(price.Id, price);

    // possible ArgumentNullException, KeyNotFoundException
    foreach (Tender tender in tenders)
    {
        tender.Material = dicMaterials[tender.MaterialId];
        tender.Supplier = dicSuppliers[tender.SupplierId];
        tender.Price = dicPrices[tender.PriceId];
    }
}
```

Представленный фрагмент исходного кода целевого приложения содержит несколько потенциально уязвимых мест при работе со словарем. Следуя принципам контрактного проектирования, прежде чем вызвать метод, разработчик должен убедиться, что выполнены его предусловия. Т. е. перед добавлением элемента нужно проверить, что элемента с таким ключом еще нет в словаре. А перед извлечением — что такой ключ в словаре, наоборот, присутствует. В обоих случаях нужно также убедиться, что передаваемый ключ не равен нулевой ссылке. Если добавлять эти проверки напрямую в код, объем данного фрагмента возрастет как минимум в полтора раза. Не говоря уже о том, что на сопровождение и внесение изменений в готовый программный код, по оценкам исследователей, тратится около 70 % времени работы разработчика.

Вместо этого реализуем контрактную спецификацию в отдельном программном модуле — аспекте:

```
[AspectDescription("Contract for IDictionary")]
public class DictionaryAspect : Aspect
{
    [AspectAction("%before %call IDictionary.Add")]
    public static void CheckAddPrecondition()
    {
        IDictionary targetObject = (IDictionary)TargetObject;
        Contract.Requires<ArgumentNullException>(key != null);
        Contract.Requires<ArgumentException>(!targetObject.Contains(key));
    }

    [AspectAction("%before %call IDictionary.get_Item")]
    public static void CheckGetItemPrecondition()
    {
        IDictionary targetObject = (IDictionary)TargetObject;
        Contract.Requires<ArgumentNullException>(key != null);
        Contract.Requires<KeyNotFoundException>(targetObject.Contains(key));
    }
}
```

При внедрении данного аспекта в указанные точки исходного кода, мы получим точно такую же программу, как если бы все проверки вносились вручную. При этом достигаются следующие преимущества:

- Во-первых, обеспечивается ясность, наглядность и легкость сопровождения исходного кода программы. Основной код компонента не подвергается изменениям, интеграция контракта осуществляется аспектным компоновщиком на уровне бинарных файлов и сборок.
- Во-вторых, АОП предоставляет возможность автоматического добавления новой сквозной функциональности в код целевых приложений, в отличие от традиционных сред разработки, в которых эти операции приходится выполнять вручную.
- Используя графический пользовательский интерфейс системы Aspect.NET, можно настраивать необходимый уровень проверок: включить все проверки, оставить только предусловия или же полностью отключить (например, в конечной версии продукта, если имеются жесткие требования к производительности). При этом исходный код компонента не модифицируется, т. е. полностью сохраняется его способность к повторному использованию.
- Наконец, АОП широко поддерживается различными средами разработки ПО, что способствует распространению концепции контрактного проектирования на практике.

Литература

1. *Meyer B.* Object-Oriented Software Construction. Prentice Hall, 1997.

2. *Safonov V. O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008.

3. *Майерс Г.* Надежность программного обеспечения. М.: Мир, 1980.

4. *Сафонов В. О.* Aspect.NET — инструмент аспектно-ориентированного программирования для разработки надежных и безопасных программ // Компьютерные инструменты в образовании. 2007. № 5.

5. *Когай А. Р.* Применение аспектно-ориентированного программирования для поддержки технологии Design-by-Contract // Компьютерные инструменты в образовании. 2010. № 4.

ПРИМЕНЕНИЕ СИСТЕМЫ ASPECT.NET ДЛЯ ОБЛАЧНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ MICROSOFT WINDOWS AZURE

А. В. Григорьева,
асп., каф. информатики, мат-мех. факультет

Д. А. Григорьев,
ст. преп. каф. информатики, мат-мех. факультет

В. О. Сафонов,
д.т.н., проф., каф. информатики, мат-мех. факультет

Санкт-Петербургский Государственный Университет

Термином «облачные вычисления» (cloud computing) называется модель исполнения приложений в «облаке» — программно-аппаратной системе, способной динамически управлять размером требуемых приложению ресурсов. Кроме того, облачному приложению доступны параллельная обработка и хранение больших объемов информации, устойчивость к программным и аппаратным сбоям, защищенность от ряда сетевых угроз и т. д. В дополнение к этим возможностям, платформа Microsoft Windows Azure [1] инкапсулирует в себе всю системную сложность оборудования и позволяет разрабатывать программы в рамках привычных концепций Microsoft.NET. Как правило, логика любого серверного приложения может быть перенесена в облако. Облачное приложение для MS Azure должно быть разработано в виде совокупности веб- и прикладных ролей (web- and worker roles). Прикладная роль выполняет длительные вычисления или задания, полученные через специальную очередь (Azure Queue) от веб-роли. Прикладная роль должна быть все время запущена и является аналогом сервиса Windows. Веб-роль принимает запросы клиентов через HTTP-протокол (в виде сайта или веб-сервиса). Для хранения и обработки реляционных данных используется распределенная база данных SQL Azure, в то время как другая информация может храниться в быстродействующем хранилище (Azure Storage). Взаимодействие с внешними распределенными приложениями осуществляется через сервисную шину (Service Bus).

Современные разработчики корпоративных приложений хорошо знакомы с принципами сервис-ориентированной архитектуры (WCF и ASP.NET), что облегчает перенос приложений из собственной инфраструктуры в облако (рис. 1).

На рис. 1 представлены компоненты MS Azure и сценарии их использования корпоративными приложениями [2]. Имеются поставщики, которые через сайт или собственную базу данных взаимодействуют с приложением,

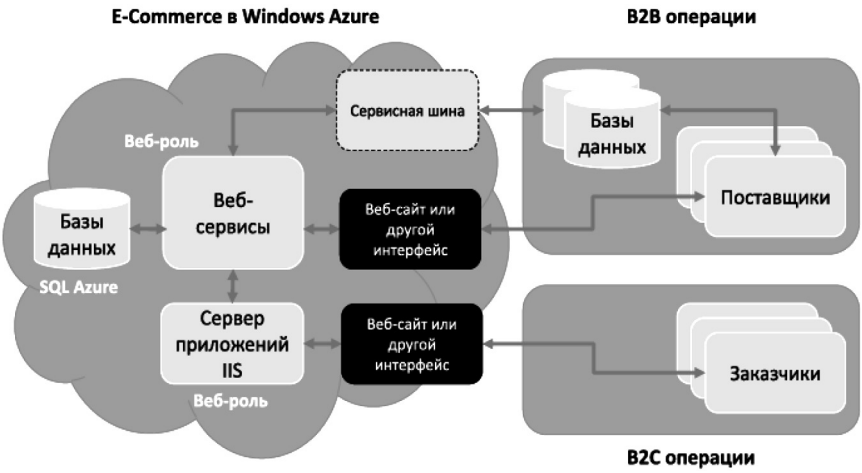


Рис. 1. Архитектура облака

например, для поставки товаров. Имеются заказчики, которые размещают заказы через сайт. Если ожидается, что и тех и других будет много, то имеет смысл реализовать всю инфраструктуру в облаке.

Однако применение объектно-ориентированной методологии программирования привнесло в создание облачных приложений традиционные проблемы сквозной функциональности (cross-cutting concerns), когда в одном логическом компоненте реализуются несколько функциональностей [3]. Например, в веб-службе часто перемешиваются: бизнес-логика, требования безопасности, кэширование, протоколирование и пр. Аспектно-ориентированное программирование (АОП) предлагает решать проблему сквозной функциональности вынесением каждой задачи в свой модуль — аспект. Затем на этапе компиляции или в процессе выполнения целевой программы, действия аспекта вставляются в ее заданные точки, обеспечивая нужное поведение.

Компания Microsoft позиционирует Enterprise Library (EL) [4] как набор наиболее удачных архитектурных решений, применимых в большинстве программных систем. По сравнению с динамическим применением аспектов в EL, когда во время выполнения программы среда .NET создает прокси-объекты в заданных местах, Aspect.NET вплетает действия на уровне MSIL-инструкций после этапа компиляции целевой сборки, что приводит к более высокой производительности целевого приложения. Достоинствами библиотеки EL является встроенная поддержка со стороны среды выполнения .NET, а также возможность настройки аспектов без компиляции через конфигурационные XML-файлы. Однако при работе с облачными вычислениями перекомпиляция для внедрения аспектов не вызывает трудностей, так как развертывание целевой системы осуществляется лишь один раз. Более

того, такая «пост-обработка» дает возможность выбирать конкретные места применения действий аспектов. Для выполнения своих задач библиотека EL ограничена только стандартными технологиями .NET 4.0, т. е. работа со специфическими особенностями платформы Azure (например, со средствами кэширования AppFabric) невозможна. Аспекты Aspect.NET могут работать как с функциями платформы .NET, так и Azure. Наконец, Aspect.NET не требует вносить изменения в исходный код целевого приложения.

В рамках данной работы предложена технология создания аспектов для решения следующих задач облачных приложений:

1. Обработка исключений. Если возникает исключительная ситуация при посылке сообщения другому сервису, аспект перепосылает запрос по тайм-ауту или запасному веб-сервису. Вся служебная информация, в том числе стек отсылается в службу протоколирования Azure. Рассмотрим пример простой веб-службы, работающей в облаке. В виде результата она выдает текст. Мы вызываем ее в обработчике щелчка мыши buttonInvoke_Click по окну формы на нашей локальной машине. Сам принцип применим и для взаимодействия служб в облаке.

```
//Аспект должен быть наследником класса Aspect
public class ExceptionCatcher : Aspect {
     //Задаем правила, куда вставлять это действие
     [AspectAction(«%instead %call *buttonInvoke_Click(…)
     && args(..)»)]
     public static void RepairAction(object sender,
     EventArgs e) {
        int count=0;
        Exception servEx=null;
        while(count++<3) {
               try {
                     //Пытаемся вызвать сервис
                     несколько раз
                     Target.buttonInvoke_Click
                     (sender,e);
               } catch {Exception e) {
                        servEx=e;
               }
        }
        if(servEx!=null) {
               try {
                     //Пытаемся вызвать запасной сервис
                     SimpleServiceGetData2();
               } catch {Exception e) {
```

```
                    System.Diagnostics.Trace("Возникла
                            ошибка: " + servEx.Message);
                }
            }
            public static void SimpleServiceGetData2() {
                SimpleServiceClient2 client = new
                SimpleServiceClient2();
                string result = client.GetData(«Hello!»);
                MessageBox.Show(result);
        }
}
```

2. Протоколирование. Этот аспект отслеживает последовательность вызовов заданных методов и переданные им аргументы. Поскольку код облачных приложений выполняется на закрытой внешней конфигурации, то стандартные отладочные механизмы неприменимы. Это делает задачу аспектного протоколирования особенно актуальной.

3. Аспекты реализации безопасности. Правила внедрения аспекта перехватывают получение сообщения сервисом, и соответствующие действия проверяют сообщения на безопасность (авторизация, распознавание атак и пр.). Также перехватывается отправка или прием сообщения для его шифрования. Аспект работает с Azure Access Control.

4. Кэширование. Облачная инфраструктура имеет широкие возможности по увеличению объема доступной оперативной памяти, что позволяет аспектам повысить общую производительность для сервисов, выполняющих длительные вычисления или выбирающих редко обновляющиеся данные из базы данных. Для работы с кэшем, разделяемым несколькими ролями используется AppFabric. Дополнительная эффективность обеспечивается внедрением аспектов кэширования на стороне клиента (нет лишних «транспортных» расходов), а также через выставление специальных заголовков HTTP-ответа (напр. pragma-cache).

5. Асинхронные операции. Согласно общепринятой практике, длительные операции следует реализовывать в асинхронном виде. Примером таких операций является посылка SMS-уведомлений при протоколировании или проверка на нежелательность введенных комментариев на сайте. Приложение запускает отдельный поток для асинхронной задачи и продолжает свое выполнение. В силу своей обособленности от кода бизнес-логики, асинхронные операции естественным образом могут быть реализованы в виде аспектов. Облачное приложение обычно состоит из нескольких ролей и не исключена ситуация, что экземпляры подобных потоков будут выполняться одновременно, конкурируя за общие ресурсы. При реализации аспектов предлагается избежать накладных расходов на инициализацию потоков и синхронизацию ресурсов путем выделения отдельной прикладной

роли для асинхронных операций. В правилах внедрения аспекта задается точка внедрения действия аспекта с вызовом асинхронной операции. В свою очередь аспект убеждается, что запущена соответствующая прикладная роль, сохраняет необходимые данные для требуемой операции в Azure Storage и ставит ее на обработку в очередь Azure Queue. Прикладная роль аспекта последовательно выбирает из очереди задачи, загружает их данные, исполняет и аналогичным образом уведомляет логику целевого кода о завершении операции. Для выборки результатов завершения асинхронной операции создается второе действие аспекта, перехватывающее обработку сообщений от прикладной роли.

6. Управление конфигурацией MS Azure. С помощью счетчиков производительности платформы аспект постоянно отслеживает текущую загрузку и по необходимости подключает или отключает новые вычислительные узлы, тем самым экономя денежные затраты клиента.

Литература

1. Официальный сайт MS Azure: http://www.microsoft.com/windowsazure/

2. *А. Федоров, Д. Мартынов.* Windows Azure — облачная платформа Microsoft. http://download.microsoft.com/documents/rus/msdn/Windows_Azure_web.pdf

3. *Safonov V. O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008.

4. Официальный сайт Enterprise Library: http://wag.codeplex.com/

РАЗРАБОТКА УНИВЕРСАЛЬНОЙ БИБЛИОТЕКИ АСПЕКТОВ ДЛЯ НАДЕЖНЫХ И БЕЗОПАСНЫХ ВЫЧИСЛЕНИЙ В ПРИЛОЖЕНИЯХ .NET

Д. А. Фролов,
асп., каф. информатики, мат.-мех. факультет

В. О. Сафонов,
д.т.н., проф., каф. информатики, мат.-мех. факультет

Санкт-Петербургский Государственный Университет

Принцип надежных и безопасных вычислений (Trustworthy Computing — TWC) подразумевает построение приложений таким образом, чтобы все исключительные ситуации обрабатывались системой рациональным способом с целью минимизации потерь — простоев системы, потери информации, несанкционированного доступа к информации и т. д. — вследствие системных либо пользовательских ошибок. Начало инициативе TWC было положено письмом президента Microsoft Билла Гейтса в январе 2002 года, направленным всем сотрудникам компании, в котором подчеркивается необходимость поставлять более надежные и безопасные приложения пользователям, в связи с резким возрастанием угроз компьютерным системам. «Нам предстоит выполнить еще много работы, перед тем, как наши компьютеры станут надежными. Необходимо решить много сложных проблем, чтобы наши системы могли самостоятельно восстанавливаться. Безопасность больших сетей — весьма интересная и нетривиальная задача. TWC — не рекламный прием. Это настойчивые усилия сделать будущую безопасность лучше» [1].

В данной работе предлагается реализовать методы TWC с помощью аспектно-ориентированного программирования (АОП) [2]. АОП — активно развивающаяся технология разработки программного обеспечения для модуляризации и использования сквозной функциональности (cross-cutting concerns). Во-первых, использование АОП позволяет определить спецификацию TWC в виде отдельного модуля (аспекта), обеспечивая ясность, наглядность и легкость сопровождения исходного кода программы. Во-вторых, АОП предоставляет возможность автоматического добавления новой сквозной функциональности в код целевых приложений, в отличие от традиционных сред разработки, в которых эти операции приходится выполнять вручную. Наконец, АОП широко поддерживается различными средами разработки ПО, что способствует распространению концепции TWC на практике.

Практической частью работы является библиотека аспектов, осуществляющих типовые проверки в стиле TWC с целью повышения надёжности улучшаемого приложения. Данные аспекты обрабатывают вызовы методов стандартных классов .NET, что позволяет внедрять полученную библио-

теку в различные системы с наименьшей трудоемкостью. Также имеется возможность адаптации библиотеки для конкретных приложений (механизм Enhancement). В качестве АОП-инструмента выбрана система Aspect.NET [3] — языково-независимая среда разработки АОП-приложений для платформы Microsoft.NET, ориентированная на поддержку языков среды Microsoft Visual Studio.

Литература

1. Writing Secure Code / Michael Howard, David LeBlanc. 2nd ed. Redmond, Washington, 2003.

2. *Safonov V. O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008. 338 p.

3. *Safonov V. O., Grigoriev D. A.* Aspect.NET: aspect-oriented programming for Microsoft.NET in practice // .NET Developers Journal. 2005. No. 7. Pp. 28–33.

ГЕНЕРАТОР АНАЛИЗАТОРОВ С ПОДДЕРЖКОЙ НЕОДНОЗНАЧНЫХ АТРИБУТНЫХ EBNF-ГРАММАТИК

С. В. Григорьев, А. С. Лукичёв

Санкт-Петербургский государственный университет

Целью работы является разработка генератора анализаторов для языков, задаваемых произвольными (в том числе неоднозначными) контекстно-свободными грамматиками. В качестве основной области применения данного инструмента предполагается разработка средств автоматизации реинжиниринга программного обеспечения [1].

Одной из главных особенностей задач, решаемых при реинжиниринге, является необходимость часто работать с устаревшими языками, которые имеют большое количество плохо специфицированных диалектов, что затрудняет задание однозначной контекстно-свободной грамматики языка [4]. Это накладывает на инструмент дополнительные требования:

- Необходима поддержка произвольных (в том числе неоднозначных) контекстно-свободных грамматик, так как они являются наиболее распространённым способом задания грамматик в спецификациях языков программирования и управления.
- Необходима поддержка регулярных выражений в правых частях правил, так как их применение очень распространено при формальном описании языков.

Необходимость работы с неоднозначными КС грамматиками определило выбор generalized LR (GLR)-алгоритма для анализа. Работу алгоритма GLR можно рассматривать как параллельное исполнение набора LR-анализаторов. При этом данный набор дополняется процедурой управления магазинами, оптимизирующей их представление путем «склеивания» и «расклеивания», что позволяет хранить и строить параллельные выводы в рамках одного LR-анализатора, лишь при возникновении конфликта добавляя параллельный анализатор.

Оказалось, что такой алгоритм может быть представлен в виде двух взаимно-рекурсивных функций (рекурсивно-восходящий алгоритм, recursive ascent[2, 3]). При этом расклеивание естественным образом реализуется ветвлением в одной из функций, а склеивание — кэшированием результатов.

Реализация такого подхода на функциональном языке программирования весьма наглядна. В этом случае магазины естественным образом заменяются на стек вызовов функций, а кэширование результатов может быть реализовано с помощью замыкания.

Для практического применения инструмента необходима поддержка семантических вычислений. Наиболее распространенным методом реализации

здесь является расширение грамматики атрибутами. Основные требования к алгоритму вычисления атрибутов: корректное вычисление атрибутов с побочными эффектами, поддержка вычислимых и наследуемых атрибутов, поддержка предикатов.

При одновременной поддержке вычислимых и наследуемых атрибутов становится полезным применение функционального программирования, так как возникает необходимость возвращать в качестве значения функцию, которая будет вычислена тогда, когда станут известны её аргументы.

При разработке прототипа системы был выбран такой способ реализации поддержки EBNF-грамматик (грамматики с регулярными выражениями в правых частях правил), при котором не происходит внутреннего преобразования исходной грамматики к не-EBNF виду. Это позволяет исключить излишние манипуляции с магазином, возникающие из-за введения дополнительных нетерминалов. Также результатом работы подобного анализатора является дерево разбора непосредственно в исходной грамматике, что избавляет от необходимости дополнительных преобразований. При этом синтаксис языка спецификаций, используемого в данном инструменте, требует решения задачи разбора последовательности сворачиваемых в магазине грамматических символов в соответствии с регулярным выражением в соответствующем правиле исходной грамматики. Это необходимо для поддержки семантических вычислений. Данную проблему удалось решить, применив метод обратного просмотра траекторий детерминированных конечных автоматов, порождаемых по правилам грамматики.

В рамках данной работы на языке F# [5] был реализован рекурсивно-восходящий алгоритм разбора. Базовый алгоритм был дополнен следующей функциональностью, продиктованной спецификой решаемых в реинжиниринге задач: поддержка неоднозначных КС-грамматик и расширенных КС-грамматик (EBNF) без их преобразования к классическому виду. Также была реализована поддержка l-атрибутных и s-атрибутных грамматик.

Разработка ведётся в рамках проекта кафедры системного программирования Математико-Механического факультета СПбГУ YaccConstructor (https://www.ohloh.net/p/YaccConstructor).

Литература

1. Автоматизированный реинжиниринг программ / Под ред. проф. А. Н. Терехова и А. А. Терехова. СПб.: Изд-во С.-Петербургского университета, 2000. 332 с.

2. *Rene Leermakers.* Non-deterministic Recursive Ascent Parsing. Philips Research Laboratories, P.O.Box 80.000, 5600 JA Eindhoven, The Netherlands.

3. *Larry Morell, David Middleton.* RECURSIVE-ASCENT PARSING. Arkansas Tech University Russellville, Arkansas.

4. *Mark G. J. van den Brand, Alex Sellink, Chris Verhoef*. Current Parsing Techniquesin Software Renovation Considered Harmful // IWPC'98: Proceedings of the 6th International Workshopon Program Comprehension. IEEE Computer Society, Washington, 1998.

5. Дистрибутивы и документация по языку F#: http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/

ИНСТРУМЕНТ РЕИНЖИНИРИНГА ГРАММАТИК

К. А. Улитин,
5 курс, каф. СП, мат.-мех. факультет

Санкт-Петербургский Государственный Университет

Задача генератора синтаксических анализаторов состоит в преобразовании грамматики, заданной на определенном языке, в код синтаксического анализатора или транслятора. Каждое приложение имеет свой язык описания грамматики и алгоритм генерации, на основе которых порождает транслятор, работающий по строго определенному алгоритму. Помимо этого, различные генераторы анализаторов характеризуются классами принимаемых языков, алгоритмами разрешения неоднозначностей, скоростью работы порожденного транслятора, возможностями языка спецификации трансляции, простотой отладки и т. д. Соответственно, с развитием некоторой спецификации трансляции может возникнуть ситуация несоответствия возможностей выбранного на начальном этапе разработки инструмента с текущими требованиями к нему. При этом решение о полном переписывании грамматики для нового инструмента не всегда является приемлемым выбором [4].

Как и в реинжиниринге программного обеспечения, возможны два пути решения данной проблемы. Первый — создание инструмента, позволяющего переиспользовать имеющуюся спецификацию трансляции, то есть исполнение в новом окружении без модификации исходного кода. Второй — перевод грамматики в формат другого инструмента.

Первый подход требует создания достаточно сложного инструмента — генератора трансляторов. В лучшем случае можно обойтись нетривиальной модификацией имеющегося генератора с открытым исходным кодом. Для упрощения решения столь сложной технической задачи был разработан инструмент создания генераторов синтаксических анализаторов YaccConstructor [3]. Но пока выбор алгоритмов порождаемых анализаторов мал, и инструмент недостаточно полно решает поставленную задачу.

Во втором варианте, помимо улучшения характеристик самого генератора, новый язык спецификации трансляций может предоставлять более удобные синтаксические конструкции, благодаря чему результирующая грамматика, возможно, окажется более простой для разработки и сопровождения. Такой подход, в том числе с восстановлением высокоуровневых конструкций, применяется при автоматизированном реинжиниринге программных комплексов [5].

В ходе данного проекта разрабатывается инструмент, на примере которого исследуется применение рассмотренного подхода к задаче трансформации спецификаций трансляций.

Несмотря на недостатки, система YaccConstructor интересна как базовый инструмент исследуемого подхода, поскольку позволяет комбинировать различные языки описания трансляций с различными генераторами. Такая гибкость достигается за счет архитектуры, построенной вокруг унифицированного внутреннего представления спецификации трансляции, с которым по описанным интерфейсам работают парсеры файла с грамматикой (Frontends), преобразования (Conversions) и генераторы (Generators). При первом случае преобразования раскрывают сложные конструкции языка описания трансляции, такие как макроправила и конструкции РБНФ, а генератор выдает код транслятора на некотором языке программирования [1].

Представляемый инструмент разрабатывается как набор front-end'ов и генераторов, и позволяет транслировать грамматику из формата одного инструмента в формат другого. Front-end'ы разрабатываются также с помощью YaccConstructor. Архитектура инструмента изображена на рис. 1:

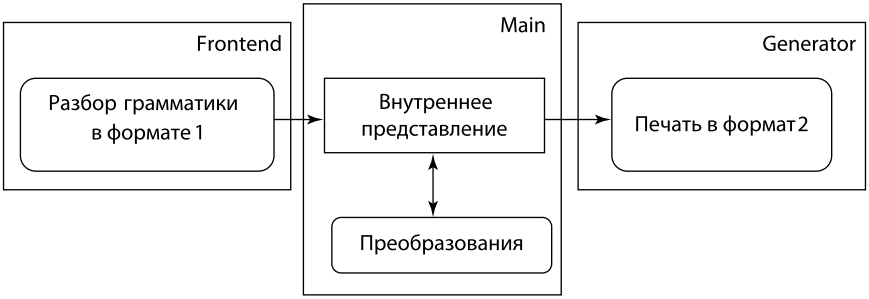


Рис. 1. Архитектура с истемы YaccConstructor

При трансляции из формата одного инструмента в формат другого используются соответствующие преобразования форматов грамматик, генератор выводит результирующую грамматику. Одни и те же преобразования применяются для трансляций между различными форматами, так как многие из них поддерживают сходные конструкции, такие как расширенные регулярные выражения, вложенные альтернативы и др.

Разработку подобных инструментов удобно выполнять для платформы .NET, так как в ней имеется возможность реализовывать разные компоненты на более подходящих для задачи языках, предоставляется богатый набор различных библиотек, удобная среда разработки и сборки проектов.

На данный момент приложение внедряется в проект Claret [2], для которой разрабатывается парсер на основе грамматики C, взятой из открытого источника. Исходный код и дополнительная информация по проекту представлены на сайте проекта [3].

Литература

1. *Улитин К. А.* Модульный генератор синтаксических анализаторов // Сборник материалов конференция «Технологии Microsoft в теории и практике программирования». 2011.

2. http://code.google.com/p/claret/

3. http://code.google.com/p/recursive-ascent/

4. *Klint P., Lammel R., Verhoef C.* Toward an engineering discipline for GRAMMARWARE. 2003.

5. Автоматизированный реинжиниринг программ // Под ред. проф. А. Н. Терехова и А. А. Терехова. СПб.: Изд-во С.-Петербургского университета, 2000. 332 с.

ОБУЧЕНИЕ ПОСТРОЕНИЮ ТРАНСЛЯТОРОВ: ТЕОРИЯ И ПРАКТИКА

Д. В. Луцив (e-mail: dluciv@math.spbu.ru),

В. С. Полозов (e-mail: victorp@math.spbu.ru)

Санкт-Петербургский Государственный Университет

Аннотация. В работе рассмотрены проблемы обучения теоретическим и практическим аспектам реализации языков программирования и трансляторов на математико-механическом факультете СПбГУ. Предложены учебные языки и трансляторы для них. Описан контекст применения данных трансляторов при обучении студентов специальностей и направлений отделения информатики.

Введение, описание проблемы

Любой сотрудник коммерческой фирмы, исследователь и преподаватель в области информационных технологий сталкивается с трансляторами практически каждый день. Среды программирования, СУБД и даже офисные пакеты содержат в себе встроенные языки, которыми пользуются программисты и квалифицированные пользователи. Обладая достаточным профессионализмом, многие исследователи и архитекторы ПО создают собственные предметно-ориентированные языки.

С трансляторами, пусть и не осознавая этого, сталкиваются почти все пользователи вычислительных средств. Работа в практически любой операционной системе общего назначения как минимум сопряжена со смешанным (включающим интерпретацию и компиляцию) исполнением большого количества сценариев, даже если оператор использует только веб-браузер.

В университетах и технических ВУЗах обучению архитектуре и использованию трансляторов уделяется должное внимание.

В работе описано, каким образом могут быть заложены базовые теоретические знания и начальные навыки в области самостоятельной реализации трансляторов.

Ниже подробнее рассмотрено обучение построению трансляторов в учебных программах математико-механического факультета СПбГУ с участием авторов и описаны два учебных прототипа трансляторов, успешно используемых при обучении студентов специальности 010503 [6] и направления 010400 [5].

Особенности образовательных программ

При организации учебных курсов авторы работы часто сталкиваются с тем, что теоретические дисциплины, посвящённые основам информатики и преподающиеся на младших курсах, либо не покрывают вопросы внутренне-

го устройства трансляторов вообще, либо покрывают в незначительной мере. Такая ситуация вполне нормальна. Опыт подсказывает, что не менее важные вопросы, касающиеся разработки архитектуры ПО и организации его производства часто излагаются не на 1–2 курсах, а на 3–5. На младших курсах же закладываются базовые понятия информатики, которые в значительной степени помогают студентам изучить общую систему профессиональных понятий.

Так же, как и подробное описание архитектуры ПО и технологических процессов начинается на старших курсах, студентам специальности 010503 [6] и направления 010400 [5] на 3-м курсе предлагается прослушать лекции по теории языков и трансляций. По хорошей университетской традиции этот лекционный курс позиционируется как математический, и изобилует формальными построениями. Следом за этим курсом студентам предлагается прослушать ряд специальных дисциплин, посвящённых углублённому изучению различных аспектов работы трансляторов. Уровень сложности этих дисциплин достаточно высок и, если выбирать из известных источников, соответствует приблизительно уровню книг [1, 3, 9, 10]. Авторы работы считают, что предварительная вводная информация, подкреплённая практическими навыками, позволит гораздо лучше воспринимать материалы последующих углублённых курсов [11, 12].

Примеры языков, модули трансляторов и сопутствующее ПО

Для обоих приведённых ниже примеров при обучении предполагается, что студенты:

- Владеют теоретическими знаниями курса «Основы информатики» приблизительно в объёме двух академических лет (2 семестра по 2 лекции в неделю), т. е. имеют представления о составлении алгоритмов, императивных алгоритмических языках, объектно-ориентированном и обобщённом программировании.
- Владеют практически навыками программирования на основе практических занятий, полученными в объёме не менее 1,5 академических лет (I и II семестры по 2 и 1 пары в неделю соответственно), т. е. программируют при помощи сред разработки и без них, умеют адекватно реализовывать предложенные алгоритмы, проектировать структуры данных, отлаживать программы, разрабатывать модульное П.О.
- Дополнительно требуется, чтобы студенты достаточно легко пользовались платформами .NET или Java. Эти платформы обладают богатыми возможностями, имея при этом достаточно высокую доступность и низкий порог вхождения. Их использование типично, начиная со второго семестра обучения и утверждено в учебных программах [8].
- Не лишним, хотя и не обязательным, будут начальные представления об устройстве трансляторов, которые студенты могут почерпнуть из вводных частей [1] и соответствующих глав [2].

$L^{A}N_{G}7$

Данный экспериментальный учебный язык был разработан летом 2007 году и осенью 2007 же года успешно использован для обучения студентов специальности 010503 [6] и повторно был использован в 2009 году.

На данный момент язык содержит единственный тип данных — число с плавающей точкой — и формально является строго статически типизированным. Добавление прочих типов данных не должно существенно повлиять на модель языка.

Синтаксис языка напоминает синтаксис Паскаля.

Транслятор языка на выходе выдаёт программы на Си, Паскале, C#. Многие технологические решения подразумевают высокоуровневые языки на выходе трансляторов [4], поэтому данных подход актуален. В будущем планируется выдавать также код для стековой виртуальной машины.

Транслятор представляет собой совокупность библиотек общих интерфейсов и структур данных, которые используются в следующих реализуемых студентами модулях:

- лексический анализатор (на основе регулярных выражений);
- синтаксический анализатор класса LL(1);
- оптимизатор;
- несколько модулей генерации кода.

Кроме того, для каждого из возможных порождений реализуется небольшая библиотека времени выполнения.

На данный момент проект реализован на платформе .NET 3.5 на языке C# 3.5. Если этого требует учебная программа, он без принципиальных изменений может быть реализован на Java.

Страница проекта: http://edu.dluciv.name/lang7

StudentML

Основным заданием на четвертый семестр обучения студентов направления 010400 [5] является разработка функционального языка StudentML. Учебное задание на семестр представляет собой цепочку заданий, начиная от простейшего разбора арифметических выражений методом рекурсивного спуска и текстового редактора, до языка со статической типизацией и интегрированной средой разработки. Таким образом, от студентов требуется свободное владение языком программирования (Java), навыки объектной декомпозиции задачи, знакомство с понятием операционной семантики, умение проектировать программы с графическим пользовательским интерфейсом, знакомство с основными шаблонами объектно-ориентированного проектирования. Задание позволяет отработать как индивидуальные навыки проектирования и программирования, так и групповую разработку, что является необходимым навыком для бакалавров в области программной инженерии.

Также для многих студентов это первый опыт исследовательской работы в рамках предусмотренной учебным планом курсовой работы. Студентам на выбор предлагаются такие темы исследований, как:

- сравнение эффективности отладчиков для итеративного и рекурсивного компиляторов;
- отладчик с «откатом» состояния;
- введение статической типизации [9] (по Чёрчу, автоматический вывод типов, полиморфные типы);
- реализация автоматизированных групповых преобразований кода в редакторе интегрированной среды (рефакторинг).

Первая версия языка с интегрированной средой разработки была реализована студентами весной 2008 года в качестве задания по выбору. Начиная с 2010 года, задание успешно используется для обучения студентов направления 010400 [5].

Страница проекта: http://edu.vpolozov.name/studentml

StudentAsm

В качестве одного из упражнений во втором семестре первого курса обучения по направлению 010400 [5] студентам предлагается реализовать виртуальную машину на основе достаточно простого стекового ассемблера. От студентов требуются минимальные навыки алгоритмического мышления и декомпозиции задач.

Задача предлагается в двух вариантах: для реализации на языке C, который изучается студентами в течении первого семестра, или как упражнение для реализации на языке Haskell. Благодаря «чистоте» языка Haskell решение подразумевает явное выделение состояния и операционной семантики команд, что показывает связь формальных систем с прикладным программированием.

Страница проекта: http://edu.vpolozov.name/studentasm

Обеспечение занятий

Проекты реализуются при помощи следующих инструментальных средств:

- Java SDK Standard Edition, NetBeans IDE — StudentML;
- Platform Haskell, WinHugs, gcc — StudentAsm;
- .NET SDK, MS Visual Studio 2008+ — Lang7;
- Subversion — все.

Во многих ВУЗах Windows, .NET SDK и MS Visual Studio доступны бесплатно по академическим лицензиям. Тем не менее, транслятор и библиотеки Lang7 нормально работают при помощи свободной реализации платформы .NET — Mono, в т.ч. в UNIX-подобных операционных системах, а проект без проблем редактируется в среде MonoDevelop.

Заключение

В работе описаны примеры учебных трансляторов, виртуальной машины и вспомогательного программного обеспечения, предназначенных для обучения студентов проектированию и реализации трансляторов и приобретения ими практических навыков на эту тему. Имея практический опыт при реализации простых трансляторов, легче вникнуть в детали теории языков и трансляций. Реализация виртуальных машин позволяет студентам лучше прочувствовать важность взаимосвязи между программным и аппаратным обеспечением [4].

Выполняя предложенные упражнения по реализации трансляторов и вспомогательного ПО, студенты учатся следовать предложенным спецификациям, разрабатывать модульное ПО, работать в коллективе. Это важно при дальнейшем обучении и в промышленном программировании.

Выработанный положительный опыт учтён в рабочих учебных программах [8] специальности 010503 [6].

Альтернативные подходы к обучению компиляторам на мат-мехе основаны на курсах и книгах Н. Н. Вояковской и В. О. Сафонова [10–12].

Литература

1. *Альфред Ахо, Рави Сети, Джеффри Ульман.* Компиляторы: принципы, технологии и инструменты. «Вильямс», 2007.

2. *Вирт Н.* Алгоритмы + структуры данных = программы. М.: Мир, 1985.

3. *Steven S. Muchnick.* Advanced compiler design and implementation. Morgan Kaufmann, 1997.

4. *Терехов А. Н.* Технология программирования. М.: Изд-во «Интернет-университет информационных технологий — ИНТУИТ.ру», 2006.

5. Учебный план. Код направления: 010400 «Информационные технологии». Квалификация: бакалавр информационных технологий. http://www.math.spbu.ru/ru/mmeh/PLANS/plan010400_bac.doc

6. Государственный образовательный стандарт высшего профессионального образования. Специальность 351500 «Математическое обеспечение и администрирование информационных систем». http://www.math.spbu.ru/ru/mmeh/Gost/351500.html

7. Федеральный государственный образовательный стандарт высшего профессионального образования по направлению подготовки 231000 Программная инженерия (квалификация (степень) «бакалавр»). http://www.edu.ru/db-mon/mo/Data/d_09/prm542-1.pdf

8. Рабочая программа учебной дисциплины «Программирование» подготовки по специальности 010503 «Математическое обеспечение и администрирование информационных систем». Федеральное агентство по образованию Российской Федерации. Санкт-Петербургский государственный университет. Математико-механический факультет, 2010.

9. *Benjamin C. Pierce.* Types and Programming Languages. The MIT Press, 2002.

10. *Safonov V. O.* Trustworthy Compilers // Wiley International. John Wiley & Sons, 2010. 295 pp.

11. *Вояковская Н. Н.* Трансляция языков программирования. Теоретический курс направления 010400 «Информационные технологии».

12. *Сафонов В. О.* Компиляторы. Теоретический курс специальности 010503 «Математическое обеспечение и администрирование информационных систем». https://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=7698&Login=

КОНТРОЛЬ УЧАЩИХСЯ НА ОСНОВЕ ИНТЕЛЛЕКТУАЛЬНЫХ ГРИДОВ

В. К. Толстых, А. Ю. Кожемякин

Донецкий национальный университет

В настоящее время используется и разрабатывается значительное число программных продуктов для информатизации процессов образования, в частности, — электронных журналов преподавателей. Однако ситуация с информатизацией образовательных учреждений по-прежнему далека от желаемой. Главная причина — отсутствие единого информационного образовательного пространства (ЕИОП). На кафедре компьютерных технологий физического факультета Донецкого национального университета разрабатываются технологии и программное обеспечение ЕИОП на основе единой регистрационной базы данных (ЕРБД) об учащихся, преподавателях и других работниках образования с применением сервис-ориентированной архитектуры для автоматизированного доступа к этим данным со стороны Internet-клиентов.

Система функционирует в виде «вычислительного облака успеваемости», построенного в рамках концепций грид, на основе сервис-ориентированной архитектуры (рис. 1). На рис. 1 узлы облака представляют собой серверы организаций участников (школ, ВУЗов, районных, городских и областных органов образования). Каждый узел имеет свою локальную базу регистрационных данных контингента (фамилия, имя, отчество, дата рождения, и.т. п.) организации и оригинальных внутренних данных (оценок, отчетов, предметов, обязанностей, и т. п.). Доступ программного обеспечения пользователей облака успеваемости осуществляется через главный узел системы, который на основании данных авторизации пользователя и его запроса производит обращение к требуемому узлу, формирует из полученных данных ответ и возвращает его пользователю. В качестве ответа могут быть как запрошенные данные из «облака», так и подробное сообщение об ошибке доступа.

Контроль производится по трем критериям: посещаемость занятий, поведение учащихся на занятиях, оценки, которые они получают. На грид-узле учебного заведения ответственное лицо вносит данные по каждому из критериев относительно всех учащихся на каждом занятии, или в конце каждой недели, или в конце семестра в локальную базу данных узла грид-системы.

Контроль осуществляется в двух видах — ручном и полуавтоматическом:

- ручной режим предполагает подключение личности к гриду с помощью web-браузера или программного клиента для просмотра фактических (текущих) данных успеваемости, посещаемости и поведения учащегося. В качестве личности в данном случае могут выступать сами учащиеся,

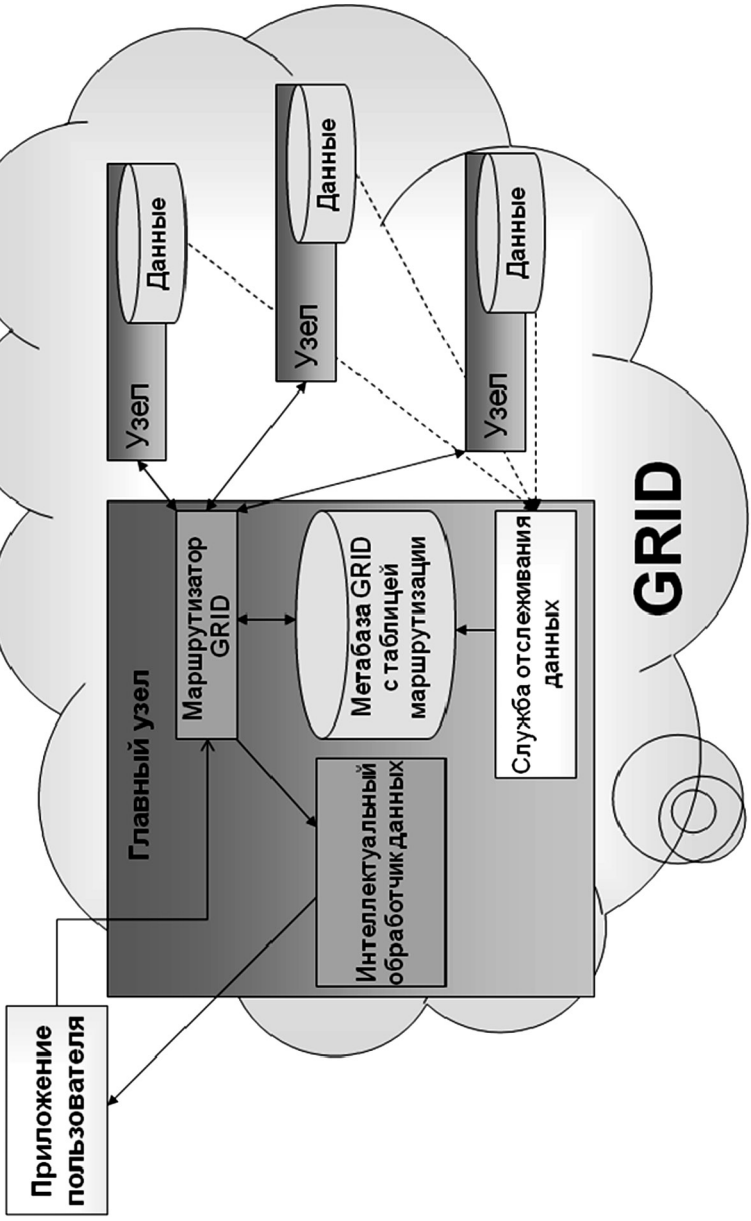


Рис. 1. Структура облака успеваемости

их родители или опекуны, преподаватели, классные руководители и администрация школы.

- полуавтоматический режим (интеллектуальный), в общем случае, предполагает автоматизированный сбор данных из облака службами грида о подчиненных узлах, генерацию интегральных статистических отчетов, их передачу вышестоящему узлу, определение рейтинга, как индивидуального учащегося, так и подразделений и заведений в целом, анализ данных для оценки качества образовательного процесса, для прогнозирования ситуаций и рекомендаций по принятию различных решений.

На грид-узлах учебных заведений система анализирует среднюю успеваемость учащихся по классу (группе), предмету, учебному заведению. После этого информирует (посредством SMS, e-mail, на страницах web-интерфейса облака успеваемости) всех заинтересованных участников образовательного процесса о недостаточной успеваемости, низкой посещаемости или неудовлетворительном поведении учащегося.

Разрабатываемая система упрощает процесс социальной коммуникации между родителями, опекунами учащихся и преподавательским составом учебного заведения, а так же способствует всем им в принятии оперативных и адекватных мер для обеспечения качественного образовательного и воспитательного процесса в учебном заведении.

Разработка всей системы осуществляется в среде Microsoft .NET Framework. Для разработки Web-служб используются технологии Microsoft Windows Communication Foundation. Выбор обоснован тем, что программные продукты, написанные с использованием этой технологии, имеют высокий уровень безопасности как внутренней, так и при Web-взаимодействии с программным обеспечением пользователей облака успеваемости. Взаимодействие осуществляется при помощи XML с автоматическим контролем доставки данных, высоким уровнем отказоустойчивости и, при необходимости, с поддержкой транзакций.

ИСПОЛЬЗОВАНИЕ ЕДИНОЙ РЕГИСТРАЦИОННОЙ БАЗЫ ДАННЫХ ГРАЖДАН

В. К. Толстых, Л. Н. Киселёва

Донецкий национальный университет

На сегодняшний день каждая организация, регистрирующая данные о своих сотрудниках, самостоятельно производит сбор необходимой информации и локальное хранение, что сопровождается рутинными, затратными операциями, повторяющимися в каждой новой организации, в которую будущий сотрудник подаёт свои документы. В данной ситуации создание и совместная поддержка единой регистрационной базы данных (ЕРБД) граждан в Web-пространстве всеми заинтересованными организациями представляется актуальным. Это позволит снизить организационные и финансовые затраты всех участников процесса регистрации и поддерживать данные в максимально актуальном состоянии [1, 2].

В настоящей работе рассматривается совместное использование ЕРБД различными организациями, официально подключившимися к данной базе (рис. 1). ЕРБД через соответствующий Web-сайт позволяет организациям (с согласия гражданина — личности ЕРБД) просматривать, создавать и модифицировать регистрационные данные граждан. Посредством Web-сервисов данные ЕРБД, созданные в т.ч. и другими организациями, могут копироваться в локальные базы организаций.

В результате анализа бизнес-процессов ЕРБД, в каждой организации выделены следующие группы пользователей, участвующие в поддержке и использовании данных ЕРБД:

1. Регистраторы — создают, модифицируют данные ЕРБД, копируют данные из ЕРБД в локальные базы данных;
2. Аналитики — формируют интегрированные отчеты, выполняют E-mail рассылки;
3. Инспектора — просматривают данные граждан, формируют аналитические отчеты, контролируют работу регистраторов и аналитиков организации;
4. Личности — контролируют корректность своих данных через Internet, модифицируют свои контактные данные.

Регистратор, на основе данных свидетельства о рождении и пароля, предоставленных личностью, зарегистрированной в ЕРБД, получает доступ к данным этой личности в Web-приложении «Просмотр и модификация» (рис. 1). На основании предоставленных личностью документов, контролирует данные, при необходимости модифицирует, дополняет их, и сохраняет в ЕРБД. С помощью соответствующего Web-сервиса ЕРБД, при необходи-

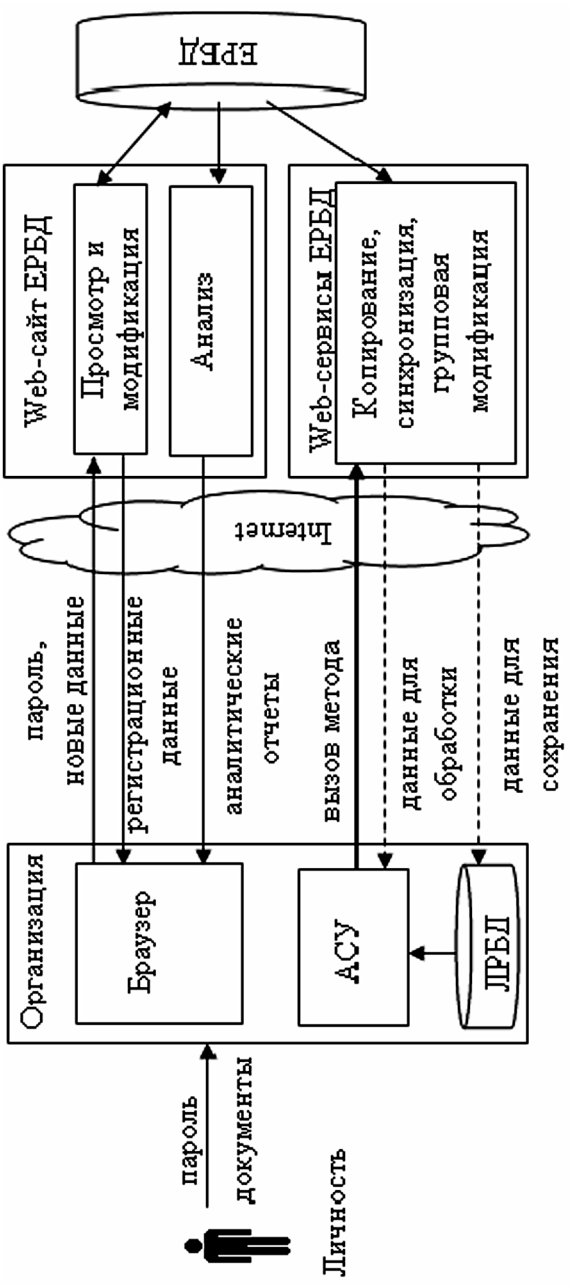


Рис. 1. Совместное использование организациями данных ЕРБД

мости, копирует данные в локальные регистрационные базы данных организации.

Аналитик, в Web-приложении «Анализ», создает аналитические отчеты на основании данных личностей зарегистрированных в его организации или в подчиненных организациях.

Инспектор, аналогично регистратору, получает доступ к Web-приложению «Просмотр и модификация», но только с правом просмотра данных и аналогично аналитику к Web-приложению «Анализ» для создания отчетов.

Кроме того, автоматизированные системы управления (АСУ) организаций могут осуществлять авторизированное подключение к ЕРБД через Web-сервисы ЕРБД. АСУ организаций, с помощью вызова различных Web-сервисов, могут получать необходимые данные ЕРБД в соответствии с их правами доступа к тем или иным личностям ЕРБД. АСУ могут копировать данные для обработки в реальном времени, для хранения в локальных базах организаций, для синхронизации данных локальных баз с ЕРБД. Кроме того, для вузов актуальны задачи групповых зачислений, переводов с курса на курс и отчислений студентов. Данные задачи в ЕРБД также могут решаться автоматизировано с использованием соответствующих Web-сервисов.

ЕРБД, её Web-сайт и Web-сервисы разрабатывается на платформе Windows Server 2008 с использованием технологий ASP.NET. Web-сервисы реализуются как сервисы Windows Communication Foundation, что позволяет относительно просто создавать надёжные, многофункциональные, транзакционные службы в Web-пространстве.

Литература

1. *Толстых В. К.* О единой регистрационной базе данных в информационно-образовательном пространстве / В. К. Толстых, Л. Н. Киселева // Инфо-Стратегия 2010. UA: Общество. Государство. Образование. Сборник материалов II Международной научно-практической конференции. Севастополь, 2010. С. 69–71.

2. *Толстых В. К.* Единая регистрационная Web-база граждан / В. К. Толстых, Л. Н. Киселева // Комп'ютерні системи та мережні технології (CSNT-2010). Збірник тез III Міжнародної науково-технічної конференції. Київ: НАУ-друк, 2010. С. 99.

Рандомизированные алгоритмы оптимизации, оценивания и кластеризации

Граничин
Олег Николаевич

д.ф.-м.н.
профессор кафедры системного программирования СПбГУ
заведующий лабораторией
стохастических вычислительных систем НИИИТ СПбГУ

НЕСТАЦИОНАРНЫЙ СЛУЧАЙ В ЗАДАЧЕ ДОСТИЖЕНИЯ КОНСЕНСУСА В СЕТИ ПРИ НЕПОЛНОЙ ИНФОРМАЦИИ

Н. О. Амелина (E-mail: ngranichina@gmail.com)

Санкт-Петербургский Государственный Университет, аспирант

В статье рассматривается задача балансировки загрузки узлов децентрализованной вычислительной сети при неполной информации о текущих состояниях узлов и переменной структуре связей. Задача балансировки загрузки переформулируется как проблема достижения консенсуса в условиях переменной топологии. Для решения предлагается использовать алгоритм типа стохастической аппроксимации, работоспособность которого иллюстрируется примерами имитационного моделирования.

Распределенное взаимодействие в сетях динамических управляемых агентов привлекает в последние время внимание все большего числа исследователей. Во многом это объясняется широким применением мультиагентных систем в разных областях, включая автоматическую подстройку параметров нейронных сетей распознавания, управление формациями [1], роение [2], распределенные сенсорные сети [3], управление перегрузкой в сетях связи [4], взаимодействие групп беспилотных летательных аппаратов (БПЛА) [5], относительное выравнивание групп спутников и др. Многие из таких задач легко переформулируются в терминах достижения консенсуса в мультиагентных системах [6–8].

Для группы взаимодействующих агентов, обменивающихся с задержкой неполной информацией в дискретные моменты времени, при изменяющейся топологии связей в [9] предложен и обоснован алгоритм стохастической аппроксимации для решения задачи достижении консенсуса. Стохастическая аппроксимация с убывающим размером шага позволяет каждому агенту получать информацию о состоянии своих соседей при одновременном снижении воздействия помех.

В последнее время все чаще при вычислениях используются распределенные системы параллельных вычислений, для которых актуальна задача разделения пакета заданий между несколькими вычислительными устройствами.

Рассмотрим модель системы разделения однотипных заданий между разными узлами для параллельных вычислений с обратной связью. Обозначим $N\{1, ..., n\}$ набор интеллектуальных агентов (вычислительных узлов), каждый из которых обслуживает поступающие заявки на вычисления по принципу очереди, то есть первый вошел — первый вышел. Будем считать, что всем агентам присылают однотипные задания, которые можно раздробить на одинаковые по сложности атомарные единицы. Задания поступают в различные моменты

времени и на разные узлы. Считается известным размер каждой задачи (или задания), то есть его трудоемкость, в секундах или циклах процессора.

В каждый момент времени t состояние агента i, $i = 1, ..., n$, описывается двумя характеристиками:

1) q_t^i — длина очереди из атомарных элементарных заданий в момент времени t;
2) p_t^i — производительность узла (количество (или доля) выполненых атомарных заданий за предшествующий такт времени, при условии полной загрузки).

Обозначим $x_t^i = q_t^i / p_t^i$.

Задача для сети агентов — выполнять поступающие последовательно задания. Будем рассматривать две постановки задачи: стационарную и нестационарную.

1. Для начала предположим, что все задания поступают в систему на разные узлы в начальный момент времени. Если все задания выполняются только тем агентом, которому они поступили, и производительности агентов не изменяются со временем, то время выполнения всех заданий определяется так: $T_m = \max_i q_0^i / p_0^i$. Для минимизации времени выполнния всех заданий при возможности их перераспределения между агентами естественно рассматривать задачу оптимизации загрузки сети.

2. При тех же предположениях задача оптимизации загрузки остается актуальной и при поступлении в сеть новых заданий с течением времени. Новые задачи могут поступить прямо на любой из n узлов. Каждый узел i в определенный момент времени t «видит» только соседей из множества N_t^i. При этом считаем, граф связности полный, т. е. из каждого узла существует цепочка в любой другой. Динамики изменений каждого из узлов при неизменных производительностях описываются следующими уравнениями

$$\begin{aligned} \dot{q}_t^i &= -p_t^i + u_t^i; \\ \dot{p}_t^i &= 0. \end{aligned} \tag{6}$$

В каждый момент времени t узел i может получить от своих «видимых» соседей $j \in N_t^i$ следующую информацию:

1) наблюдения о том, какова его загрузка $\overline{y}_t^{ij}$;
2) p_t^j — производительность узла.

Задача состоит в том, чтобы составить протокол общения между агентами, при котором все узлы будут загружены равномерно, т. е. $q_t^i / p_t^i \to c_t^*$, независящей от i, т. е. если в систему не будут поступать новые заказы, то все узлы закончат работать одновременно.

Для поставленной задачи будем использовать алгоритм стохастической аппроксимации для достижения консенсуса в следующем виде:

$$x_{t+1}^i = x_t^i + \alpha_t \sum_{j \in N_t^i} y_t^j / p_t^j - x_t^i / p_t^i, \tag{17}$$

где α_t — последовательность положительно определенных размеров шагов, x_t^i — состояние узла i в момент времени t, p_t^i — производительность узла i.

Моделирование работы алгоритма в нестационарном случае, т. е. когда в процессе работы в систему поступает новые заказы, приведено на рис. 1. Используется постоянный размер шага $\alpha_t = 0.1$.

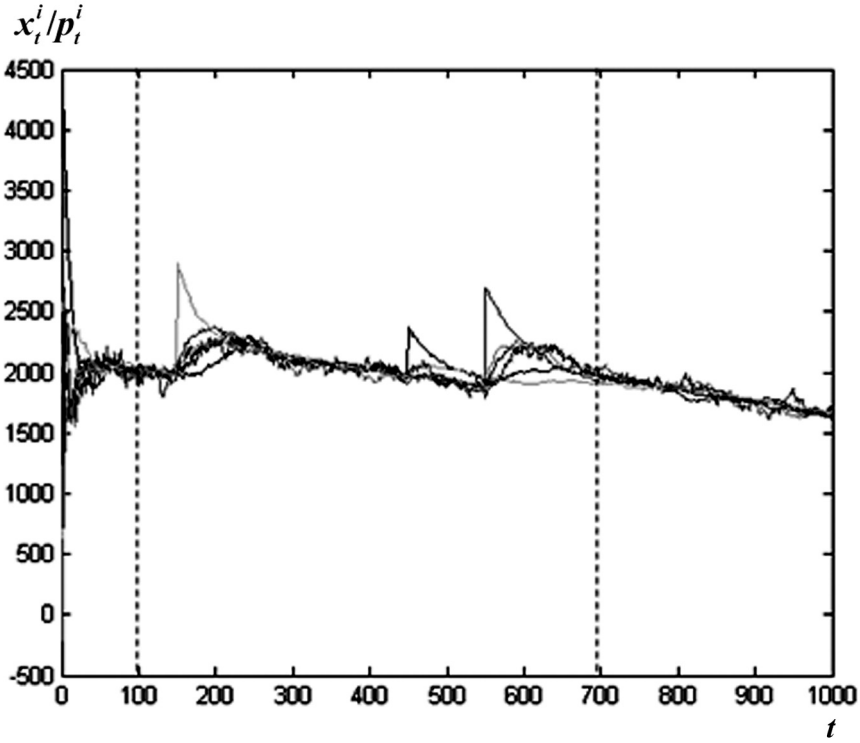


Рис. 1. График нормированных состояний системы для нестационарного случая

В дальнейшем планируется исследование и дальнейший анализ алгоритма при влиянии разного типа помех. Также будут предприняты попытки усовершенствования алгоритма для возможности его использования в случае наличия систематичской погрешности в измерениях.

Литература

1. *Fax, R. M. Murray.* Information flow and cooperative control of vehicle formations — IEEE Trans. Automat. Contr., vol. 49, pp. 1465–1476, Sept. 2004.

2. *J. Toner, Y. Tu.* Flocks, herds, and schools: a quantitative theory of flocking — Phys. Rev. E, vol. 58, no. 4, pp. 4828–4858, Oct. 1998.

3. *J. Cortes, F. Bullo.* Coordination and geometric optimization via distributed dynamical systems — SIAM J. Control Optim., May 2003.

4. *F. Paganini, J. Doyle, S. Low.* Scalable laws for stable network congestion control — presented at the Int. Conf. Decision and Control, Orlando, FL, Dec. 2001.

5. *C. Antal, O. Granichin* and *S. Levi.* Adaptive autonomous soaring of multiple UAVs using SPSA — In Proc. of the 49th IEEE CDC, Atlanta, GA. USA. P. 3656–3661, 2010.

6. *Jadbabaie, J. Lin*, and *A. S. Morse*. Coordination of groups of mobile autonomous agents using nearest neighbor rules — IEEE Trans. Automat. Contr., vol. 48, pp. 988–1000, June 2003.

7. *R. Olfati-Saber* and *R. M. Murray.* Consensus problems in networks of agents with switching topology and time-delays — IEEE Trans. Automatic Control, vol. 49, pp. 1520–1533, Sep., 2004.

8. *W. Ren* and *R. W. Beard.* Consensus seeking in multiagent systems under dynamically changing interaction topologies — IEEE Trans. Automat. Control, vol. 50, no. 5, pp. 655–661, 2005.

9. *M. Huang*. Stochastic Approximation for Consensus with General Time-Varying Weight Matrices — Proc. the 49th IEEE CDC, Atlanta, GA, USA, pp. 7449–7454, Dec. 2010.

ЗАДАЧА АВТОМАТИЧЕСКОГО СЛИЯНИЯ И МОДЕЛЬ СЛУЧАЙНОГО МАРКОВСКОГО ПОЛЯ В СИСТЕМАХ КОНТРОЛЯ ВЕРСИЙ

Д. В. Павленко (e-mail: dmit10@gmail.com)

Санкт-Петербургский Государственный Университет, аспирант

1. Введение

В последние годы наблюдается массовая разработка программного обеспечения. В связи с этим актуальными являются задачи, связанные с облегчением процесса разработки для получения его большей производительности и управляемости. В качестве примера таких задач можно привести: задача отслеживания ошибок (bug tracking), задача статического анализа кода, задача резервного копирования и контроля версий программного продукта (version control) и другие. В данном докладе мы рассмотрим последнюю задачу поподробнее.

Задача контроля версий заключается в обеспечении надёжного хранения и пересылки файлов исходного кода, их промежуточных состояний (называемых ревизиями или коммитами), сравнении этих состояний и обнаружении изменений, обеспечении взаимодействия нескольких разработчиков, разграничении доступа. В настоящий момент наиболее популярными системами контроля версий (VCS) являются Subversion [1], Git [2] и Mercurial [3].

Все существующие системы контроля версий можно условно поделить на централизованные и децентрализованные. Централизованные системы (например, CVS,

Subversion характеризуются тем, что вся история изменений (так называемый репозиторий) хранится на сервере, а на клиентской машине разработчика находится лишь «снимок» проекта определённой ревизии. Для проведения большинства операций, в т.ч. для создания новой ревизии, клиентское приложение делаем запросы к серверу.

В последнее время, однако, набирают популярность так называемые, децентрализованные системы контроля версий (например, Git и Mercurial). Их основное отличие заключается в том, что работа с ними начинается с полного клонирования всей истории проекта на клиентскую машину, в результате чего у пользователя локально появляется полноценный, полнофункциональный репозиторий, в результате чего все операции производятся локально. Популярность таких систем обеспечивается преимуществами:

- более высокая скорость операций;
- возможность создавать коммиты/ревизии вне зависимости от наличия соединения с сервером;
- отсутствие необходимости в системе резервного копирования.

Однако, несмотря на наличие на сегодняшний день множества решений на рынке систем контроля версий, остаётся актуальной проблема обеспечивания взаимодействия между пользователями, вносящими изменения в один и тот же проект.

Предположим, что два пользователя *A* и *B* хотят внести исправления в один и тот же файл проекта. Существует два основных подхода, организующих их взаимодействие:

- метод блокировок;
- метод слияния.

Первый заключается в том, что пользователь A перед изменением файла выставляет на нём блокировку, в результате чего пользователь B не сможет создавать коммиты со своими изменениями к этому файлу до тех пор, пока пользователь A не снимет блокировку. Проблемой этого подхода является то, что пользователь A может забыть снять блокировку или у него может пропасть соединение, в итоге вся работа над проектом может быть заморожена. В связи с этим наиболее популярен на сегодня другой подход, заключающийся в том, что пользователи могут изменить и один и тот же файл, однако при попытке отправки своих изменений на сервер, фиксируются изменения только первого отправившего пользователя, остальные же для отправки своих изменений должны скачать с сервера изменения первого пользователя и произвести сляние (merge) со своими изменениями. Если эти изменения не пересекаются, то такое слияние может быть выполнено автоматически. В противном случае слияние приходится производить вручную.

Такая ситуация очень часто встречается. Причиной тому: появление всё более мощных инструментов разработки, возможности которых вышли далеко за пределы текстовых редакторов, в результате чего существенные изменения файлов производятся просто комбинацией клавиш (переформатирование кода, рефакторинги), иногда даже неявно (генерации, оптимизация и переупорядочивание import-деклараций в Java средах).

Проведение слияния — достаточно трудоёмкая операция. Она производится на основе трёх состояний проекта: базовой версии (версии до изменений), версии после изменения первым пользователем и версии после изменения вторым пользователем. Для решения задачи автоматического слияния нужно решить сперва задачу сопоставления.

2. Постановка задач

2.1. Задача сопоставления

Задача сопоставления состоит в поиске соответствующих друг другу структурных элементов двух различных версий проекта.

Обычно в системах контроля версий производится сопоставление строк.

Поскольку при разработке программного обеспечения происходит работа со структурированными текстами, представимыми в виде дерева синтаксических элементов (AST), то вместо сопоставления строк можно сопоставлять синтаксические элементы того или иного языка программирования. Таким образом мы будем представлять проект в виде абстрактного синтаксического дерева, и задача сопоставления будет состоять в поиске соответствующих друг другу элементов двух деревьев.

Для формализации понятия соответствия необходимо построить модель, в которой задать функцию схожести, что само по себе является отдельной задачей.

2.2. Задача поиска модели

Задача поиска модели заключается в поиске такой функций схожести элементов этих синтаксических элементов проекта, которая бы выдавала наиболее адекватные с практической точки зрения результаты. Для формализации оценки практической значимости результатов, можно создать тестовую выборку примеров, в которой желаемые результаты сопоставления задать вручную.

3. Случайное марковское поле

Задача сопоставления — одна из нередко встречающихся задач информатики. Например, её решение актуально в задаче стереозрения, когда для пары изображений надо сопоставить соответствующие друг другу пиксели. Для решения этой задачи предложены различные подходы и модели. Одна из наиболее эффективных на сегодняшний день моделей — модель случайного марковского поля (MRF) [4, 5].

Случайное марковское поле — графическая модель, в которой случайные величины изображаются вершинами графа, а зависимости — рёбрами.

Вероятность принятия его случайной величиной того или иного значения при условии известных значений других случайных величин зависит только от соседних случайных величин в этом графе. Таким образом рёбра графа отражают зависимости между случайными переменными.

Применительно к задаче сопоставления двух деревьев случайными величинами являются вершины одного дерева, а множеством значений этих величин — множество вершин другого дерева. И задача заключается в поиске наиболее вероятной комбинации значений случайных величин (MAP-MRF подход), а значит, наиболее вероятного результата сопоставления. Один из наиболее известных алгоритмов оптимизации в случайном марковском поле — алгоритм распространения свидетельства (belief propagation) [4].

Случайное марковское поле — это параметрическая модель, которая задаётся не только отношением соседства (связи) случайных величин, но и так называемыми потенциальным функциями, определяющими «силу» этих

связей. Для построения модели случайного марковского поля необходима подстройка потенциальных функций по обучающей выборке. Обычно такие задачи решаются методом оптимизации.

Специфика этой задачи заключается в том, что для одной оценки качества модели при данном наборе параметров (т. е. для измерения функционала качества от параметров) необходимо полностью произвести достаточно трудоёмкую операцию оптимизации в марковском поле при этих параметрах.

Таким образом, среди методов оптимизации лучше всего подходят те, которые требуют минимального количества измерения функционала качества.

Одним из таких методов является метод стохастической оптимизации со случайным пробным возмущением (SPSA) [6]. Его особенность заключается в том, что для шага оптимизации необходимо только два измерения, что выгодно отличает его от других алгоритмов оптимизации, таких, как, например, метод градиентного спуска.

4. Заключение

Задачи автоматизированного слияния и автоматизированного поиска соответствий являются актуальными на сегодняшний момент. Существующие системы контроля версий применяют методы, основанные на сопоставлениях строк, что работает для любых видов текстовых файлов. Однако в случае структурированного текста возможно улучшение качества сопоставления, применив предложенный в докладе подход.

Литература

1. http://subversion.apache.org/

2. http://git-scm.com/

3. http://mercurial.selenic.com/

4. *Yedidia J. S., Freeman W. T., Weiss Y.* Understanding belief propagation and its generalizations // Exploring artificial intelligence in the new millennium. 2003. P. 239–269.

5. *Tipwai P., Madarasmi S.* A coarse-and-fine Bayesian belief propagation for correspondence problems in computer vision // Proceedings of the artificial intelligence 6th Mexican international conference on Advances in artificial intelligence. 2007. P. 683–693.

6. *Граничин О. Н.* Об одной стохастической рекуррентной процедуре при зависимых помехах в наблюдении, использующей на входе пробные возмущения // Вестник Ленингр. ун-та. Сер. 1. 1989. Вып. 1 (4). С. 19–21.

СЕТЕВОЕ АДАПТИВНОЕ УПРАВЛЕНИЕ ГРУППОЙ ЛЕГКИХ БЕСПИЛОТНЫХ ЛЕТАТЕЛЬНЫХ АППАРАТОВ НА ОСНОВЕ МУЛЬТИАГЕНТНОГО ПОДХОДА

К. С. Амелин (E-mail: konstantinamelin@mail.ru)

Санкт-Петербургский Государственный Университет, аспирант

В статье рассматривается проблема адаптивного управления группой беспилотных летательных аппаратов (БПЛА) как с позиций аппаратной реализация, так и выбора алгоритма. Принцип применения сети БПЛА для группового выполнения задач, в которых обмен информацией осуществляется только между близкими БПЛА и возможно с большими затратами с базовой станцией, характеризуется отсутствием автономной жесткой постановки задач, что позволяет группе более эффективно решать задачи мониторинга местности и оптимизации планов полёта. Для решения предлагается использовать мультиагентный подход для создание сетевого управления группой БПЛА. Также в статье рассмотрены возможности использования восходящих воздушных потоков для увеличения продолжительности полётов. Для определение центра восходящего потока в условиях существенных погрешностей измерений применяется рандомизированный алгоритм стохастической оптимизации.

Использование сети «интеллектуальных» БПЛА для группового выполнения задач характеризуются отсутствием автономной «жесткой» постановки задачи, позволяя группе оперативно принимать решения по изменению сценария выполнения поставленной задачи. Для решения подобных задач применяются мультиагентные технологии.

В [1] описан пример мультиагентного взаимодействия главного компьютера (на управляемом человеком самолете) и группы БПЛА, которые выполняют задачи поставленные первым. Основа использования мультиагентного подход в нашем проекте — «общение» моделей друг с другом, при этом образуется новый источник информации для корректировки полета автопилотом кажого БПЛА из группы.

Для возможности реализации применения мультиагентного подхода к группе БПЛА в двухуровневую схему управления комплексом БПЛА мы добавляем промежуточный средний уровень, который реализуется за счёт дополнительного микрокомпьютера установленного в БПЛА [2].

Таким образом, мы получаем новую трёхуровневую систему управления БПЛА-агента (рис. 1).

Рассмотрим два основных типа алгоритмов управления полётом БПЛА, мониторинг местности (на примере исследования экологической ситуации) и оптимизации полёта БПЛА (на примере использования термических потоков для увеличения дальности полёта [3]).

При решении задачи мониторинга экологической обстановки в акватории залива (например, поиск разливов нефти) работа системы организуется следующим образом:

- Выбирается тип задачи.
- Формируются отдельные задачи для каждого члена.
- В микрокомпьютер каждого БПЛА группы записывается глобальная задача и отдельная задача этого самолета-агента.
- Каждый агент приступает к выполнению поставленной ему задачи.
- Когда в зону Wi-Fi видимости одного БПЛА из группы попадает другой, при «общении» происходит передача между агентами накопленной информации и при необходимости взаимное уточнение отдельных задач.
- Базовые наземные станции, обеспечивая связь с центром обработки данных (ЦОД), принимают/передают информацию от БПЛА, находящихся в их зоне видимости или поддерживающих связь через Интернет.
- Полученная в ЦОДе информация обрабатывается и визуализируется для заказчика.
- Наличие обратной связи с мобильными агентами (БПЛА) позволяет оперативно формировать из ЦОДа инструкции по корректировке их заданий.

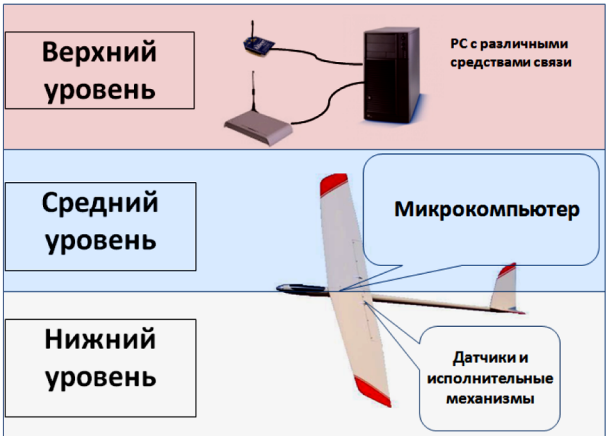


Рис. 1. Три уровня управления БПЛА-агента

Не менее развитой тематикой при разработке программ управления БПЛА являются алгоритмы оптимизации полёта. Одним из способов накопления энергии и увеличения дальности полёта является использование восходящих воздушных потоков (термических потоков или термиков), образующихся в нижних слоях атмосферы за счёт срыва теплого воздуха с поверхности земли при её нагреве под воздействием солнечных лучей. Подобные исследования были проведены в работе Эллана [4], где было показано решение задачи длительного патрулирования местности одним БПЛА с ис-

пользованием энергии термиков. При возникновении положительной вертикальной скорости БПЛА использует стратегию, разработанную планеристом Х. Райнером [5]. Она в заключается в ледующем:

- При увеличении вертикальной скорости, уменьшить радиус кривизны траектории БПЛА.
- При уменьшении вертикальной скорости, увеличить радиус кривизны траектории БПЛА.
- При постоянной вертикальной скорости, держать постоянный радиус кривизны траектории БПЛА.

Для групповой работы дополнительной задачей после вхождения БПЛА в термический поток остается точное определение координат центра потока для передачи их другим БПЛА. В [6] показано насколько неточный и трудоёмкий способ определения координат центра термика, описанный в [4].

Нами предлагается использование модификации метода типа SPSA для определения центра термического потока. Детали SPSA метода подробно описаны в [7–10]. Цель центрирования найти координаты центра термического потока.

Общая схема задачи и метода SPSA:

Мы выбираем точки, в которых проводим измерения $x_1, x_2 \ldots \in \mathbb{R}^d$, и наблюдения в этих точках $y_1, y_2 \ldots \in \mathbb{R}^1$ удовлетворяют соотношениям

$$y_n = F(x_n, \omega_n) + \vartheta_n,$$

где $\omega_1, \omega_2, \ldots$ — неконтролируемая неизвестная случайная последовательность с неизвестным распределением в $P(\cdot)$, а $\vartheta_1, \vartheta_2, \ldots$ — неизвестная, но ограниченная (неслучайная).

Цель — максимизировать SPSA — метод:

$$f(x) = \int F(x, \omega)\, P(d\omega).$$

$\Delta_1, \Delta_2, \ldots$ — выборка с распределением Бернулли , где

$$\Delta_n = \begin{pmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{pmatrix} \in \mathbb{R}^d.$$

$$\hat{\theta}_0 \in \mathbb{R}^d, \qquad x_n^{\pm} = \hat{\theta}_{n-1} \pm \beta_n^{\pm} \Delta_n,$$

$$\hat{\theta}_n = \hat{\theta}_{n-1} + \frac{\alpha_n}{\beta_n^{+} + \beta_n^{-}} \Delta_n \left(y_n^{+} - y_n^{-} \right)$$

или с одним измерением:

$$x_n = \hat{\theta}_{n-1} + \beta_n \Delta_n, \quad \hat{\theta}_n = \hat{\theta}_{n-1} + \frac{\alpha_n}{\beta_n} \Delta_n y_n.$$

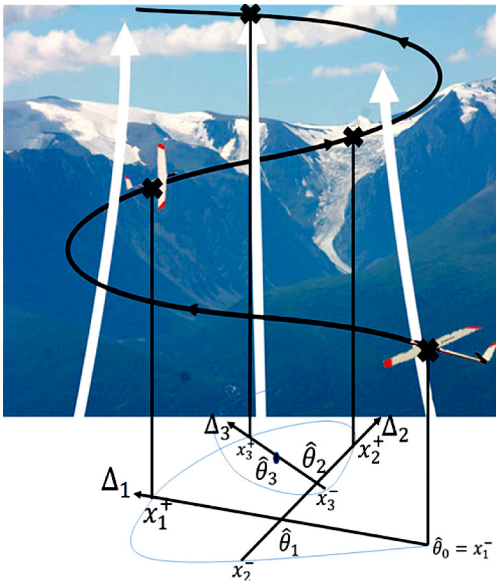


Рис. 2. Иллюстрация метода SPSA

В [11] приводятся преимущества использования системы термических потоков группой взаимодействующих друг с другом БПЛА по сравнению с одиночным планером.

В дальнейшем планируется исследования и создание алгоритмов избежания столкновения БПЛА-агентов при сближении. Также планируется исследование протоколов передачи данных и конвертирования изображения для ускорения передачи данных. Планируется применение алгоритмов в реальной группе БПЛА.

Литература

1. *Baxter J. W., Horn G. S., Leivers D. P.* Fly-by-Agent: Controlling a Pool of UAVs via a Multi-Agent System. QinetiQ Ltd Malvern Technology Centre St Andrews Road. Malvern. UK. 2007.

2. *Амелин К. С., Антал Е. И., Васильев В. И., Граничина Н. О.* Адаптивное управление автономной группой беспилотных летательных аппаратов // Стохастическая оптимизация в информатике. Вып. 5. 2009. С. 157–166.

3. Официальный сайт по беспилотным летательным аппаратам. http://bp-la.ru/

4. *Allen M. J.* Autonomous soaring for improved endurance of a small uninhabited air vehicle // AIAA 2005–1025, 43rd AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, 10–13 January, 2005.

5. *Reichmann H.* Cross-Country Soaring. — Minnesota: Soaring Society of America, Inc. 1978.

6. *Daniel J. Edwards*. Implementation Details and Flight Test Results of an Autonomous Soaring Controller // North Carolina State University.

7. *Spall J. C.* Multivariate stochastic approximation using a simultaneous perturbation gradient approximation // IEEE Trans. Automat. Contr. Vol. 37. 1992. P. 332–341.

8. *Granichin O. N.* A stochastic recursive procedure with dependent noises in the observation that uses sample perturbations in the input // Vestnik Leningrad Univ. Math. 1989. Vol. 22. No. 1(4). P. 27–31.

9. *Granichin O. N.* Procedure of stochastic approximation with disturbances at the input // Automation and Remote Control. 1992. Vol. 53 No. 2, part 1. P. 232–237.

10. *Granichin O. N., Polyak B. T.* Randomized Algorithms of an Estimation and Optimization Under Almost Arbitrary Noises. —Moscow: Nauka. 2003.

11. *Antal C., Granichin O., Levi S.* Adaptive Autonomous Soaring of Multiple UAVs Using Simultaneous Perturbation Stochastic Approximation // 49th IEEE Conference on Decision and Control, Hilton Atlanta Hotel, Atlanta, GA, USA, December 15–17, 2010.

ИССЛЕДОВАНИЕ ПРОГРАММНЫХ МОДЕЛЕЙ И ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ДЛЯ ПЕРСПЕКТИВНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Е. С. Логунова, Н. В. Фролова, В. С. Холушкин

Саровский физико-технический институт — филиал Национального исследовательского ядерного университета «МИФИ», г. Саров

В работе представлены результаты аналитических исследований, выполненных с целью определения и анализа наиболее перспективных программных моделей и языков программирования для разработки новых и переноса существующих научных приложений, предназначенных для будущих надпетафлопных вычислительных систем.

Параллельные высокопроизводительные вычисления в настоящее время перешли из узкоспециализированных технологий в повседневную жизнь, являясь перспективной и максимально востребованной областью IT.

Подготовка специалистов в сфере параллельных вычислений началась в СарФТИ НИЯУ МИФИ еще в 1990-х годах, благодаря уникальному кадровому потенциалу ИТМФ ВНИИЭФ и востребованности специалистов данного профиля.

Программы подготовки ориентированы на решение задач математического моделирования сложных физических проблем. В учебном процессе рассматриваются авторские методики решения задач, классические схемы, гетерогенные вычислительные среды. Подготовка специалистов строится на использовании в учебном процессе неоднородных вычислительных кластерных систем с применением специализированного программного обеспечения.

В данной работе сделана попытка анализа состояния сферы программных технологий, как существующих, так и находящихся в разработке, способных стать востребованными в эпоху мульти-петафлопной производительности, как собственно и ускорить ее наступление. Современные тенденции в развитии рынка вычислительных систем для высокопроизводительных вычислений характеризуются появлением большого числа новых архитектурных и аппаратных решений, некоторые из них фундаментальны и способны потенциально обеспечить желаемый прирост производительности. Две самые значительные идеи, определяющие развитие вычислительных средств в последние годы — это иерархический параллелизм и специализация. Наиболее значительным результатом применения этих идей и их комбинаций явилось создание многоядерных процессоров, как однородных, так и разноядерных, а также появление и использование различного рода аппаратных ускорителей.

Очевидно, что без появления новых концепций параллельного программирования и разработки программного обеспечения, воплощающего их в форме удобных и простых в применении программных продуктов, использование аппаратного потенциала новых поколений вычислительных систем не будет возможно, а их пиковая производительность будет подтверждаться лишь вручную написанными на языках низкого уровня бенчмарками. Помимо способствования извлечению максимальной производительности важными задачами являются также повышение производительности программирования, обеспечение переносимости приложений (в том числе и уже существующих), их масштабируемости и надежности.

В работе помимо собственно описания и анализа существующих и разрабатываемых моделей, языков и программных систем [1–5] предпринята попытка их определенной классификации. Рассмотренные модели и языки классифицированы по следующим категориям:

- действующие стандарты параллельного программирования;
- языки программирования модели PGAS;
- модели и языки следующего поколения;
- программирование аппаратных ускорителей.

В работе приведено описание результатов практического применения языка параллельного программирования UPC в ряде проектов лаборатории БиПро (совместная учебно-исследовательская лаборатория СарФТИ и корпорации Интел), потенциально способного найти широкое применение в программировании будущих высокопроизводительных систем.

Литература

1. The Message Passing Interface standard. http://www.mcs.anl.gov/research/projects/mpi/

2. *D. Bonachea.* GASNet Specification, U.C. Berkeley Tech Report CSD-02-1207 (2002).

3. Chapel Language Specification vO.78. http://chapel.cray.com/spec-0.750.pdf

4. *J. Dongarra* et. al., DARP A's HPCS Program: History, Models, Tools, Languages, Advances in Computers (2008).

5. GCC UPC, http://www.intrepid.com/upc.html

Локальные алгоритмы и распараллеливание

Демьянович
Юрий Казимирович

д.ф.-м.н., профессор
заведующий кафедрой параллельных алгоритмов СПбГУ

РАЗРАБОТКА НАБОРА ПОДКЛЮЧАЕМЫХ МОДУЛЕЙ К GSTREAMER ДЛЯ ВЕЙВЛЕТНОГО ПРЕОБРАЗОВАНИЯ С ЦЕЛЬЮ СЖАТИЯ И ВОССТАНОВЛЕНИЯ ЗВУКОВЫХ ФАЙЛОВ

П. Каколин

Санкт-Петербургский государственный университет,
математико-механический факультет,
кафедра параллельных алгоритмов

Современные потоки информации в процессе обработки, хранения и передачи имеют электронную форму: чаще всего это последовательности битов огромной длины, которые можно быстро обрабатывать лишь при наличии больших вычислительных ресурсов.

Сжатие звуковых данных стало одной из важнейших технологий эпохи мультимедиа. Произошедшие за последние десятилетия изменения в сфере телекоммуникационных сервисов также сыграли не последнюю роль в развитии методов передачи речи и прочих звуковых сигналов. Во многих сферах технической деятельности, таких как разработка мультимедийных рабочих станций, а также передача и хранение высококачественных аудиоданных, более всего важна возможность прозрачного кодирования мультимедиа при можно меньшем использовании пропускной способности каналов передачи данных и объемов накопителей.

Примененные различных решений задачи сжатия аудиоданных в реальном времени на персональных компьютерах становится все более распространенным с увеличением производительности их аппаратных составляющих одновременно с уменьшением стоимости, что во многом определяет множество используемых в таких решениях алгоритмов и требования к качеству их результата. В наши дни даже сравнительно дешевый и малопроизводительный персональный компьютер способен обрабатывать аудио CD-качества в реальном времени, расширяя область беспроблемного применения такой обработки не только на задачи малых звукозаписывающих студий, но и на задачи среднестатистического пользователя.

Обработка аудио-сигналов в реальном времени допускает воспроизведение модифицированного потока непосредственно в момент обработки, и, хотя и требует для этого значительных вычислительных мощностей, существенно совершенствует процессы обработки звука: наложение на поток заранее запрограммированного эффекта или изменение параметров и настроек фильтра может быть услышано с пренебрежимо малой временной задержкой, а потому крайне удобно для подбора комбинации оптимальных установок в короткий промежуток времени.

Еще одна возможность использования обработки аудио-сигналов в реальном времени заключается в необходимости адаптировать плотность потока оцифрованных данных к загруженности каналов их передачи. Наконец, такая обработка находит применение в записи воспроизводимого звука на физические носители.

В работе дается краткий обзор существующих методов кодирования аудиоданных. При более подробном их рассмотрении оказывается, что одни требуют значительных вычислительных ресурсов, другие не могут быть применены к аудиопотоку в реальном времени, а третьи (например вариации анализа Фурье) осуществляют настолько избыточное кодирование, что использование их для решения задачи сжатия данных совершенно неприемлемо, и, более того, связанно с некоторыми неудобствами, среди которых невозможность одновременной локализации по отношению к временной и частотной переменным.

Вейвлет-анализ представляется в меру простым и достаточно хорошо подходящим инструментом для сжатия аудиоданных. В работе описан метод анализа функции, заданной на равномерной сетке и принимающей вещественные значения; анализ проводится с помощью *B*-сплайнов. Произведен выбор такого *B*-сплайна, значения которого в узлах сетки получаются целыми. Кроме того, значения разных *B*-сплайнов получаются друг из друга сдвигом. Обнаружено, что укрупнение сетки возможно осуществлять, пересчитывая вейвлетные коэффициенты в непересекающихся окрестностях каждого из удаляемых узлов, что позволило удалять узлы независимо друг от друга.

Алгоритм был реализован на языке C в цепочке элементов GStreamer и апробирован на звуковых файлах высокого качества. Обнаружилась возможность двадцатипроцентного сжатия данных без ощутимой разницы в восприятии оригинального и обработанного звука. Кроме того, была произведена обработка и запись на физический носитель речи, регистрируемой устройством ввода. Используемый алгоритм позволил осуществить такую запись с незначительной задержкой, обусловленной скорее не характеристиками алгоритма, а особенностями его реализации и архитектурой GStreamer.

Будучи, безусловно, несравнимым по степени сжатия например с технологией MP3, рассмотренный метод вейвлетного анализа хорошо проявил себя при работе с аудиоданными среднего качества, продемонстрировав существенные преимущества, среди которых относительная простота реализации и широкие возможности управления степенью сжатия.

КОМПЛЕКС ПРОГРАММ СЖАТИЯ/ВОССТАНОВЛЕНИЯ СИГНАЛОВ НА ОСНОВЕ СПЛАЙН-ВЭЙВЛЕТОВ

В. А. Ракчаев

Современные потоки информации в процессе обработки, хранения и передачи имеют электронную форму: чаще всего это последовательности нулей и единиц огромной длины (10^{12}–10^{16} символов). Такие последовательности можно быстро обрабатывать лишь в случае, когда имеются большие компьютерные ресурсы (быстродействие, память, мощные каналы связи). Задача сокращения объемов цифровой информации за счет отбрасывания несущественных ее составляющих весьма актуальна, причем степень важности эффективного решения этой задачи постоянно возрастает.

На первом месте среди средств решения этой задачи несомненно находятся вэйвлеты. Основной результат теории вэйвлетов — эффективные алгоритмы обработки больших потоков информации. Под эффективностью в данном случае понимают экономное (с точки зрения экономии ресурсов компьютера: памяти и времени обработки) разложение потока информации на составляющие, так чтобы можно было выделить основной информационный поток и вэйвлетный информационный поток. Как правило, основной информационный поток значительно менее плотный, чем исходный поток информации, поэтому его можно передать быстро. Вэйвлетный информационный поток можно передавать фрагментарно либо вообще отбросить.

Цель данной работы заключается в создании пакета программ сжатия и восстановления числовой информации, основанного на сплайн-вэйвлетных разложениях эрмитова типа и в анализе возможностей распараллеливания указанного алгоритма.

Краткое описание алгоритма: входной поток чисел разбивается по формулам декомпозиции на основной и вэйвлетный (каждый процесс на этапе декомпозиции из исходного потока выделяет свою вэйвлетную составляющую, которые образуют весь вэйвлетный поток, а основной поток получается из входного некоторым преобразованием). Затем из основного и вэйвлетного потоков по формулам реконструкции восстанавливают исходный поток. Сравнивая исходный и восстановленный потоки, можно убедиться в успешной работе программы и корректности результатов разбиения, а также определить время выполнения программы при разном количестве процессоров.

Программа корректно работает для любого допустимого входного потока.

Программа была протестирована на 10 выполняющих процессорах кластера. Результатом выполнения программы стала успешная реализация вышеописанного алгоритма, подтверждением чего является полученный результирующий разностный поток, состоящий из нулей.

В данной работе программно реализован параллельный алгоритм сжатия/восстановления числовой информации, основанного на сплайн-вэйвлетных

разложениях эрмитова типа. Он может применяться для многих задач сжатия, решаемых на практике. С помощью реализованного алгоритма можно обрабатывать потоки цифровой информации (видео- и аудиопотоки), отбрасывая несущественные части (частично или полностью игнорируя вэйвлетный поток) для ускорения передачи информации по каналам связи. Можно проводить дальнейшее разложение полученных потоков, используя рассматриваемый алгоритм для нескольких уровней точности (например, для последовательного его уточнения основного контура изображения).

Литература

1. *Демьянович Ю. К., Ходаковский В. А.* Введение в теорию вэйвлетов. (Учеб. пособие.) СПб.: Санкт-Петербургский государственный университет путей сообщения», 2008. 51 с.

2. *Демьянович Ю. К., Зимин А. В.* О всплесковом разложении сплайнов эрмитова типа. Проблемы математического анализа. 2007. Т. 35. С. 33–45.

ОБРАБОТКА ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СПЛАЙН-ВЭЙВЛЕТНЫХ РАЗЛОЖЕНИЙ

Д. М. Трескунов

Изображение в самом простейшем виде может быть представлено потоком чисел (например, числовым потоком интенсивности цветов для монохромного изображения). Такие потоки обычно имеют огромную длину, и потому задача их сжатия актуальна. Последнее время для этой цели широко применяются сплайн-вэйвлетные разложения, отличительными особенностями которых являются вычислительная простота и хорошие результаты сжатия.

Вэйвлетное разложение подразумевает представление потока в виде двух составляющих: основной и вэйвлетной (см. [1–4]). Основная составляющая может рассматриваться как сжатие исходного потока, а вэйвлетная как поправка к основной, с помощью которой можно восстановить исходный поток с точностью до ошибок округления.

Современная компьютерная графика делится на два класса: растровая и векторная; простейшее растровое изображение может быть представлено в виде матрицы пикселей. Для примера рассмотрим несжатое изображение в формате BMP с цветовой моделью RGB — оно представляет собой ряд специальных заголовков, содержащих дополнительную информацию об изображении, и само изображение в виде массива пикселей, записанных построчно снизу вверх. Каждый пиксель занимает три байта и представляет собой цвет в модели RGB, т. е. три целых числа от 0 до 255 включительно. Таким образом изображение может быть представлено в виде трех потоков целых чисел из отрезка [0, ..., 255] одинаковой длины. Более качественное вэйвлетное разложение получается при плавном изменении этих чисел; поэтому следует получать поток из изображения не построчно, а «змейкой», т. к. при этом переходы цветов оказываются более плавными.

Каждый из упомянутых трех потоков можно рассматривать как функцию $u(t)$, заданную на целочисленных точках отрезка $[0, ..., N-1]$, где N — число всех пикселей изображения. Функцию $u(t)$ называем исходной функцией, указанное множество целых точек — исходной сеткой, а целые точки — узлами исходной сетки. Сжатие изображения получается за счет выбрасывания некоторого набора узлов из этой сетки, и дальнейшего проецирования исходной функции с помощью получившейся укрупненной сетки.

Таким образом, возникает две задачи: определение номеров выбрасываемых узлов и соответствующее разложение исходной функции. Первая задача решается с помощью формул (см. [2])

$$\bar{S}^* = (f', i, q, P, h, g, \bar{C}) \stackrel{def}{=} \left[\frac{\bar{C}}{P\left(\frac{1}{q} \sum_{i'=i}^{i+q-1} \left| f'(i'h) + \frac{\bar{C}}{g} \right| \right)} \right],$$

где $S-1$ — количество следующих выбрасываемых узлов; i — номер рассматриваемого узла x_i сетки $\{x_i'\}_{i=0,\,1,\,\ldots,\,N}$; P — относительное количество сохраняемых узлов; q — параметр усреднения; g — параметр, характеризующий максимальное количество единовременно выбрасываемых узлов; $\bar{C} = \dfrac{1}{N}\sum\limits_{i'=0}^{N-1} |f'(x_{i'})|$; константа $\bar{C}$ может вычисляться приближенно.

Параметр P характеризует степень сжатия изображения; он выбирается из интервала (0, 1).

Опишем алгоритм разложения исходного потока на основную и вэйвлетную составляющие. Рассмотрим полиномиальный B-сплайн ω_j^B второй степени на сетке X; его можно задать следующим образом:

$$\omega_j^B(t) = (t-x_j)^2 (x_{j+1}-x_j)^{-1}(x_{j+2}-x_j)^{-1} \quad \text{при} \quad t \in [x_j,\, x_{j+1}),$$

$$\omega_j^B(t) = (x_{j+2}-x_j)^{-1}(x_{j+2}-x_{j+1})^{-1}(x_{j+3}-x_{j+1})^{-1} \times$$
$$\times[(x_j - x_{j+2} - x_{j+3} + x_{j+1})t^2 - 2(x_{j+1}x_j - x_{j+2}x_{j+3})t +$$
$$+ x_j x_{j+1} x_{j+3} - x_j x_{j+2} x_{j+3} + x_j x_{j+1} x_{j+2} - x_{j+1} x_{j+2} x_{j+3}] \quad \text{при} \quad t \in [x_{j+1},\, x_{j+2}),$$

$$\omega_j^B(t) = (t-x_{j+3})^2 (x_{j+3}-x_{j+2})^{-1}(x_{j+3}-x_{j+1})^{-1} \quad \text{при} \quad t \in [x_{j+2},\, x_{j+3}],$$

$$\omega_j^B(t) = 0 \ \text{ — в остальных случаях.}$$

Дальше понадобятся линейные функционалы g^i, определяемые на непрерывно дифференцируемых функциях по формуле

$$\langle g^i,\, u\rangle \overset{def}{=} u(x_{i+1}) + (x_{i+2} - x_{i+1})\frac{u'(x_{i+1})}{2}.$$

В качестве функции $u(t)$ рассмотрим функцию

$$u(t) = \sum_{x_j \in X} c_j \omega_j(t),$$

где c_j — элементы исходного потока, а $X = \{x_j\}$ — исходная сетка.

Проекция функции $u(t)$ с помощью укрупненной сетки $\tilde{X} = \{\tilde{x}_j\}$ имеет вид

$$\tilde{u}(t) = \sum_{\tilde{x}_j \in \tilde{X}} \langle \tilde{g}^i,\, u\rangle\, \tilde{\omega}_i(t).$$

Используя определение функционалов g^i, получаем

$$\tilde{u}(t) = \sum_{\tilde{x}_j \in \tilde{X}} a_i \tilde{\omega}_i(t) = \sum_{\tilde{x}_j \in \tilde{X}} \tilde{\omega}_i(t) \sum_{x_j \in X} c_j \left(\omega_j(\tilde{x}_{i+1}) + (\tilde{x}_{i+2} - \tilde{x}_{i+1})\frac{\omega_j'(x_{i+1})}{2} \right).$$

Из этой формулы находятся числа a_i, которые представляют собой основной поток. Вэйвлетный поток рассчитывается по следующей формуле:

$$b_s = \langle g_s, u - \tilde{u} \rangle = c_s - \langle g_s, \tilde{u} \rangle = c_s - \sum_{\tilde{x}_j \in \tilde{X}} a_i \left(\tilde{\omega}_i(x_{s+1}) + (x_{s+2} - x_{s+1}) \frac{\tilde{\omega}_i'(x_{s+1})}{2} \right).$$

Эта формула используется и для восстановления исходного потока. Элементы вэйвлетного потока часто близки к нулю, что позволяет экономить вычислительные ресурсы. С помощью элементов основного потока и формулы $\tilde{u}(t) = \sum_{i \in \tilde{x}_j \in \tilde{X}} a_i \tilde{\omega}_i(t)$ получаются приближенные значения исходного потока в любой точке исходной сетки, что часто ведет к достаточно хорошей аппроксимации исходного изображения.

Литература

1. *Демьянович Ю. К.* Вэйвлеты & минимальные сплайны. 2003.

2. *Демьянович Ю. К., Ходаковский В. А.* Введение в вэйвлеты. 2007.

3. *Демьянович Ю. К., Иванцова О. Н.* Новые представления сплайн-вэйвлетных разложений // Сб. «Проблемы математического анализа». Вып. 46 (2010). С. 73–104.

4. http://www.jpeg.org/jpeg2000/index.html

МИНИМАЛЬНЫЕ СПЛАЙНЫ ТРЕТЬЕГО ПОРЯДКА И БИОРТОГОНАЛЬНЫЕ СИСТЕМЫ

Ю. К. Демьянович, В. О. Дронь

1. Введение

Минимальные сплайны третьего порядка ранее исследовались для бесконечной сетки на открытом интервале (см. [2]). Поскольку на практике обрабатываемые числовые потоки конечны, важно рассмотреть случай конечной сетки на отрезке вещественной оси. Для минимальных сплайнов второго порядка такое исследование проведено в работе [3].

Цель данной работы — провести подобное исследование для минимальных сплайнов третьего порядка. Здесь получены представления пространств минимальных сплайнов на замкнутом интервале в условиях, когда сплайновая сетка конечна.

2. О представлениях (X, A, φ)-сплайнов

2.1. На интервале (α, β) вещественной оси $\mathbb{R}^1$ рассмотрим сетку

$$X: \qquad \ldots x_{-2} < x_{-1} < x_0 < x_1 < x_2 < \ldots,$$

где $\alpha = \lim_{j\to-\infty} x_j$, $\beta = \lim_{j\to+\infty} x_j$ (случаи $\alpha = -\infty$, $\beta = +\infty$ не исключаются).

Каждому $j \in \mathbb{Z}$ поставим в соответствие вектор-столбец $\mathbf{a}_j \in \mathbb{R}^4$ и введем квадратную матрицу четвертого порядка $A_j \stackrel{\text{def}}{=} (\mathbf{a}_{j-3}, \mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \mathbf{a}_j)$.

Определение 1. Бесконечная цепочка векторов $\mathbf{A} \stackrel{\text{def}}{=} \{\mathbf{a}_j\}_{j\in\mathbb{Z}}$ называется полной, если $\det A_j \neq 0 \quad \forall j \in \mathbb{Z}$.

Множество всех бесконечных полных цепочек обозначим $\mathbb{A}_\infty$.

Пусть цепочка $\mathbf{A}$ лежит в $\mathbb{A}_\infty$, а $\varphi(t)$ — четырехкомпонентная вектор-функция, которая задана на интервале (α, β), и компоненты которой линейно независимы на любом подинтервале (c, d) интервала (α, β).

Координатными (X, $\mathbf{A}$, φ)-сплайнами третьего порядка называются функции $\omega_j(t)$, удовлетворяющие аппроксимационным соотношениям

$$\sum_j \mathbf{a}_j \omega_j(t) = \varphi(t) \qquad \forall t \in (\alpha, \beta), \qquad \operatorname{supp} \omega_j \subset [x_j, x_{j+4}].$$

Этим соотношениям можно придать вид

$$\mathbf{a}_{k-3}\omega_{k-3}(t) + \mathbf{a}_{k-2}\omega_{k-2}(t) + \mathbf{a}_{k-1}\omega_{k-1}(t) + \mathbf{a}_k\omega_k(t) = \varphi(t) \qquad \forall t \in (x_k, x_{k+1}), \quad (2.1)$$

где

$$\operatorname{supp} \omega_j \subset [x_j, x_{j+4}]. \quad (2.2)$$

Теорема 1. Если $\mathbf{A} \in \mathbb{A}_\infty$, то из соотношений (2.1)–(2.2), рассматриваемых при всех $k \in \mathbb{Z}$, однозначно определяются сплайны $\omega_j(t)$:

$$\omega_j(t) = \frac{\det(\mathbf{a}_{j-3}, \mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \varphi(t))}{\det(\mathbf{a}_{j-2}, \mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \mathbf{a}_j)} \qquad \text{при } t \in (x_j, x_{j+1}),$$

$$\omega_j(t) = \frac{\det(\mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \varphi(t), \mathbf{a}_{j+1})}{\det(\mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \mathbf{a}_j, \mathbf{a}_{j+1})} \qquad \text{при } t \in (x_{j+1}, x_{j+2}),$$

$$\omega_j(t) = \frac{\det(\mathbf{a}_{j-1}, \varphi(t), \mathbf{a}_{j+1}, \mathbf{a}_{j+2})}{\det(\mathbf{a}_{j-1}, \mathbf{a}_j, \mathbf{a}_{j+1}, \mathbf{a}_{j+2})} \qquad \text{при } t \in (x_{j+2}, x_{j+3}),$$

$$\omega_j(t) = \frac{\det(\varphi(t), \mathbf{a}_{j+1}, \mathbf{a}_{j+2}, \mathbf{a}_{j+3})}{\det(\mathbf{a}_j, \mathbf{a}_{j+1}, \mathbf{a}_{j+2}, \mathbf{a}_{j+3})} \qquad \text{при } t \in (x_{j+3}, x_{j+4}),$$

$$\omega_j(t) = 0 \qquad \text{при } t \notin (x_j, x_{j+4}),$$

где $j \in \mathbb{Z}$ и $t \in (\alpha, \beta) \backslash X$.

Пусть $\mathbf{A} \in \mathbb{A}_\infty$. Линейная оболочка $\mathbb{S}(X, \mathbf{A}, \varphi)$ функций $\{\omega_j\}_{j \in \mathbb{Z}}$ называется пространством сплайнов третьего порядка на сетке X:

$$\mathbb{S}(X, \mathbf{A}, \varphi) \stackrel{\text{def}}{=} \left\{ u \mid u = \sum_j c_j \omega_j \quad \forall c_j \in \mathbb{R}^1 \right\}.$$

Нетрудно видеть, что для вектора

$$\mathbf{d}_k \stackrel{\text{def}}{=} \left(\det \begin{pmatrix} [\mathbf{a}_{k-3}]_1 & [\mathbf{a}_{k-2}]_1 & [\mathbf{a}_{k-1}]_1 \\ [\mathbf{a}_{k-3}]_2 & [\mathbf{a}_{k-2}]_2 & [\mathbf{a}_{k-1}]_2 \\ [\mathbf{a}_{k-3}]_3 & [\mathbf{a}_{k-2}]_3 & [\mathbf{a}_{k-1}]_3 \end{pmatrix}, \; -\det \begin{pmatrix} [\mathbf{a}_{k-3}]_0 & [\mathbf{a}_{k-2}]_0 & [\mathbf{a}_{k-1}]_0 \\ [\mathbf{a}_{k-3}]_2 & [\mathbf{a}_{k-2}]_2 & [\mathbf{a}_{k-1}]_2 \\ [\mathbf{a}_{k-3}]_3 & [\mathbf{a}_{k-2}]_3 & [\mathbf{a}_{k-1}]_3 \end{pmatrix}, \right.$$

$$\left. \det \begin{pmatrix} [\mathbf{a}_{k-3}]_0 & [\mathbf{a}_{k-2}]_0 & [\mathbf{a}_{k-1}]_0 \\ [\mathbf{a}_{k-3}]_1 & [\mathbf{a}_{k-2}]_1 & [\mathbf{a}_{k-1}]_1 \\ [\mathbf{a}_{k-3}]_3 & [\mathbf{a}_{k-2}]_3 & [\mathbf{a}_{k-1}]_3 \end{pmatrix}, \; -\det \begin{pmatrix} [\mathbf{a}_{k-3}]_0 & [\mathbf{a}_{k-2}]_0 & [\mathbf{a}_{k-1}]_0 \\ [\mathbf{a}_{k-3}]_1 & [\mathbf{a}_{k-2}]_1 & [\mathbf{a}_{k-1}]_1 \\ [\mathbf{a}_{k-3}]_2 & [\mathbf{a}_{k-2}]_2 & [\mathbf{a}_{k-1}]_2 \end{pmatrix} \right)^T$$

справедливы следующие соотношения:

$$\mathbf{d}_j^T \mathbf{a}_{j-4} \neq 0, \quad \mathbf{d}_j^T \mathbf{a}_{j-3} = \mathbf{d}_j^T \mathbf{a}_{j-2} = \mathbf{d}_j^T \mathbf{a}_{j-1} = 0, \quad \mathbf{d}_j^T \mathbf{a}_j \neq 0 \tag{3}$$

(первое и последнее соотношение являются следствием полноты цепочки; оставшиеся можно установить, заметив, что $\mathbf{d}_k^T \mathbf{x} = \det(\mathbf{a}_{k-3}, \mathbf{a}_{k-2}, \mathbf{a}_{k-1}, \mathbf{x})$).

Теорема 2. Для функций $\omega_j, j = k-3, k-2, k-1, k$ на промежутке (x_k, x_{k+1}) справедливы представления

$$\omega_{k-3}(t) = \frac{\mathbf{d}_{k+1}^T \varphi(t)}{\mathbf{d}_{k+1}^T \mathbf{a}_{k-3}},$$

$$\omega_{k-2}(t) = \frac{\mathbf{d}_{k+2}^T \varphi(t)}{\mathbf{d}_{k+2}^T \mathbf{a}_{k-2}} - \frac{\mathbf{d}_{k+2}^T \mathbf{a}_{k-3}}{\mathbf{d}_{k+2}^T \mathbf{a}_{k-2}} \frac{\mathbf{d}_{k+1}^T \varphi(t)}{\mathbf{d}_{k+1}^T \mathbf{a}_{k-3}},$$

$$\omega_{k-1}(t) = \frac{\mathbf{d}_{k-1}^T \varphi(t)}{\mathbf{d}_{k-1}^T \mathbf{a}_{k-1}} - \frac{\mathbf{d}_{k-1}^T \mathbf{a}_k}{\mathbf{d}_{k-1}^T \mathbf{a}_{k-1}} \frac{\mathbf{d}_k^T \varphi(t)}{\mathbf{d}_k^T \mathbf{a}_k},$$

$$\omega_k(t) = \frac{\mathbf{d}_k^T \varphi(t)}{\mathbf{d}_k^T \mathbf{a}_k},$$

причем каждое из них зависит лишь от векторов $\mathbf{a}_{k-3}$, $\mathbf{a}_{k-2}$, $\mathbf{a}_{k-1}$, $\mathbf{a}_k$.

Теорема 3. Для функции ω_j справедливы представления

$$\omega_j(t) = \frac{\mathbf{d}_j^T \varphi(t)}{\mathbf{d}_j^T \mathbf{a}_j} \qquad \text{при } t \in (x_j,\ x_{j+1}),$$

$$\omega_j(t) = \frac{\mathbf{d}_j^T \varphi(t)}{\mathbf{d}_j^T \mathbf{a}_j} - \frac{\mathbf{d}_j^T \mathbf{a}_{j+1}}{\mathbf{d}_j^T \mathbf{a}_j} \frac{\mathbf{d}_{j+1}^T \varphi(t)}{\mathbf{d}_{j+1}^T \mathbf{a}_{j+1}} \qquad \text{при } t \in (x_{j+1},\ x_{j+2}),$$

$$\omega_j(t) = \frac{\mathbf{d}_{j+4}^T \varphi(t)}{\mathbf{d}_{j+4}^T \mathbf{a}_j} - \frac{\mathbf{d}_{j+4}^T \mathbf{a}_{j-1}}{\mathbf{d}_{j+4}^T \mathbf{a}_j} \frac{\mathbf{d}_{j+3}^T \varphi(t)}{\mathbf{d}_{j+3}^T \mathbf{a}_{j-1}} \qquad \text{при } t \in (x_{j+2},\ x_{j+3}),$$

$$\omega_j(t) = \frac{\mathbf{d}_{j+4}^T \varphi(t)}{\mathbf{d}_{j+4}^T \mathbf{a}_j} \qquad \text{при } t \in (x_{j+3},\ x_{j+4}),$$

Координатная функция $\omega_j(t)$ определяется узлами $x_{j+1}, x_{j+2}, x_{j+3}, x_{j+4}$ и векторами $\mathbf{a}_{j-3}$, $\mathbf{a}_{j-2}$, $\mathbf{a}_{j-1}$, $\mathbf{a}_j$, $\mathbf{a}_{j+1}$, $\mathbf{a}_{j+2}$, $\mathbf{a}_{j+3}$.

Замечание 1. Если $\varphi(t) = (1,\ t,\ t^2,\ t^3)^T$, то ω_j — полиномиальный сплайн третьей степени с носителем $\operatorname{supp} \omega_j = [x_j,\ x_{j+4}]$.

2.2. Из бесконечной сетки X выделим конечную сетку X_N,

$$X_N : x_0 < x_1 < \ldots < x_{N-1} < x_N,$$

а из полной бесконечной цепочки $\mathbf{A}$ выделим конечную цепочку векторов $\mathbf{A}_N = \{\mathbf{a}_{-3},\ \mathbf{a}_{-2},\ \mathbf{a}_{-1},\ \ldots,\ \mathbf{a}_{N-1}\}$.

Введем обозначения:

$$a \overset{\text{def}}{=} x_0, \quad b \overset{\text{def}}{=} x_N, \quad J_s \overset{\text{def}}{=} \{-3, -2, -1, 0, \ldots, s\}, \ s \in \mathbb{N}.$$

Сузим все функции пространства $\mathbb{S}(X, \mathbf{A}, \varphi)$ на множество $[a, b]\setminus X$. Совокупность этих сужений представляет собой конечномерное линейное пространство

$$\mathbb{S}_{[a,b]}(X_N, \mathbf{A}_N, \varphi) \stackrel{\text{def}}{=} \left\{ u \mid u = \sum_{j \in J_{N-1}} c_j \omega_j \quad \forall c_j \in \mathbb{R}^1, \ j \in J_{N-1} \right\}.$$

Теорема 4. Пусть цепочка векторов $\{\mathbf{a}_j\}_{j\in\mathbb{Z}}$ полная, а компоненты вектор-функции $\varphi(t)$ представляют собой линейно независимую систему функций на любом интервале $(c, d) \subset (\alpha, \beta)$. Тогда сужение системы функций $\{\omega_j\}_{j\in\mathbb{Z}}$ на любой интервал $(c, d) \subset (\alpha, \beta)$ является линейно независимой системой на упомянутом интервале.

Для дальнейших построений кроме векторов цепочки **A** потребуется привлечь еще векторы $\mathbf{a}_{-6}, \mathbf{a}_{-5}, \mathbf{a}_{-4}, \mathbf{a}_N, \mathbf{a}_{N+1}, \mathbf{a}_{N+2}$; заметим, что благодаря полноте бесконечной цепочки **A** конечная цепочка векторов $A' \stackrel{\text{def}}{=} \{\mathbf{a}_{-6}, \mathbf{a}_{-5}, ..., \mathbf{a}_{N+2}\}$ обладает свойством

$$\det(\mathbf{a}_{j-3}, \mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \mathbf{a}_j) \neq 0 \qquad \forall j \in \{-3, -2, ..., N+2\}.$$

Теперь построим цепочку $\{\mathbf{d}_j\}$, полагая

$$\mathbf{d}_j^T \mathbf{x} \stackrel{\text{def}}{=} \det(\mathbf{a}_{j-3}, \mathbf{a}_{j-2}, \mathbf{a}_{j-1}, \mathbf{x}),$$

где $j = -3, -2, ..., N, N+1, N+2$.

Теорема 5. В условиях теоремы 4 сужения функций ω_j образуют линейно независимую систему на $[a, b]$, для них справедливы заключения теорем 2 и 3; кроме того, $\dim \mathbb{S}_{[a,b]}(X_N, \mathbf{A}_N, \varphi) = N+3$.

Теорема 6. Функция $u_N(t) \stackrel{\text{def}}{=} \sum_{j=-3}^{N-1} c_j \omega_j(t)$, $t \in [a, b]$, является следом функции $u \stackrel{\text{def}}{=} \sum_j c_j \omega_j$ на отрезке $[a, b]$, лежит в пространстве $\mathbb{S}_{[a,b]}(X_N, \mathbf{A}_N, \varphi)$ и полностью определяется набором узлов $\{x_j\}_{j\in\{0, 1, ..., N\}}$, набором векторов $\{\mathbf{a}_j\}_{j\in J_{N-1}}$ и набором коэффициентов $\{c_j\}_{j\in J_{N-1}}$.

3. Биортогональная система функционалов и ее реализации

3.1. Справедливо следующее утверждение.

Теорема 7. Пусть цепочка векторов $\{\mathbf{a}_j\}_{j\in\mathbb{Z}}$ полная, а компоненты вектор-функции $\varphi(t)$ представляют собой линейно независимую систему функций на любом интервале $(c, d) \subset (\alpha, \beta)$. Тогда для того, чтобы система функционалов $\{g_i\}_{i\in\mathbb{Z}}$ была биортогональна системе функций $\{\omega_j\}_{j\in\mathbb{Z}}$ (а именно, $\langle g_i, \omega_i\rangle = \delta_{i,j}$ $i, j \in \mathbb{Z}$) необходимо и достаточно, чтобы $\langle g_i, \varphi\rangle = \mathbf{a}_i \quad \forall i \in \mathbb{Z}$.

Рассмотрим линейное пространство $\mathfrak{B}$ над полем вещественных чисел и сопряженное ему пространство $\mathfrak{B}^*$ линейных функционалов f над пространством $\mathfrak{B}$. Значение функционала f на элементе $v \in \mathfrak{B}$ обозначается $\langle f, v\rangle$. Множество четырехкомпонентных векторов-столбцов $\mathbf{u}$ с компонентами $[\mathbf{u}]_i$, $i = 0, 1, 2, 3$, из пространства $\mathfrak{B}$ обозначим $\mathfrak{B}^4$. Будем рассматривать также четырехкомпонентные вектор-столбцы $\mathbf{f}$, компоненты $[\mathbf{f}]_j$ которых лежат в пространстве $\mathfrak{B}^*$, $j = 0, 1, 2, 3$; образуемое этими векторами линейное пространство обозначим $\mathfrak{B}^{*4}$.

Лемма 1. Пусть $\mathbf{f} \in \mathfrak{B}^{*4}$ и $\mathbf{v} \in \mathfrak{B}^{*4}$ таковы, что матрица $\mathbf{f}\mathbf{v}^T$ — неособенная; пусть вектор $\mathbf{u} \in \mathfrak{B}^4$ удовлетворяет соотношению

$$A\mathbf{u} = \mathbf{v}, \tag{4}$$

где A — неособенная квадратная числовая матрица. Тогда система из компонентов $[\mathbf{l}]_r$ вектора $\mathbf{l} \overset{\text{def}}{=} A^T(\mathbf{f}\mathbf{v}^T)^{-1}\mathbf{f}$, $r = 0, 1, 2, 3$, представляет собой систему функционалов, биортогональную к системе элементов $[\mathbf{u}]_i$, $i = 0, 1, 2, 3$, так что $\mathbf{l}\mathbf{u}^T = I$, где I — единичная матрица четвертого порядка.

3.2. В дальнейшем потребуется линейное пространство $C\langle c, d\rangle$, состоящее из функций $u(t)$ пространства $C(c, d)$, которые имеют конечные пределы $\lim_{t\to c+0} u(t)$ и $\lim_{t\to d-0} u(t)$. Введем также пространства

$$\mathbb{C}_X \overset{\text{def}}{=} \bigotimes_{k\in\mathbb{Z}} C\langle x_k, x_{k+1}\rangle, \quad \mathbb{C}_X^S \overset{\text{def}}{=} \{u \mid u^{(i)} \in \mathbb{C}_X, \quad \forall i = 0, 1, ..., S\}.$$

Символом $(\mathbb{C}_X^S)^*$ обозначается пространство, сопряженное к пространству $\mathbb{C}_X^S$. При $\varphi \in \mathbb{C}^S(\alpha, \beta)$ пространства $\mathbb{S}(X, \mathbf{A}, \varphi)$ лежат в пространстве $\mathbb{C}_X^S$.

Для функционала $f \in (\mathbb{C}_X^S)^*$ будем писать $\operatorname{supp} f \subset [c, d]$, если значение $\langle f, u\rangle$ определяется значениями функции $u \in \mathbb{C}_X^S$ на интервале (c, d).

Теорема 8. Пусть $\varphi \in \mathbb{C}_X^S$, $\mathbf{A} \in \mathbb{A}_\infty$, и для каждого фиксированного $k \in \mathbb{Z}$ определен вектор-столбец $\mathbf{f}_k$, компонентами которого служат функционалы $[\mathbf{f}_k]_i \in (\mathbb{C}_X^S)^*$, $i = 0, 1, 2, 3$, такие, что $\operatorname{supp}[\mathbf{f}_k]_i \subset [x_k, x_{k+1}]$. Пусть матрица $\mathbf{f}_k\varphi^T$ со столбцами вида

$$\big(\langle[\mathbf{f}_k]_0, [\varphi]_i\rangle,\ \langle[\mathbf{f}_k]_1, [\varphi]_i\rangle,\ \langle[\mathbf{f}_k]_2, [\varphi]_i\rangle,\ \langle[\mathbf{f}_k]_3, [\varphi]_i\rangle\big)^T,$$

$i = 0, 1, 2, 3$, неособенная. Тогда при каждом фиксированном $r \in \{0, 1, 2, 3\}$ система $\{g_{\langle r\rangle}^{(k)}\}_{k\in\mathbb{Z}}$ функционалов $g_{\langle r\rangle}^{(k)} \in (\mathbb{C}_X^S)^*$, определяемых равенствами

$$g_{\langle r\rangle}^{(k)} \overset{\text{def}}{=} \Big[A_{k-r+3}^T(\mathbf{f}_{k-r+3}\varphi^T)^{-1}\mathbf{f}_{k-r+3}\Big]_r, \qquad k \in \mathbb{Z}, \tag{5}$$

представляет собой продолжение на $\mathbb{C}_X^S$ системы функционалов, биортогональной системе минимальных (X, $\mathbf{A}$, φ)-сплайнов $\{\omega_{k'}\}_{k'\in\mathbb{Z}}$, и

$$\left\langle g_{\langle r\rangle}^{(k)},\, \omega_{k'} \right\rangle = \delta_{k,k'} \;\; \forall k,\, k' \in \mathbb{Z}, \quad \operatorname{supp} g_{\langle r\rangle}^{(k)} \subset [x_{k-r+3},\, x_{k-r+4}] \;\; \forall k \in \mathbb{Z}.$$

Введем обозначения

$$\mathcal{W}(t) \stackrel{\text{def}}{=} (\varphi(t),\, \varphi'(t),\, \varphi''(t),\, \varphi'''(t)), \quad W(t) \stackrel{\text{def}}{=} \det \mathcal{W}(t) \quad \forall t \in (\alpha,\, \beta)\setminus X.$$

Следствие 1. Пусть $\varphi \in \mathbb{C}_X^3$, $\mathbf{A} \in \mathbb{A}_\infty$, $\langle [\mathbf{f}_k]_i,\, u\rangle \stackrel{\text{def}}{=} u^{(i)}(x_k + 0)$, $i = 0,\, 1,\, 2,\, 3$, и $W(x_k + 0) \neq 0 \quad \forall k \in \mathbb{Z}$.

Тогда при каждом фиксированном $\sigma \in \{0,\, 1,\, 2,\, 3\}$ функционалы $g_{k\langle\sigma\rangle} \stackrel{\text{def}}{=} g_{\langle 3-\sigma\rangle}^{(k)}$ обладают свойством $\langle g_{k\langle\sigma\rangle},\, \omega_{k'}\rangle = \delta_{k,k'}$ при $\forall\, k,\, k' \in \mathbb{Z}$, а кроме того, $\operatorname{supp} g_{k\langle\sigma\rangle} \subset [x_{k+\sigma},\, x_{k+\sigma} + \varepsilon]$ для $\forall \varepsilon > 0$.

Литература

1. *Завьялов Ю. С., Квасов Б. И., Мирошниченко В. Л.* Методы сплайн-функций. М.: Наука, 1980. 352 с.

2. *Демьянович Ю. К.* Всплесковые разложения на неравномерной сетке // Труды СПбМОб, 2007. Т. 13. С. 27–51.

3. *Демьянович Ю. К., Иванцова О. Н.* Гладкость пространств сплайнов третьего порядка // Сб. Математические модели. Теория и приложения, 2006. Вып. 7. СПб. С. 58–64.

О РАСПАРАЛЛЕЛИВАНИИ НЕВЫРОЖДЕННЫХ АЛГОРИТМОВ АППРОКСИМАЦИИ ОБЛАСТИ И ОБ АППРОКСИМАЦИИ ФУНКЦИЙ

Е. П. Арсентьева

Санкт-Петербургский государственный университет

Методы построения различных сеток интенсивно развивались в течение последних пятидесяти лет. Это обусловлено большим числом приложений, связанных с моделированием физических процессов и тем, что сеточные методы активно используются при численном решении задач гидродинамики, электродинамики, газовой динамики, теории упругости, при численном моделировании климата и океанических течений, а также в других областях. Для решения широкого класса задач, встречающихся на практике, активно используется метод конечных элементов (МКЭ). При решении этих задач требуется подразделить заданную область на симплексы так, чтобы углы между их ребрами были ограничены снизу некоторым положительным числом, не зависящим от мелкости подразделения.

Как правило, при моделировании физических процессов, существенное и резкое изменение параметров происходит на небольших участках рассматриваемой области, часто на границе области. В этих зонах необходимо сильно измельчать сетку для того, чтобы получить численное решение с заданной точностью. Но использование подробной равномерной сетки во всей области приводит к неоправданно большим затратам ресурсов ЭВМ, времени счёта и оперативной памяти. Таким образом, актуальным и важным разделом сеточных методов является построение адаптивных сеток, сгущающихся в зонах больших градиентов решения физической задачи.

Последние десятилетия отмечены возрождением интереса к локальной аппроксимации и, в частности, к сплайнам и всплескам. Это объясняется широким применением последних при решении задач вычислительной математики и к различным видам обработки числовых потоков — носителей информации.

В случае, когда сетка равномерная, для построения вэйвлетных разложений удаётся применить мощный аппарат гармонического анализа (в $L_2(R)$ и l_2). Этому случаю посвящено большое количество исследований. Однако, при обработке цифровых потоков с резко меняющимися характеристиками (со сменой плавного поведения на скачкообразное и наоборот) целесообразно использовать неравномерную сетку, приспосабливаемую к обрабатываемому потоку. Работ по вэйвлетам на неравномерной сетке не много, непосредственное применение гармонического анализа здесь затруднительно. При-

менение неравномерной сетки позволяет улучшить приближение функций без усложнения вычислений. Так, для улучшения приближения могут понадобиться различные степени измельчения сетки в разных частях рассматриваемого промежутка, а для сжатия приближения — различные степени укрупнения сетки.

Цель данной работы — предложить новые варианты измельчения симплициального подразделения вблизи границы области с сохранением свойства невырожденности, а также построить адаптивные сплайн-вэйвлетные разложения в пространствах сплайнов с локальным базисом на неравномерной укрупняющейся сетке. В работе выводятся соответствующие формулы реконструкции и декомпозиции, а также изучаются вопросы их распараллеливания.

В работе даётся описание алгоритмов симплициального топологически правильного подразделения параллелепипеда при измельчении симплексов с приближением к его основаниям; при этом углы всех симплексов содержатся в интервале $(\varepsilon, \pi-\varepsilon)$, где число $\varepsilon \in (0, \pi/4)$ фиксировано. Симплициальное подразделение называется топологически правильным, если его вершины не лежат внутри ребер или внутри граней подразделения. Особенность предлагаемых алгоритмов в том, что результирующее симплициальное подразделение рассматриваемого некомпактного параллелепипеда содержит бесконечное (счетное) множество симплексов; таким образом, получаемый симплициальный комплекс бесконечен. Применение описываемого алгоритма приводит в итоге к бинарному дереву аналогичных комплексов, топологически правильное подразделение которых может быть построено параллельно применением описанного алгоритма симплициального разбиения к каждому из комплексов.

Построение адаптивных сплайн-вэйвлетных разложений использует аппроксимационные соотношения, вложенность сплайновых пространств и их проектирование с помощью биортогональных систем функционалов. В случае двумерных потоков числовой информации наиболее эффективные аппроксимации (Куранта, Зламала, Аргириса) базируются на триангуляции, правильной в топологическом смысле; однако, построение связанных с этими аппроксимациями адаптивных вэйвлетов наталкивается на трудности: оказывается не всякая триангуляция допускает локальное укрупнение с сохранением правильности триангуляции; например, триангуляция, часто используемая в методе конечных элементов не допускает упомянутого укрупнения. В данной работе рассмотриваются триангуляции области, допускающие локальное укрупнение с сохранением правильности; использование этой триангуляции и ее укрупнение приводят к калибровочным соотношениям для координатных функций Р.Куранта и к вложенности соответствующих прстранств. Здесь предлагаются адаптивные сплайн- вэйвлетные разложения и выводятся соответствующие формулы декомпозиции и реконструкции двумерных числовых потоков.

Литература

1. *Сьярле Ф.* Метод конечных элементов для эллиптических задач. М., 1980. 512 с.

2. *Демьянович Ю. К.* Симплициальные распространения сеточных функций // Методы вычислений / Под ред. З. И. Царьковой. СПб.: Изд-во ЛГУ, 1973. Вып. 8. С. 32–50.

3. *Арсентьева Е. П., Демьянович Ю. К.* Алгоритмы невырожденного симплициального подразделения с измельчением вблизи границы // Компьютерные инструменты в образовании. 2010. № 6. С. 23–30.

4. *Арсентьева Е. П., Демьянович Ю. К.* О невырожденной триангуляции со сгущением к границе области // Проблемы математического анализа. 2010. Вып. 48. С. 3–14.

5. *Демьянович Ю. К., Зимин А. В.* Аппроксимации курантова типа и их вэйвлетные разложения // Проблемы математического анализа. 2008. Т. 37. С. 3–22.

ВЭЙВЛЕТЫ И ИХ ПРИМЕНЕНИЕ В КРИПТОГРАФИИ

Е. А. Литовченко

Санкт-Петербургский государственный университет

Сообщение, передаваемое получателю, называется *открытым текстом*. Изменение вида сообщения так, чтобы спрятать его суть, называется *шифрованием*. Шифрованное сообщение называется *шифротекстом*. Процесс преобразования шифротекста в открытый текст называется *дешифрованием*. Существует два основных типа алгоритмов, основанных на ключах: симметричные и с открытым ключом. В криптосистемах с открытым ключом один ключ является секретным, а второй открыт и доступен всем желающим. При этом считается, что вычислить секретный ключ по информации об открытом невозможно. При симметричном шифровании обоим сторонам известен шифрующий алгоритм, и применяется один и тот же ключ. Надежность при этом в немалой степени зависит от секретности передачи ключа.

Также криптоалгоритмы различаются по способу обработки входных данных:

- Поточный шифр обрабатывает поток данных по битам,
- Блочный шифр за один прием обрабатывает блок открытого текста.

В нашем случае будет описан симметричный блочный алгоритм шифрования данных.

В качестве ключа будет использована конечная неравномерная сетка вэйвлетного разложения. Для шифрования данных используются формулы декомпозиции. Для расшифрования — формулы реконструкции. Вэйвлетные разложения получены в [6].

При анализе криптографической стойкости принято брать наихудший вариант развития событий — когда взломщику известен алгоритм шифрования, и в его распоряжении имеется некоторое количество пар открытых и зашифрованных текстов.

Безопасность симметричной криптосистемы является функцией двух факторов: надежности алгоритма и длины ключа. При достаточно большой надежности алгоритма лучшим путем взлома криптосистемы является вскрытие грубой силой с помощью перебора всех возможных ключей.

В рассматриваемом алгоритме достаточно высокая криптостойкость была достигнута за счет увеличения надежности алгоритма. Также очевидно криптостойкость будет возрастать при увеличение размера ключа.

В криптографии вычисления производятся в кольце вычетов по модулю некоторого числа. В данном случае число должно быть простым и достаточно большим. Операции совершаются в кольце вычетов по модулю двух простых чисел p_1 и p_2, в сумме дающих 2^{8n}, где n — число байт в элементе

потока. Например для $n = 3$ количество возможных пар таких простых чисел равно 45 745. Для p_1 и p_2 своя сетка-ключ, сами числа p_1 и p_2 также входят в ключ и их значений не «видно» в алгоритме. Информация в шифруемом файле считывается в форме потока байтов. Исходный поток данных делится на 2 других потока по принципу: если элемент меньше p_1, то считывается в первый поток, иначе — от него отнимается p_1, и полученное число считывается во второй поток. Далее для обоих потоков процесс шифрования идет параллельно. Алгоритм работает в режиме, при котором ход программы разбит на итерации, и на каждой итерации используются результаты, полученные на предыдущем шаге. На каждой итерации циклично меняется k, и из общей сетки-ключа выбрасывается соответствующий узел. Считываются первые 5 элементов потока и преобразовываются с помощью вэйвлетных разложений по формулам декомпозиции. В выходном поток записывается первый элемент, остальные 4 используются на следующем шаге. Из элементов исходного потока $[c_{k-2}, c_{k-1}, c_k, c_{k+1}, c_{k+2}]$ получаются элементы $[a_{k-2}, a_{k-1}, a_k, a_{k+1}, b_k]$. В силу особенности вэйвлетного разложения первый элемент при первой итерации остается без изменения. Это создает узкое место, и для того, чтобы избежать ослабления криптоалгоритма, в начало потока дописывается случайное нечетное число (оно должно быть больше, чем $10n$) случайных байтов. В зашифрованном виде внутри шифротекста передается длина исходного потока и количество добавленных в начало случайных байт. В процессе дешифрования данные обрабатываются с конца файла , для преобразований используются формулы реконструкции.

В пакете программ реализован алгоритм разложения числа 2^{8n} на сумму простых чисел p_1 и p_2, алгоритм генерации неравномерной сетки $X_{(N)}$, алгоритм шифрования и алгоритм дешифрования.

Оценка сложности подбора ключа.

Пусть длина ключа N, тогда число всевозможных ключей в кольце вычетов по модулю p равно

$$C_p^N.$$

Т. к. при шифровании используется двойной ключ, то число возможных ключей увеличивается:

$$C_{p_1}^N C_{p_2}^N.$$

Т. к. p_1 и p_2 неизвестны, и их можно вычислить только прямым перебором, то

$$p(2^{8n}) C_{p_1}^N C_{p_2}^N,$$

где $p(2^{8n})$ — число всевозможных пар простых чисел, в сумме дающих $8n$.

Литература

1. *Смарт Н.* Криптография. М., 2005.
2. *Шнаер Б.* Прикладная криптография. 2005.
3. *Демьянович Ю. К.* Вэйвлеты & минимальные сплайны. 2003.
4. *Демьянович Ю. К., Ходаковский В. А.* Введение в вэйвлеты. 2007.
5. *Демьянович Ю. К., Косогоров О. М.* Сплайны и биортогональные системы // Сборник трудов ПОМИ. 2009. Т. 367. С. 9–26.
6. *Демьянович Ю. К., Косогоров О. М.* Сплайн-вэйвлетные разложения на открытом и замкнутом интервалах // Проблемы математического анализа. 2009. Вып. 43. С. 69–86.

Теория и практика защиты и кодирования информации

Крук
Евгений Аврамович

д.т.н., профессор
декан факультета информационных систем и защиты информации
заведующий кафедрой
безопасности информационных систем СПбГУАП

УЯЗВИМОСТИ ВЕБ-ПРИЛОЖЕНИЙ, ИСПОЛНЯЕМЫХ НА СТОРОНЕ КЛИЕНТА: JAVA-АППЛЕТЫ И JAVASCRIPT

Р. Ф. Жаринов, студент ГУАП,
А. В. Сергеев, начальник отдела ГУАП

Введение в предметную область

Клиенты и серверы неразрывно связаны между собой, поэтому модель безопасности часто разделяется между ними. Безопасность сервера зависит от безопасности клиентской стороны, в то же время безопасность клиента зависит от безопасности серверов, с которыми он взаимодействует. С одной стороны проблемы безопасности серверной стороны намного важнее, чем клиентской, но при помощи клиентских технологий могут быть проведены более масштабные и изощренные атаки (возможность проникновения злоумышленникам в надежно защищенные компьютерные системы).

На сегодняшний день JavaScript покрывают более 80 % всех веб-клиентских приложений [1], в то время как Java-апплеты занимают около 0,2 % и применяется в критических, с точки зрения безопасности приложениях: биллинг системы, веб-банкинг, электронно-цифровая подпись (ЭЦП) документов и т. д.

Цели и задачи

Одним из способов атак на получение доступа к личной (конфиденциальной) информации на стороне клиента сегодня является использование уязвимостей веб-приложений. К сожалению, среди средств, обеспечивающих безопасность клиента, на текущий момент не представлены средства, ориентированные на выявление в реальном времени угроз и атак со стороны веб-приложений и своевременное информирование пользователя (табл. 1). Задача данной статьи является — анализ технологий JavaScript и Java-апплетов с точки зрения безопасности веб-клиента, а так же выявление потенциальных источников угроз и направлений атак. Как результат, обосновывается необходимость реализации специальных средств обнаружения потенциально опасных клиентских веб-приложений в режиме реального времени.

Функционал веб-технологий

Итак, рассмотрим функциональные возможности JavaScript и Java-апплетов.

Java-апплет — полнофункциональный, объектно-ориентированный язык программирования, который компилируется в платформонезависи-

Таблица 1

ПО, обеспечивающее безопасность клиента от потенциально опасных веб-приложений

Продукт	Назначение	Плюсы	Минусы
Локальные антивирусы (Eset, AVP и др.)	Поиск вирусов и вредоносного ПО сигнатурам/эвристический анализ.	Обнаружение и борьба с вирусами	Не ориентированы на клиентские веб-приложения Не отличают вредоносные клиентские веб-приложения от «полезных» Требуют профессиональной настройки
Межсетевые экраны (Outpost Firewall и др.)	Контроль сетевых потоков, поиск рекламы, нежелательных сайтов и утечки ПД	Удаление рекламы Возможность блокировки сценариев, сайтов Блокировка передали ПД	
Внутренние средства браузеров	«AntiDos js»	Блокирование многочисленных js окон Возможность контроля браузера (доступ к страницам)	Очень ограниченный функционал

мый байт-код, выполняемый интерпретатором, называемым виртуальная Java машина (JVM, Java Virtual Machine). Сам по себе Java-апплет, в отличие от других программ, написанных на языке Java, выполнятся не может — присутствие браузера тут обязательно (хотя есть возможность отлаживать Java-апплет в программе AppletViewer от Sun). Большинство современных браузеров поддерживают Java путем дополнительной загрузки Java-модуля (как правило, это происходит при первом посещении какой-нибудь веб-страницы с Java-апплетом). Java-апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не могут быть предоставлены HTML и JavaScript. Когда web-браузер видит ссылки на Java-код, он загружает код и затем выполняет его, используя встроенную JVM. Так как байт-код Java платформо-независим, то Java-апплеты могут выполняться браузерами многих платформ, поддерживающих графические оболочки, таких как: Microsoft Windows, Linux/UNIX и Apple Mac OS.

JavaScript — кроссплатформенный скриптовый язык программирования, который может быть встроен в стандартные веб-страницы для создания динамических веб-сайтов. Веб-браузеры, интерпретируют операторы клиентского JavaScript, внедренные в HTML-страницу и могут реагировать на пользовательские события, такие как щелчок мыши, ввод данных в форму (проверка правильности введённой пользователем информации — номер телефона, ИНН, …), навигация по странице и т. д.. Без передачи по сети,

JavaScript может проверить введенную информацию и вывести диалоговое окно, если пользователь ввел неверные данные.

Сравнение по основным критериям вышеописанных веб-технологий представлены в табл. 2.

Таблица 2

Сравнение JavaScript с Java-апплетами

JavaScript	Java-апплеты
Интерпретируется (не компилируется) клиентом	Скомпилированные байт-коды загружаются с сервера, выполняются на клиенте
Предварительная валидация форм	Возможность распределения бизнес-логики между сервером и клиентом (почти десктоп-приложения)
Объектно-ориентированный. Нет отличий в типах объектов. Наследование идёт через механизм прототипов, а свойства и методы могут динамически добавляться к любому объекту	На основе классов. Объекты делятся на классы и экземпляры с наследованием по всей цепи иерархии классов. Классы и экземпляры не могут иметь свойства или методы, добавляемые динамически
Тип данных переменной не объявляется (динамическая типизация)	Тип данных переменной обязан быть объявлен (статическая типизация)
Нет доступа к файловой системе (ФС) пользователя	Полный доступ к ФС пользователя (если java-апплет подписан)
Полная поддержка и интеграция в современных браузерах	Установка расширения для браузера
Возможность параллельной обработки JavaScript во время загрузки страницы	Пока JVM не запуститься, Java-апплет не будет работать

Архитектура безопасности

Перейдем к обзору архитектуры безопасности веб-технологий (JavaScript и Java-апплет). Условно можно разделить на встроенную, для каждой технологии своя, и расширяемую (в основном это предложения для увеличения безопасности веб-приложений). Рассмотрим подробнее *встроенную* безопасность исследуемых веб-технологий.

JavaScript не позволяет выполнять потенциально опасный код, это обуславливает наличие двух принципиальных ограничений:

- JavaScript-программы выполняются в так называемой «песочнице», в которой они могут выполнять только ограниченный круг действий — манипуляция с объектной моделью (DOM, Document Object Model), доступ к привилегированным функциям (например, создание файлов, работа с сокетами) запрещен;

- для JavaScript-кода применяется политика общего происхождения, в соответствии с которой скрипт, встроенный в страницу, не может получить доступ к свойствам объектов другой страницы (в частности, к свойствам объекта document) при отличии в протоколе, хосте и номере порта этих страниц. Более того, браузер обычно ограничивает выполнение скрипта той станицей, на которую он был загружен.

Java-апплеты так же придерживается модели безопасности «песочница», использующей изолирование памяти и методов доступа, поддержку нескольких независимых доменов выполнений, а так же предотвращения выполнения неавторизованных операций, например, просмотр или изменение файлов в файловой системе клиента и использование сетевых соединений. Код апплета загружается с веб-сервера, и браузер либо вставляет апплет в веб-страницу, либо открывает новое окно с собственным пользовательским интерфейсом апплета. Java-апплет может быть отображен на веб-странице путем использования устаревшего HTML элемента *applet*, или рекомендуемого элемента *object* [2]. Java-апплет работает по схеме подписей и цифровых сертификатов. Чтобы разрешить доступ апплету к привилегированным функциям, таким как чтение, модификация и удаление файлов на компьютере пользователя, создание сетевых подключений за пределами межсетевого экрана, а так же запуск сторонних программ/библиотек, необходимо получить доверенный или самоподписанный цифровой сертификат и подписать им Java-апплет.

При получении любого Java-апплета с ЭЦП, браузер выводит предупреждение о получении дополнительных прав. Данные предупреждения визуально почти идентичны и, как следствие, уровень безопасности зависит от решения пользователя см. рис. 1. Такая ситуация может увеличить риск выполнения потенциально-опасного кода.

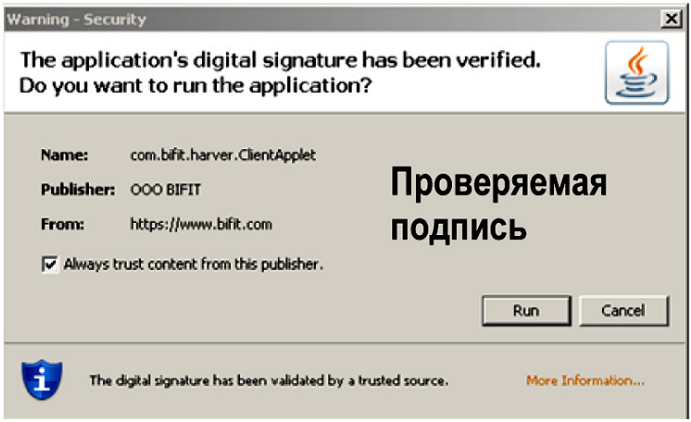


Рис. 1, *а*. Предупреждение JMV с правильной ЭЦП

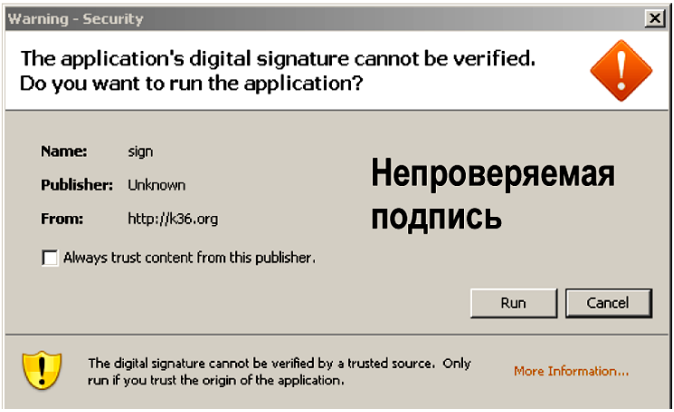


Рис. 1, *б*. Предупреждение JMV с неправильной ЭЦП

Расширенная безопасность может обеспечиваться:

- Создание безопасного соединения между клиентом и сервером. В веб-среде под безопасным соединением обычно понимают использование HTTPS (Hypertext Transfer Protocol Secure) протокола. Наиболее безопасный метод — использование двусторонней аутентификации по стандарту SSL;
- Использование обфускаторов для исходного кода. Под обфускацией понимается преобразование кода с целью максимально затруднить его анализ и модификацию. Суть работы любого обфускатора апплетов — с помощью определенного алгоритма внести в код изменения, значительно затрудняющие корректную декомпиляцию, сохранив при этом работоспособность самой программы;
- Использование разрабатываемого детектора потенциально-опасных клиентских веб-приложений.

Потенциальные атаки с использованием JavaScript и Java-апплетов

Выделим риски безопасности связанные с рассматриваемыми веб-технологиями (см. табл. 3).

Таблица 3

Потенциальные атаки на JavaScript/Java-апплеты

Потенциальная атака	Исполнить на JavaScript	Исполнить Java-апплетом
DoS — генерация окон	+ while (true) {alert(‘hello’)}	+ while(true) {JOptionPane. showMessageDialog(this, “hello”);}

Потенциальная атака	Исполнить на JavaScript	Исполнить Java-апплетом
DoS — нагрузка ЦПУ и память	+ while (true) {some_calc()}	+ while (true) {some_calc()}
Сбор информации доступной браузеру: email, ПД, cookie...	+ document.getElement-ById(…)	+* Вызов js кода из класса
Обмен информации с сервером без перезагрузки страницы (AJAX)	+ new XMLHttpRequest()…	+ URLConnection()
Доступ к файловой системе — чтение, модификация файлов	–	+ Только подписанный апплет
Запуск сторонних программ/ библиотек	–	+ Только подписанный апплет

Возможности системы детектирования потенциальных атак

На данном этапе реализуемая система обнаружения атак позволяет детектировать большинство из приведенных угроз безопасности (см. таблицу 3), такие как доступ к ФС пользователя, попытка запуска сторонних программ, возможность обмена с сервером при помощи технологии AJAX, доступ к cookie текущего сайта.

Алгоритм работы детектора уязвимостей веб-приложений.

Рассмотрим алгоритм работы детектора (см. рис. 3):

- При загрузке страницы производится поиск всего JavaScript кода (как внутреннего, так и внешнего) и всех Java-апплетов (представляет собой Jar архив, состоящий, в основном, из class-файлов);

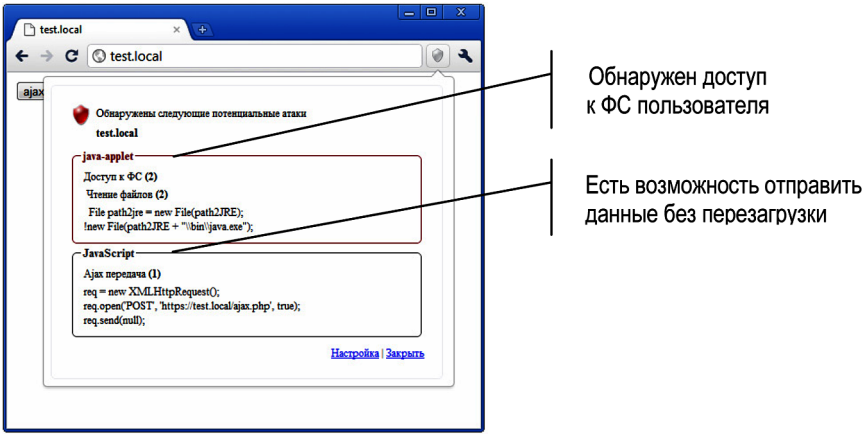


Рис. 2. Пример графического интерфейса системы

- Полученный Jar архив распаковывается во временный каталог;
- Поскольку исходники Java скомпилированы в байт-код и содержаться в class-файлах, декомпилировать (преобразовать) их обратно в исходный код не составляет особого труда. В разрабатываемом решении используется бесплатный java декомпилятор Jad;

• С использованием базовых сигнатур ищется потенциально опасный код, и если он обнаружен, выводится предупреждение пользователю (см. рис. 2).

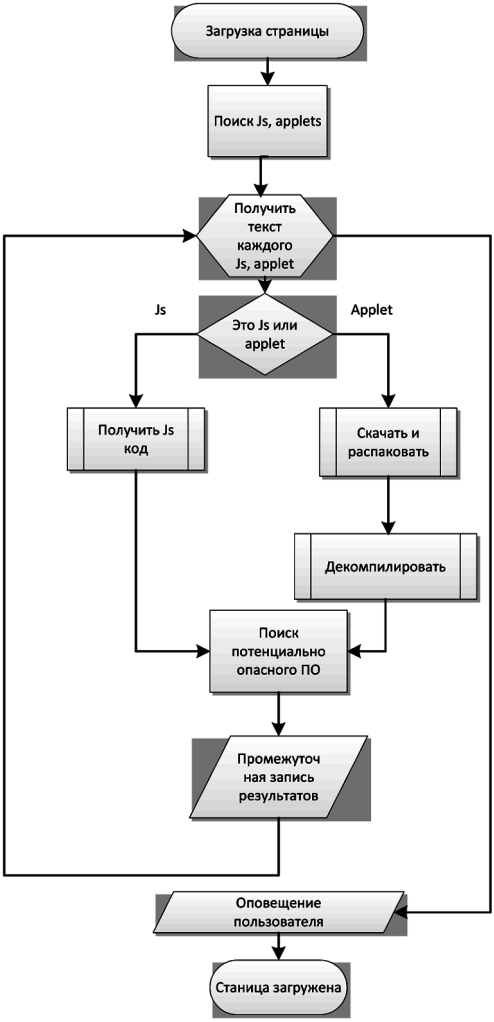


Рис. 3. Алгоритм работа детектора потенциальных угроз

В качестве итога работ по оценке рассматриваемых веб-технологий можно сделать вывод, что встроенные средства безопасности браузера не обеспечивают необходимый уровень защиты. Java-апплеты и Javascript-программы могут быть использованы для различных вредоносных действий на стороне пользователя: сбор конфиденциальной информации доступной браузеру, передача собранной информации на сторонние сайты потенциального нарушителя, модификация файлов на машине пользователя, и т. д. (см. табл. 3). Более того, наличие специализированных средств защиты, таких как межсетевой экран или антивирус предотвращает лишь часть атак, т. к. уровень безопасности во многом определяется действиями пользователя. Например, в разделе «Архитектура безопасности» показано, что успех атакующего при подмене Java-апплета в процессе передачи полностью зависит от того, обратит ли пользователь внимание на тип предупреждения при загрузке апплета (является ли подпись Java-апплета доверенной или нет). С учетом того, что оба предупреждения отличаются весьма незначительно (см. рис. 1) задача атакующего существенно упрощается. Подчеркнем, что для легальных апплетов, использующих самоподписанный цифровой сертификат, нет даже такой защиты и их подмена в процессе передачи на любой вредоносный апплет является тривиальной задачей. По итогам представленной работы было решено реализовать специализированный детектор потенциально опасного программного обеспечения (ПО) Java-апплетов и Javascript-программ. Использование подобного ПО, с одной стороны, может помочь системным администраторам, разработчикам ПО и специалистам в области безопасности упростить задачу по анализу веб-систем, построенных на основе указанных технологий. С другой стороны, такое ПО будет являться узкоспециализированным антивирусом, и поможет низкоквалифицированным пользователям защитить свои личные данные и рабочее место от атак через веб.

Литература

1. Usage of client-side programming languages for websites. http://w3techs.com/technologies/overview/client_side_language/all

2. Deploying Java Applets With Family JRE Versions in Java. http://www.oracle.com/technetwork/java/javase/family-clsid-140615.html

3. *Scott Oaks.* Java Security. Second Edition. O'Reilly Media, 2001.

4. *Jonathan Knudsen*. Java Cryptography. O'Reilly Media, 1998

5. Java security evolution and concepts. Part 3: Applet security. http://www.javaworld.com/javaworld/jw-12-2000/jw-1215-security.html?page=1

6. Java 2 Platform Security. http://www.informit.com/articles/article.aspx?p=433382&seqNum=2

7. Java-апплеты. http://corpsite.ru/(X(1)S(lvilzw55zfwuq455qelv3zqy))/Encyclopedia/Technology/Language/JavaApplet%20.aspx?AspxAutoDetectCookieSupport=1

СИСТЕМА РАЗДЕЛЕНИЯ СЕКРЕТА ОБЩЕГО ВИДА, ПОЗВОЛЯЮЩАЯ СКРЫВАТЬ СТРУКТУРУ ДОСТУПА

А. Ю. Абрамов,
студент ГПУ

Аннотация. В статье описан алгоритм построения системы разделения секрета (СРС), реализующей структуру доступа общего вида, на основе односторонних функций. Система позволяет восстанавливать секрет без публикации структуры доступа, вместо которой публикуется косвенная информация, используя которую каждый из участников структуры доступа может определить свою принадлежность к той или иной коалиции. Все СРС общего вида, известные на данный момент, основываются на том, что структура доступа известна при восстановлении секрета.

Введение

Ограничение доступа к конфиденциальной информации в любой информационной системе является важной задачей. Обычно доступ к ней может быть получен при помощи ключа или пароля, а в том случае, если ключ должен быть разделен между некоторой группой пользователей системы таким образом, что только совместные действия этой группы приведут к получению ключа, система в целом становится более надежной. Задача распределения ключей (также, ключи называют проекциями секрета или тенями) между пользователями таким образом, чтобы доступ к секретной информации имели только допустимые группы пользователей, называется задачей разделения секрета.

Множество групп пользователей, имеющих доступ к секрету, называются структурой доступа. Если структура доступа произвольна (наиболее известно построение систем разделения секрета для пороговых структур доступа [1, 2]), то задача разделения секрета для нее называется задачей разделения секрета общего вида общего вида. При этом допустимая коалиция обладает свойством, что если произвольное множество участников A' включает в себя допустимую коалицию A, то A' тоже является допустимой коалицией, то есть принадлежит структуре доступа [3]. Схемы, позволяющие разделять секрета между пользователями для структуры доступа общего вида, называются системами разделения секрета, реализующими структуру доступа общего вида. Формально система разделения секрета может быть описана следующим образом:

- пусть имеется n пользователей с номерами $\{1, \ldots, n\}$;
- подмножество A множества $\{1, \ldots, n\}$ будет называться допустимой коалицией, если пользователи с номерами из A совместно имеют право доступа к некоторой информации (секрету);
- $\Gamma = \{A_i\}, i \in \{1, \ldots, m\}$ — множество допустимых коалиций системы разделения секрета называется структурой доступа;
- $A \in \Gamma, A \subseteq A' \Rightarrow A' \in \Gamma$;

- если множество $B \notin \Gamma$, то оно называется недопустимой коалицией, и пользователи такого множества не получают доступа к секрету.

Построение всех известных на данный момент систем разделения секрета общего вида основывается на том факте, что при восстановлении секрета известна структура доступа, т. к. в противном случае невозможно восстановить секрет для коалиции $A' : A \in \Gamma, A \subseteq A'$ за линейное от количества участников время в силу того, что потребуется перебор либо для нахождения самой допустимой коалиции A', либо для поиска подходящего ключа, в случае, если у участника их несколько. Однако публиковать структуру доступа не всегда безопасно. Поэтому представляет интерес построение системы, в которой информация о структуре доступа публикуется в неявном виде, и существует возможность проверить, является ли собравшаяся коалиция участников допустимой, используя ключи участников, состоящих в ней.

Описание схемы

В данном разделе описывается предлагаемая система разделения секрета для структуры доступа общего вида. Система основана на использовании односторонней функции от двух переменной. Существование таких функций и примеры их построения описаны в [4]. Одним из свойств односторонней функции такого вида является следующее: при известных значениях $h(x, i_1)$, $h(x, i_2)$, ..., $h(x, i_k)$, вычисление значения $f(x, r)$, $r \neq i_j, j \in \{1, ..., k\}$ является вычислительно сложной задачей.

Все известные на данный момент системы разделения секрета общего вида основываются на том, что при восстановлении секрета известна структура доступа. Особенностью предлагаемой системы является то, что она позволяет не публиковать структуру доступа, а вместо этого выдавать некоторую косвенную информацию. Используя эту информацию и свой секретный ключ, пользователь может определить, каким допустимым коалициям он принадлежит. Благодаря этому секрет может быть восстановлен не только для тех коалиций, которые описаны в структуре доступа, но и для коалиций $A' : A \in \Gamma, A \subseteq A'$ то есть включающих в себя допустимые.

Предлагаемая схема является одноразовой, т. к. для того, чтобы восстановить секрет, требуется передача ключа пользователя пользователю, собирающему секрет, для проверки его (пользователя) принадлежности к структуре доступа. Это отличает её от большинства систем, основанных на использовании односторонних функций, которые являются многоразовыми, однако, позволяет несколько уменьшить размер публичной информации и упростить восстановление секрета.

Фаза инициализации

Дилер выбирает двухместную одностороннюю функцию $h(x, y)$ и публикует её.

Фаза выдачи ключей

Дилер выбирает n случайных независимых величин $\{x_1, ..., x_n\}$, которые являются ключами пользователей системы с соответствующими номерами, и передает эти ключи пользователям по защищенному каналу.

Для каждой коалиции A_j, $j \in \{1, ..., m\}$ дилер вычисляет:

1. $P_j = S \oplus h(x_{i_1} \oplus x_{i_2} \oplus ... \oplus x_{i_k}, j), \{i_1, ..., i_k\} \equiv A_j$.
2. $g_{i_k j} = h(x_{i_k}, j)$ для всех $i_k \in A_j$.

После этого вычисленные значения публикуются. Значение $g_{i_k j} = h(x_{i_k}, j)$ однозначно идентифицирует пользователя в коалиции j.

Фаза восстановления секрета

Каждый из пользователей собравшейся коалиции посылает ключи тому, кто выбран в качестве восстанавливающего секрет («combiner»).

Для каждого ключа x_i вычисляются значения $g'_{ij} = (x_i, j), j \in \{1, ..., m\}$, а затем эти значения сравниваются с опубликованными. После выполнения этой процедуры определяется, собралась ли допустимая коалиция пользователей, а если да, то найти тех пользователей, которые в неё входят.

После этого если собралась допустимая коалиция A_j, то секрет восстанавливается как $S = P_j \oplus h(x_{i_1} \oplus x_{i_2} \oplus ... \oplus x_{i_k}, j), \{i_1, ..., i_k\} \equiv A_j$.

Анализ безопасности схемы

Безопасность схемы основывается на сложности поиска коллизий односторонней функции h и том факте, что нахождение $h(x, i)$ при известных значениях $h(x, i_1)$, $h(x, i_2)$, ..., $h(x, i_k)$, $i_j \neq i$ является сложной задачей. В силу того, что структура доступа неизвестна недопустимой коалиции участников полностью, достаточно большая часть атак, которые осуществимы на классические системы разделения секрета, к предложенной схеме неприменима.

Основные виды атак:

- Поиск перебором значения ключа x_i i-того участника структуры доступа.
- После нескольких успешных попыток восстановления секрета пользователи могут собрать часть информации о структуре доступа. После чего существует возможность искать ключ x по известным значениям $h(x, 1)$, $h(x, 2)$, ..., $h(x, m)$.

Анализ эффективности схемы

1. Размер секретного ключа

В предложенной схеме размер секретного ключа не отличается от размера, используемого в других схемах, основанных на использовании одно-

сторонних функций, и может быть вычислен как $\log_2 p$, где p — параметр безопасности, задающий требуемую стойкость системы.

2. Размер публичного ключа

В предложенной схеме публичный ключ состоит из двух частей. Одна содержит информацию о секрете для каждой коалиции P_j. Вторая содержит информацию о структуре доступа и состоит из значений $g_{i_k j}$.

Общий размер публичного ключа может быть вычислен как

$$Pub = m^* | h(\bullet) | + \sum_{j=1}^{m} | A_j | * | h(\bullet) |,$$

где $|A_j|$ обозначает количество участников в j-той коалиции, а $| h(\bullet) |$ — размер значений односторонней функции.

3. Сложность фазы выдачи ключей и восстановления секрета

Сложность обеих фаз одинакова и может быть найдена как

$$m^* O(h(\bullet)) + \sum_{j=1}^{m} | A_j | * O(h(\bullet)),$$

где $O(h(\bullet))$ — сложность вычисления односторонней функции, а $|A_j|$ — количество участников в j-той коалиции.

Заключение

В статье описана одноразовая система разделения секрета для структур доступа общего вида, позволяющая эту структуру скрыть. Данная система наиболее эффективна в случае динамически изменяющейс я структуры доступа, что не позволит недопустимым коалициям собрать достаточно информации для использования эффективных методов атак. Также, непосредственно способ разделения и восстановления секрета может быть использован в других схемах на основе односторонних функций, т. к. является более вычислительно эффективным (односторонняя функция вычисляется один раз для всей допустимой коалиции).

Представляет интерес дальнейшее развитие возможности скрывать структуру доступа. В данной схеме с помощью своего ключа может самостоятельно выяснить, каким коалициям он принадлежит, что не является достаточно безопасным. Необходимо улучшить процедуру проверки принадлжености коалиции к множеству допустимых следующим образом.

Пусть $f(\bullet)$ — процедура проверки коалиции на принадлежность к множеству допустимых. Тогда она должна обладать следующими свойствами:

1. $A \in \Gamma \Leftrightarrow f(A) = \text{true}$.
2. $A' : A \subset A', A \in \Gamma \Rightarrow f(A') = \text{true}$.
3. $O(f(\bullet)) \sim u$, где u — количество участников структуры доступа.

Литература

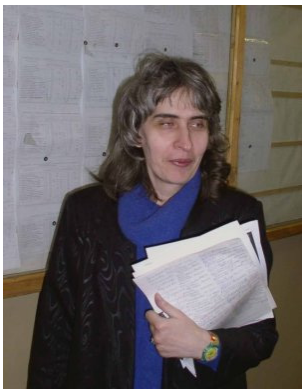
1. *Adi Shamir*. How to share a secret. 1979. Communications of the ACM 22 (11): 612–613.

2. *Blakley G. R.* Safeguarding cryptographic keys. 1979. Proceedings of the National Computer Conference 48: 313–317.

3. *M. Ito, A. Saito, T. Nishizeki.* Secret sharing scheme realizing general access structure. 1987.

4. *He J.*, and *Dawson E*. 1994. Multistage Secret Sharing Based on One-Way Function // Electronics Letters. Vol. 30. No. 19. Pp. 1591–1592.

Сплайновые приближения и вопросы распараллеливания в OpenMP



Бурова
Ирина Герасимовна

д.ф.-м.н.
профессор кафедры параллельных алгоритмов СПбГУ

РАСПАРАЛЛЕЛИВАНИЕ ПОСТРОЕНИЯ СРЕДНЕКВАДРАТИЧЕСКОГО ПРИБЛИЖЕНИЯ МИНИМАЛЬНЫМИ СПЛАЙНАМИ ТРЕТЬЕГО ПОРЯДКА АППРОКСИМАЦИИ

К. Ю. Бондаренко

Санкт-Петербургский государственный университет

Построение полиномиальных интерполяционных минимальных базисных сплайнов, удобных для построения приближений третьего порядка аппроксимации. Рассмотрим упорядоченную сетку узлов:

$$\ldots < x_{j-1} < x_j < x_{j+1} < \ldots$$

На промежутке $[x_j, x_{j+1}]$ рассмотрим систему уравнений:

$$\begin{cases} \omega_j(x) + \omega_{j+1}(x) + \omega_{j+2}(x) = 1, \\ x_j\omega_j(x) + x_{j+1}\omega_{j+1}(x) + x_{j+2}\omega_{j+2}(x) = x, \\ x_j^2\omega_j(x) + x_{j+1}^2\omega_{j+1}(x) + x_{j+2}^2\omega_{j+2}(x) = x^2 \end{cases}$$

относительно $\omega_j(x)$, $\omega_{j+1}(x)$, $\omega_{j+2}(x)$, считая, что $\operatorname{supp}\omega_k = [x_{k-2}, x_{k+1}]$.

Для вычисления $\omega_j(x)$ на промежутках $[x_{j-2}, x_{j-1}]$ и $[x_{j-1}, x_j]$ рассматриваются аналогичные системы уравнений. Объединяя результаты, получаем формулу полиномиального базисного сплайна:

$$\omega_j(x) = \begin{cases} \dfrac{(x-x_{j-1})(x-x_{j-2})}{(x_j-x_{j-1})(x_j-x_{j-2})}, & x \in [x_{j-2},\ x_{j-1}]; \\ \dfrac{(x-x_{j+1})(x-x_{j-1})}{(x_j-x_{j+1})(x_j-x_{j-1})}, & x \in [x_{j-1},\ x_j]; \\ \dfrac{(x-x_{j+1})(x-x_{j+2})}{(x_j-x_{j+1})(x_j-x_{j+2})}, & x \in [x_j,\ x_{j+1}]. \end{cases}$$

Построение аппроксимаций. Рассмотрим аппроксимации функции $u(x)$ на $[x_j, x_{j+1}]$ вида

$$\tilde{u}(x) = u(x_j)\,\omega_j(x) + u(x_{j+1})\,\omega_{j+1}(x) + u(x_{j+2})\,\omega_{j+2}(x).$$

Нетрудно проверить, что в случае полиномиальных сплайнов:

$$u(x) - \tilde{u}(x) = 0, \quad u = 1, x, x^2.$$

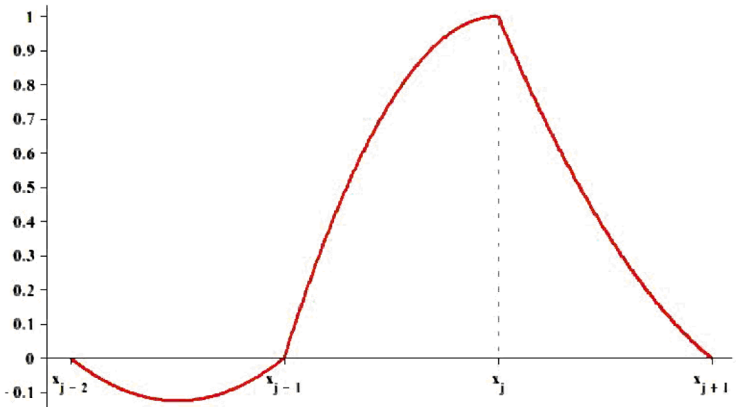


Рис. 1. Полиномиальный базисный сплайн

Также верно, что $|u(x)-\tilde{u}(x)|=O(h^3)$ для $u\in C^3[x_{j-1}, x_{j+2}]$.

Построение приближений методом наименьших квадратов. Рассмотрим на $[x_j, x_{j+1}]$ приближение для $u(x)$ вида

$$\tilde{\tilde{u}}(x)=c_j\omega_j(x)+c_{j+1}\omega_{j+1}(x)+c_{j+2}\omega_{j+2}(x),$$

где c_k находим с помощью метода наименьших квадратов. Пусть дан промежуток $[a, b]$. Возьмем целое число n и найдем шаг сетки $h=\dfrac{b-a}{n}$. Таким образом, построили сетку равноотстоящих узлов $x_k=a+kh$, $k=0, ..., n$. С каждым узлом сетки свяжем базисную функцию ω_j. Коэффициенты c_k находим с помощью решения системы уравнений $MC=F$, где $M=\left((\omega_i, \omega_j)\right)_{i,\,j=0}^{n+1}$, $F=\left((\omega_i, u)\right)_{i=0}^{n+1}$. Приведем M и F для $n=4$:

$$M=\begin{pmatrix}
(\omega_0,\omega_0) & (\omega_1,\omega_0) & (\omega_2,\omega_0) & 0 & 0 & 0\\
(\omega_0,\omega_1) & (\omega_1,\omega_1) & (\omega_2,\omega_1) & (\omega_3,\omega_1) & 0 & 0\\
(\omega_0,\omega_2) & (\omega_1,\omega_2) & (\omega_2,\omega_2) & (\omega_3,\omega_2) & (\omega_4,\omega_2) & 0\\
0 & (\omega_1,\omega_3) & (\omega_2,\omega_3) & (\omega_3,\omega_3) & (\omega_4,\omega_3) & (\omega_5,\omega_3)\\
0 & 0 & (\omega_2,\omega_4) & (\omega_3,\omega_4) & (\omega_4,\omega_4) & (\omega_5,\omega_4)\\
0 & 0 & 0 & (\omega_3,\omega_5) & (\omega_4,\omega_5) & (\omega_5,\omega_5)
\end{pmatrix}.$$

Матрица M — пятидиагональная, вследствие ограниченности носителя ω_i, симметричная, положительно определенная, вследствие линейной независимости базисных сплайнов. Интегралы, определяющие значения (ω_i, ω_j), вычисляются аналитически и зависят только от параметра h:

$$(\omega_0, \omega_0) = \int_{x_0}^{x_1} \omega_0^2(x)\,dx = \frac{31h}{120},$$

$$(\omega_1, \omega_0) = \int_{x_0}^{x_1} \omega_0(x)\,\omega_1(x)\,dx = \frac{23h}{120},$$

$$(\omega_1, \omega_1) = \int_{x_0}^{x_2} \omega_1^2(x)\,dx = \frac{19h}{24},$$

$$(\omega_2, \omega_0) = \int_{x_0}^{x_1} \omega_1(x)\,\omega_2(x)\,dx = \frac{h}{30}.$$

В программе используем заранее вычисленные значения. При $n \geq 4$ получается 9 различных элементов.

$$F = \begin{pmatrix} (u, \omega_0) \\ (u, \omega_1) \\ (u, \omega_2) \\ (u, \omega_3) \\ (u, \omega_4) \\ (u, \omega_5) \end{pmatrix},$$

$$(u, \omega_0) = \int_{x_0}^{x_1} u(x)\,\omega_0(x)\,dx, \qquad (u, \omega_1) = \int_{x_0}^{x_2} u(x)\,\omega_1(x)\,dx,$$

$$(u, \omega_2) = \int_{x_0}^{x_3} u(x)\,\omega_2(x)\,dx, \qquad (u, \omega_3) = \int_{x_1}^{x_4} u(x)\,\omega_3(x)\,dx.$$

Эти интегралы вычисляем в программе приближенно с погрешностью не менее $O(h^3)$ по формуле Гаусса.

Решение системы уравнений $MC = F$ методом Некрасова. Метод Некрасова решения СЛАУ основан на разложении $M = L + D + R$, где

[Sorry. Ignored `\begin{gather*} ... \end{gather*}`]

Исходная система уравнений приобретает вид $(L + D + R)\,C = F$, и после алгебраических преобразований получается формула

$$C = -(D^{-1}L)\,C - (D^{-1}R)\,C + D^{-1}F,$$

позволяющая построить итерационный процесс:

$$c_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{m_{i,j}}{m_{i,i}} c_j^{(k+1)} - \sum_{j=i+1}^{n+2} \frac{m_{i,j}}{m_{i,i}} c_j^{(k)} + \frac{f_i}{m_{i,i}}.$$

Итерационный процесс сходится, т. к. матрица M симметричная и положительно определенная.

Программная реализация. Расспараллеливание программы осуществлялось средствами OpenMP. Вычисления можно разделить на три части. Расчет элементов вектора F — компоненты независимы, распределяются по потокам выполнения. Итерационный процесс — существует зависимость между компонентами, для параллельного счета отрезок $[a, b]$ делиться на количество потоков. Оценка близости к решению — взаимодействие потоков с помощью двойной проверки и блокировки.

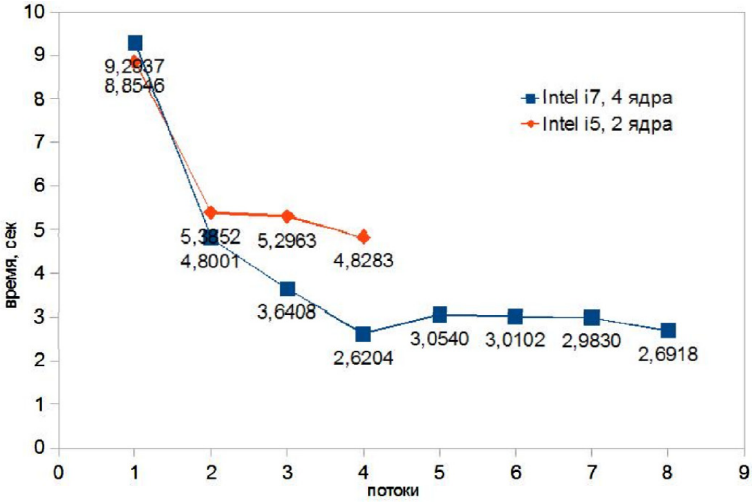


Рис. 2. Разультат численного эксперимента

Литература

1. *Бурова И. Г., Демьянович Ю. К.* Теория минимальных сплайнов. СПб.: Изд-во СПбГУ, 2000.

2. *Бурова И. Г.* Приближения минимальными сплайнами максимального дефекта. СПб., 2007.

3. *Березин И. С., Жидков Н. П.* Методы вычислений. М., 1962.

4. *Фаддеев Д. К., Фаддеева В. Н.* Вычислительные методы линейной алгебры. М., 1963.

О ВЫЧИСЛЕНИИ ОДНОГО ОПРЕДЕЛИТЕЛЯ

И. Г. Бурова, О. В. Родникова

При выводе формул базисных сплайнов, необходимых для решения интерполяционной задачи Эрмита с помощью минимальных сплайнов ненулевой высоты [1], возникает необходимость вычисления определителя

$$W_M = \left| \chi_1,\ \chi_1',\ ...,\ \chi_1^{(s_1)},\ ...,\ \chi_M,\ \chi_M',\ ...,\ \chi_M^{(s_M)} \right|, \tag{1}$$

где

$$\chi_j = \begin{pmatrix} 1 \\ \varphi(x_j) \\ \varphi^2(x_j) \\ \vdots \\ \varphi^m(x_j) \end{pmatrix},$$

а для целых чисел m, M, s_0, ..., s_M выполняется соотношение

$$m + 1 = \sum_{j=1}^{M} s_j + M.$$

Справедливо следующее соотношение:

Лемма. Определитель (1) может быть вычислен по следующей формуле:

$$W_M = \prod_{i=1}^{M} (1! \cdot 2! \cdots s_i!) \times$$

$$\times \prod_{i=1}^{M} (\varphi'(x_i))^{\frac{s_i(s_i+1)}{2}} \prod_{1 \le j < i \le M} (\varphi(x_i) - \varphi(x_j))^{(s_i+1)(s_j+1)}.$$

Формула для вычисления определителя получается дифференцированием определителя Вандермонда и соответствующей заменой переменных.

Литература

1. *Бурова И. Г., Демьянович Ю. К.* Теория миниимальных сплайнов. СПб., 2000. 316 с.

OPENMP НА КЛАСТЕРЕ: ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

А. А. Красноперов

OpenMP — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран для программирования многопоточных приложений на многопроцессорных системах с общей памятью. Он реализует параллельные вычисления с помощью многопоточности, в которых «главный» (master) поток создает набор подчиненных (slave) потоков и задача распределяется между ними, при этом количество создаваемых потоков может регулироваться как самой программой при помощи вызова библиотечных процедур, так и извне, при помощи переменных окружения.

Рассмотрим практическое применение параллельных вычислений на примере задачи вычисления скалярного произведения векторов. Для примера будем использовать простейший последовательный алгоритм. Ниже приведен фрагмент кода:

```
#define N 110000000
double gScaledSum = 0;
int main()
{
     s = 1.011; // Масштабный коэффициент
     for (i = 0; i < N; i++)
     {
       a[i] = 1.034; // Первый вектор
       b[i] = 1.057; // Второй вектор
     }
     start = omp_get_wtime();
     for ( i = 0; i < N; i++ )
       {
             gScaledSum += s * a[i] + b[i];
       }
     stop = omp_get_wtime();
}
```

Для выполнения кода и замера времени выполнения операции суммирования был использован кластер на 8 машин при Санкт-Петербургском государственном университете. Характеристики кластера:

- 7 машин с процессором Intel Pentium D;
- 1 машина с процессором Intel Dual Core;
- 2 Гб оперативной памяти на процессор;

- 100Mbit LAN;
- Каждому ядру выделяется фиксированный кеш.

Среднее время выполнения суммирования в последовательном алгоритме составило 0.9601675 сек.

Теперь рассмотрим модификацию исходного алгоритма для применения параллельных вычислений. Будем разбивать вычисления по шагам:

```
#define gNumThreads <количество потоков>
#define N 110000000
void *threadFunction(void *arg)
{
     int i;
     int myNum = *((int *)arg);
     double partialSum = 0;
     for ( i = myNum; i < gNumSteps; i += gNumThreads )
     {
       partialSum += c * a[i] + b[i];
     }
     gVectorSum += partialSum;
     return 0;
}
int main()
{
     pthread_t tid[gNumThreads];
     …
     for (i = 0; i < gNumThreads; i++)
     {
       tNum[i] = i;
        pthread_create(&tid[i], NULL, threadFunction,
     &tNum[i]);
     }
     for (i = 0; i < gNumThreads; i++)
       pthread_join(tid[i], NULL);
     …
}
```

Протестировав данный алгоритм при различном количестве потоков мы получили следующие результаты:

Среднее время:

- 1 поток: 0.8934866;
- 2 потока: 0.7380924 (−17,4%);
- 4 потока: 1.4592800 (+63,3%).

Теперь модифицируем наш алгоритм, будем использовать разбиение на блоки:

```
void *threadFunction(void *arg)
{
      …
      for ( i = myNum*(gNumSteps / gNumThreads);
 i < (myNum+1)*(gNumSteps / gNumThreads); i++ )
      {
        partialSum += c * a[i] + b[i];
      }
      gVectorSum += partialSum;
      pthread_exit(0);
}
```

Среднее время в данном случае:

- 1 поток: 0.899721;
- 2 потока: 0.4804558 (−46,6%);
- 4 потока: 0.4798832 (−46,7%).

Таким образом, в ходе выполнения работы были сделаны следующие выводы:

- Существуют процессы, распараллеливание которых не дает прироста в общем случае: например, операции ввода-вывода.
- Существует оптимальное количество потоков для различных процессов после которого предельные издержки на дополнительный поток будут превышать прирост производительности.
- Параллельные вычисления при правильном применении могут значительно ускорить работу программы.

РАСПАРАЛЛЕЛИВАНИЕ МЕТОДА ГАУССА РЕШЕНИЯ СЛАУ

А. Алексеев

Последовательный алгоритм

Прямой ход:

- Работаем с расширенной матрицей: матрица + вектор правой части.
- Осуществляем перебор по столбцам матрицы.
- Производим выбор ведущего (максимального по абсолютной величине) элемента в текущем столбце. Если он не на диагонали, меняем местами соответствующие строки.
- Делим строку с ведущим элементом на значение этого элемента.
- Вычитаем из всех строк нашу строку, помноженную на значение каждой строки в текущем столбце.
- После прямого хода матрица приведена к верхнетреугольному виду с единицами на главной диагонали.

Прямой ход метода Гаусса: приведение расширенной матрицы системы

$$Ar = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}$$

к ступенчатому виду

$$\begin{pmatrix} 1 & c_{12} & \dots & c_{1n} & d_1 \\ 0 & 1 & \dots & c_{2n} & d_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & d_n \end{pmatrix}.$$

Обратный ход:

- Так как работали с расширенной матрицей, последнему значению решений соответствует нижнее значение вектора правой части.
- Поднимаемся постепенно вверх, новые значения вектора решения очевидно находятся из предыдущих значений.

Параллельный алгоритм

- Распараллеливание ведется с использованием директив Open MP.

- Реализация алгоритма написана на языке C++, для распараллеливания используем

```
#pragma omp parallel for shared (ex1,ex2) private (ex3)
```

специальная директива для цикла for

Участки для распараллеливания:

- при выборе ведущего элемента, когда меняем строки матрицы местами;
- при делении элементов строки на ведущий элемент;
- при вычитании строк друг из друга.

Весь параллелизм проявляется только в прямом ходе, во время обратного хода каждая операция напрямую зависит от предыдущей, поэтому распараллелить нельзя.

Результаты прогонов

Процессор	1	2	4	8	16
Intel Core 2 4400 (2 процессора)	469	312	390	578	843
Ускорение	1	1,503	1,202	0,811	0,556
Intel Core i5 650 (4 процессора)	281	203	203	374	499
Ускорение	1	1,384	1,384	0,751	0,563

Эксперименты проводились на компьютерах Intel Core 2 4400 и Intel Core i5 650, в среде разработки Microsoft Visual Studio 2010. Матрицы размера 200×200, 100 прогонов. Фрагменты программы приведены ниже.

```
void ParallelGauss(double A[N][N+1],double x[N]){
      int change[N];
      double a;
      omp_set_num_threads(Nproc);
            for (int j=0; j<N; j++){
            change[j]=j;
            double max=0;

            for (int i=j; i<N; i++){
                  if (fabs(A[i][j])>max) {max=fabs(A[i]
[j]); change[j]=i;}
            }
            if (change[j]!=j)
#pragma omp parallel for shared (A,j) private (a)
                  for (int k=j;k<N+1; k++) {a=A[j][k];
A[j][k]=A[change[j]][k]; A[change[j]][k]=a;}
if (A[j][j]!=0)
```

```
      {
            a=A[j][j];
            #pragma omp parallel for shared (A,j,a) //
private (k)
            for (int k=j;k<N+1;k++){
                  A[j][k]=A[j][k]/a;
            }
#pragma omp parallel for shared (A,j) private (a)
            for (int i=j+1; i<N; i++){
                  a=A[i][j];
                  for (int k=j; k<N+1;k++){
                  A[i][k]=A[i][k]-a*A[j][k];
                  }
            }
      }
      else printf("Ooops!!!Failed");

          }
      /*
      printf("Straight\n");
      for (int m=0;m<N;m++){
        for (int n=0;n<N+1;n++){
              printf("%.2f ",A[m][n]);
              }
        printf("\n");
      }
      */
      for (int i=N-1;i>=0;i--){
            x[i]=A[i][N];
            for (int j=i+1;j<N;j++){
                  x[i]=x[i]-A[i][j]*x[j];
            }
                  }
//    printf(«Back\n»);
//for (int i=0;i<N;i++){printf(«x%d= %f\n»,i,x[i]);}

 }
```

Литература

1. *Миллер Р., Боксер Л.* Последовательные и параллельные алгоритмы.

2. *Гергель В. П.* Курс «Теория и практика параллельных вычислений» http://www.intuit.ru/department/calculate/paralltp/

3. *Левин М. П.* Курс «Параллельное программирование с использованием OpenMP». http://www.intuit.ru/department/se/openmp/

О МЕТОДИКЕ ОБУЧЕНИЯ РАСПАРАЛЛЕЛИВАНИЮ БОЛЬШИХ ПОТОКОВ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ OPENMP (ПРОСТЫЕ ПАТТЕРНЫ)

И. Д. Мирошниченко

Санкт-Петербургский государственный университет, математико-механический факультет

Сегодня решение любых серьезных задач прогнозирования, эксперимента, проектирования и других направлений как в хорошо формализуемых, так и плохо формализуемых областях знаний, не представляется без анализа математической модели предметной области и многокритериального эксперимента над этой моделью и/или итерациями этой модели, качественно уточняющими модель исследуемой области знаний. Современные большие проекты порождают большие потоки разного типа входных, выходных, временных данных, а также инициируют процесс разработки новых и улучшения существующих методов эффективной обработки таких потоков.

Один из аспектов эффективности такой обработки и, пожалуй, наиболее важный в современных условиях — минимизация времени получения возможно большего количества результатов качественных исследований математической модели. С этой точки зрения активно используются методы распараллеливания в многопроцессорных вычислительных системах с использованием специальных языков параллельного программирования.

Современные многопроцессорные вычислительные системы, предназначенные для параллельной обработки больших потоков данных очень разнообразны композиционно по своей архитектурной реализации. Классически выделяют два основных типа архитектур:

- с единым логическим адресным пространством, разделяемым параллельными процессами (кластеры с набором программ библиотеки MPI, суперЭВМ и многоядерные компьютеры с технологией, поддерживающей параллельные языки программирования, например, OPENMP.) и
- с распределенными ресурсами (GRID-системы, к примеру, известный проект *LHC@Home* — моделирования отказоустойчивой работы Большого адронного коллайдера).

Для каждой архитектуры многопроцессорной системы существует своя специфика технологии эффективного распараллеливания алгоритмов и потоков данных, связанная также и с особенностями поддерживаемых параллельных языков программирования. Поэтому программы, эффективные для одной архитектуры, могут быть неэффективны для другой, методы распараллеливания, используемые для одной технологии и языка, могут оказаться неэффективными для другой технологии или языка.

Для кластерных параллельных вычислительных систем на сегодняшний момент существует довольно много работ, посвященных различным проблемам распараллеливания больших потоков данных (см. например, [4–8]), для технологии OPENMP их значительно меньше, в особенности, для плохо формализуемых задач.

Рассмотрим один из способов исследования и решения таких плохо формализуемых задач (распознавания образов, кластеризации, прогнозирования и т. п.), связанных с большими потоками информации, — конструирование нейронных сетей. В нейронных сетях поиск решения считается успешным при нахождении решения, оптимального по многим критериям даже при неполных или искаженных входных данных. Методика поиска решения состоит из двух частей: сначала нейронные сети обучаются на заданных примерах, т. е. конструируется требуемый алгоритм распознавания с заданной метрикой (подбираются коэффициенты связей между нейронами, выявляются зависимости между входными и выходными данными, добавляются промежуточные), а затем производится анализ произвольно взятого объекта. В обеих частях методики просматриваются возможности эффективного распараллеливания.

При обучении студентов методам параллельных вычислений представляется интересным показать разнообразные примеры, обучающие технике распараллеливания в технологии OPENMP на многоядерных компьютерах (начиная с двух ядерных), совершенствуемых Intel, широко внедряемых на мировом рынке и более или менее доступных для персонального использования (по крайней мере, на базе платформы Intel Core Duo, иногда Quadro), т. к. такой подход представляет реальную возможность не зависеть от особенностей технической реализации конкретной информационной системы и ее программного обеспечения, в частности, в виду различной степени защиты информации при обеспечении безопасности. Это существенно, в большой степени, если обучение проводится с группами обучающихся в классах, базирующихся на однотипных аппаратно-операционных платформах, но предполагающих различную степень делегирования пользователям некоторых полномочий администрирования необходимого для установки соответствующего программного обеспечения, или, прежде всего, оборудованных одноядерными компьютерами.

Рассмотрим применение методики обучения распараллеливанию данных на базе двух ядерных ноутбуках (так приходится работать со студентами гуманитарных факультетов, т. к., как правило, классы у них оборудованы одноядерными компьютерами).

Обоснование выбора подхода

Для выбора методики распараллеливанию важно проанализировать вычислительную систему с различных точек зрения: архитектуры, топологии, подхода.

В случае работы с двух ядерными ноутбуками архитектура и топология вычислительной системы в достаточной степени определены.

В общем случае подходов к программированию параллельных вычислительных систем существует много. Их можно разбить на 4 такие группы в зависимости от используемых средств программирования:

- **специализированных языков параллельного** программирования и **параллельных *расширений* последовательных языков** (параллельные реализации Fortran и C и пр.);
- средств **автоматического и полуавтоматического *распараллеливания*** последовательных программ (BERT 77, FORGE и др.);
- **последовательных языков** программирования с использованием **коммуникационных библиотек и интерфейсов** для организации **межпроцессорного взаимодействия** (MPI, PVM, **OpenMP** *и* др.);
- **последовательных языков** с использованием **параллельных библиотечных процедур** (ScaLAPACK, HP Mathematical Library, PETSc и др.).

Нас будет интересовать возможность установки таких средств распараллеливания, которые позволяли бы использовать возможность работы с технологией OpenMP.

Такой подход позволяет создавать параллельные конструкции, не зависящие от конкретной архитектуры или типа параллельных вычислительных систем, что, принципиально, повышает эффективность и надежность написанных на этом **языке** программ. Платить за это придется определенной трудностью освоения и сложностью создания эффективных объектных кодов при **трансляции**.

Понятно, что прежде, чем обучать методике распараллеливания, потребуется время на установку необходимых средств работы с параллельной оболочкой: используем активизацию технологии OPENMP в оболочке VISUAL C/C++, а также средства анализа производительности программ VTune Performance Analyzer и Intel Thread Checker, позволяющие находить критические секции в многопоточных программах и существенно ускорять их выполнение.

Кроме того, компанией Intel разработаны библиотеки стандартных функций Intel Math Kernel Library, Intel Cluster Math Kernel Library и Intel Integrated Performance Primitives, позволяющие существенно ускорить работу параллельных программ по обработке как числовой, так и графической информации, а также продукт Intel Cluster OpenMP, позволяющий распространить технологию OpenMP на кластерные системы с *распределенной* памятью.

Плюсы технология OpenMP в том, что она предоставляет пользователю

— **способ создания потоков** в приложениях, не требуя от программиста знаний о создании, синхронизации и уничтожении потоков, а также необходимости определения, сколько потоков следует создать.
— **независимый от платформы** набор прагм, директив, вызовов функций и переменных среды, которые явным образом указывают компилятору, как и где именно следует вставить потоки в приложение.

Перейдем теперь к существу методикии. Представим простейшие примеры (простейшие паттерны), тем не менее существенно влияющие на результат распараллеливания.

Набор примеров распараллеливания больших потоков данных включает как ставшие уже стандартными примеры распараллеливания операций с векторами и матрицами (хорошо формализуемые ИС), так и примеры, связанные с параллельными вычислениями в областях интеллектуальной обработки информации (плохо формализуемые ИС).

Однако, прежде чем анализировать сложные конструкции, требуемые для обработки больших объемов данных, рассмотрим простые паттерны, а именно, конструкцию — распараллеливание цикла. Ввиду того, что технология OPENMP применяется к последовательному языку программирования, конструкция цикла является важнейшей конструкцией распараллеливания, применяемой к блокам независимых данных. (Большие массивы независимых данных, чаще всего, можно попытаться представить некоторыми наборами этих данных, независимыми и примерно равными по времени исполнения.)

Как известно, большинство циклов можно распараллелить, вставив всего одну прагму непосредственно перед циклом. Это справедливо для независимых данных.

В этом случае, оставив исполнение рутинных функций компилятору и OpenMP, можно больше времени уделить определению “горячих точек”, т. е. наиболее трудоемких циклов в приложении для того, чтобы определить,

— какие циклы следует распараллелить, и
— как наилучшим образом изменить структуру алгоритмов

для достижения максимальной производительности.

Демонстрируем высказанное положение известным простым примером.

Пример 1. Преобразование 32-разрядного пикселя цветовой модели RGB в 8-разрядный пиксель модели Grayscale. Требуется для распараллеливания одна прагма, вставленная непосредственно перед циклом:

```
1. #pragma omp parallel for
2. for (i=0; i < numPixels; i++)
3. {
4.  pGrayScaleBitmap[i] = (unsigned BYTE)
5.      (pRGBBitmap[i].red * 0.299 +
6.      pRGBBitmap[i].green * 0.587 +
7.      pRGBBitmap[i].blue * 0.114);
8. }
```

В этом примере используется «**распределение нагрузки**» между потоками директивой **for OPENMP**. При таком распределении нагрузки, итерации цикла распределяются между несколькими потоками так, что каждая итера-

ция цикла выполняется только один раз, параллельно одним или несколькими потоками. OpenMP автоматически определяет:

— сколько потоков следует создать, а также
— наилучший способ создания, синхронизации и уничтожения потоков.

Программисту необходимо лишь указать OpenMP, какой именно цикл следует распараллелить.

В OpenMP существуют пять ограничений на то, какие циклы можно распараллелить:

- переменная цикла должна иметь тип **signed integer**, беззнаковые целые числа работать не будут;
- операция сравнения должна иметь следующий формат: **переменная_цикла <, <=, >, >= инвариант_цикла_целого_типа**;
- третье выражение (или инкрементная часть цикла *for*) должно являться либо **целочисленным сложением**, либо **целочисленным вычитанием** и должно практически совпадать со значением инварианта цикла;
- если используется операция сравнения < или <=, переменная цикла должна увеличиваться при каждой итерации, а при использовании операции > или >= переменная цикла должна уменьшаться;
- цикл должен являться **базовым блоком** (не разрешены переходы из цикла, за исключением оператора *exit*, который завершает работу всего приложения. Если используются операторы *goto* или *break*, они должны приводить к переходам внутри цикла, а не вне его. То же самое относится к обработке исключений; исключения должны перехватываться внутри цикла).

Зависимости данных и состояния гонки

А теперь обратимся к особенностям распараллеливания данных

Если цикл соответствует всем ограничениям и компилятор распараллелил цикл, это отнюдь не гарантирует правильной работы, ввиду того, что может существовать **зависимость данных.** Зависимость данных существует, если различные итерации цикла (точнее говоря, итерация, которая выполняется в другом потоке) *выполняют* **чтение или запись общей** *памяти*.

Рассмотрим стандартный пример цикла, в теле которого есть зависимости от предыдущих итераций.

Пример 2. Вычисление факториалов.

```
1.  #pragma omp parallel for
2.  for (i=2; i < 10; i++)
3.  {
4.    factorial[i] = i * factorial[i-1]; //зависимость
от данных другой итерации
5.  }
```

Компилятор создает из этого цикла поток, но программа завершается ошибкой, т. к., по крайней мере, **одна из итераций цикла зависит от данных другой итерации**. Имеем состояние гонки или гонки данных. (Напомним, что состояние гонки возникает только при использовании **общих ресурсов — например, памяти — и при параллельном выполнении.)** Для решения этой проблемы следует либо изменить цикл, либо выбрать другой алгоритм, который не приведет к состязанию потоков.

Состояние гонки трудно обнаружить, поскольку, в данном примере, переменные могут «выигрывать гонку» в том порядке, который обеспечивает правильную работу программы. Но то, что программу удалось выполнить один раз правильно, не означает, что она будет работать правильно **всегда.** Полезно тестирование программы в различных системах, например, в технологии Hyper-Threading, или с помощью инструмента Intel Thread Checker. Традиционные программы отладки не в состоянии обнаружить гонки данных, поскольку вынуждают один поток **остановить** «гонку» в то время, как остальные потоки продолжают вносить значительные изменения в динамическое поведение.

Общие и локальные данные

Но ведь в каждом невырожденном (полезном) цикле выполняется **чтение данных из памяти и запись в память.** Значит, в программе есть общие для всех потоков участки памяти и локальные, и программист должен **явно** указать компилятору тип области памяти для переменных. Если память определена как **общая**, все потоки получают доступ к одному и тому же адресу памяти. Если же память определена как **локальная**, создается отдельная копия переменной для каждого потока, чтобы обеспечить локальный доступ. После завершения цикла эти локальные копии уничтожаются. По умолчанию, **общими** являются все переменные, за исключением переменной цикла, которая является локальной.

Память можно объявить как локальную следующими двумя способами:

- объявить переменную внутри цикла (внутри директивы OpenMP) без ключевого слова static;
- указать выражение локальным в директиве OpenMP.

Пример 3. В этом примере программа работает неправильно, т. к. переменная *temp* указана **общей**, а должна быть локальной ввиду зависимости данных в теле цикла.

```
1.  #pragma omp parallel for
2.  for (i=0; i < 100; i++)
3.  {
4.    temp = array[i]; //переменная temp по умолчанию
определена как общая
```

```
5.   array[i] = do_something(temp);
6.  }
```

Решить эту проблему можно, объявив *temp* локальной, например, способами, демонстрируемыми в **примерах 4** и **5**.

Пример 4.

```
1.  #pragma omp parallel for
2.  for (i=0; i < 100; i++)
3.  {
4.     int temp; // temp  -  локальная
5.     temp = array[i];
6.     array[i] = do_something(temp);
7.  }
```

Пример 5.

```
1.  #pragma omp parallel for private(temp)
2.  // temp  -  локальная
3.  for (i=0; i < 100; i++)
4.  {
5.     temp = array[i];
6.     array[i] = do_something(temp);
7.  }
```

Следует учитывать, что переменные, объявленные в директиве **parallel**, определяются как локальные, за исключением того случая, когда они объявляются с описателем **static**, т. к. статические переменные не размещаются в стек.

Оптимизация суммирования элементов массива.

В OpenMP имеется специальная конструкция для выполнения суммирования элементов массива, аналогичная в записи соответствующей конструкции последовательного языка.

Пример 6.

```
1.  sum = 0;
2.     for (i=0; i < 100; i++)
3.     {
4.          sum += array[i];    // переменная sum здесь
объявлена  общей  по умолчанию
5.      }
```

Однако для обеспечения доступа нескольких потоков переменная *sum* должна быть локальной, (кроме того, чтобы быть общей) для получения правильного результата параллельных вычислений.

Для разрешения этой ситуации в OpenMP предлагается использовать выражение **reduction**, которое применяется для эффективного сочетания математического уменьшения одной или нескольких переменных в цикле.

Пример 7.

```
1.        sum = 0; //переменная sum – по умолчанию
объявлена общей
2.        #pragma omp parallel for reduction(+:sum) //
для потоков создаются локальные копии переменной sum
3.        for (i=0; i < 100; i++)
4.        {
5.         sum += array[i];
6.        }
```

Для каждого потока *sum* — локальна, т. е. OpenMP предоставляет локальные копии переменной *sum* для каждого потока, а после завершения потоков складывает значения и помещает результат в одну глобальную копию переменной. В OpenMP допускаются использование следующих операций, действующих и определяемых подобно описанной для переменной *sum*: +, - , *, &, |, ^, &&.

В цикле можно выполнить несколько таких операций, указав переменные, разделенные запятыми, и соответствующие им операции в данной директиве *parallel*. Важно соблюдение следующих требований:

1. выше перечисленные переменные можно перечислить только в одном *reduction*;
2. переменные нельзя объявить константами;
3. переменные нельзя объявить локальными в директиве *parallel*.

Планирование циклов

Задача распределения рабочей нагрузки поровну между потоками — определения **баланса нагрузки** — одна из наиболее важных **составляющих** параллельного выполнения программы. Баланс нагрузки гарантирует **одновременную работу всех процессоров в максимальном интервале времени** (т. е. без баланса нагрузки некоторые потоки могут завершить работу значительно раньше остальных, что приводит к простою вычислительных ресурсов и потере производительности).

Если времена вычислений в различных итерациях цикла различаются, — это приведет к отсутствию баланса нагрузки. Поэтому важно предварительно

проанализировать исходный код в циклах. В большинстве случаев итерации цикла занимают одинаковое время. Однако, сли это не так, то можно попытаться подобрать наборы итераций, занимающие одинаковое время. Например, возможно,

— набор всех четных итераций может занимать примерно столько же времени, как и набор всех нечетных итераций;
— первая половина итераций цикла может занимать примерно столько же времени, как и вторая половина;
— но иногда может оказаться невозможным найти наборы итераций, имеющие одинаковое время выполнения.

Независимо от того, какой из этих вариантов имеет место в конкретном случае, необходимо **предоставить** OpenMP эту дополнительную информацию о планировании цикла, чтобы он мог правильно распределить итерации цикла между потоками (и, следовательно, между процессорами) для оптимизации распределения нагрузки.

По умолчанию, OpenMP предполагает, что все итерации цикла занимают одинаковое время. OpenMP распределяет итерации цикла между потоками примерно поровну и таким образом, чтобы минимизировать вероятность возникновения конфликтов памяти вследствие ее неправильного совместного использования. Это возможно, поскольку итерации цикла обычно обращаются к памяти последовательно. Поэтому при разделении цикла на две большие части (например, на первую и вторую половины) при использовании двух потоков вероятность наложения памяти оказывается наименьшей.

Однако, хотя это и может быть наилучшим вариантом для избежания конфликтов памяти, с точки зрения баланса нагрузки это может быть плохим выбором. Обратное — тоже справедливо: то, что хорошо для баланса нагрузки, может быть плохо для работы с памятью. Поэтому программисту обязательно нужно проводить **сравнительный анализ вариантов программы**, чтобы определить баланс между **оптимальным использованием памяти *и* оптимальным распределением нагрузки**, измеряя **производительность**, чтобы определить, **какие методы дают наилучшие результаты.**

Информация о планировании цикла передается в OpenMP с помощью директивы **parallel**, имеющей следующий формат.

```
#pragma omp parallel for schedule(kind [, chunk
size])
```

Различают четыре различных типа планирования цикла в OpenMP. Параметр (**chunk**) должен являться постоянным для цикла положительным целым числом.

Тип	Описание
Статический	Делит цикл на блоки — равные или максимально приближенные к равным, если количество повторов цикла не может быть поровну разделено на количество потоков, умноженное на размер блока. По умолчанию размер блока равняется количеству циклов; разделенному на количество потоков, Для реализации чередования повторов в качестве значения блока следует задать 1.
Динамический	Использует внутреннюю рабочую очередь для передачи группы повторов цикла (по размеру блока) на каждый поток. При завершении потока следующая группа повторов цикла передается из начала рабочей очереди. По умолчанию размер блока = 1. При использовании данного метода распределения требуются дополнительные ресурсы.
Управляемый	Аналогичен динамическому распределению, однако размер блока постепенно уменьшается, что позволяет снизить продолжительность времени, требуемого для перехода потоков к рабочей очереди для получения дополнительной нагрузки. Дополнительный параметр chunk указывает минимально допустимый размер блока, По умолчанию размер блока равен количеству циклов, разделенному на количество потоков,
Время выполнения	Использует переменную OVIP_SCHEDULE для указания одного из трех типов распределения. OMP_SCHEDULE — строковая переменная, форматированная аналогично строкам, используемым в конструкции parallel

Пример 8. Распараллелить следующий цикл

```
1. for (i=0; i < NumElements; i++)
2.      {
3.         array[i] = StartVal;
4.         StartVal++;
5.      }
```

Исходный код цикла содержит **зависимость данных**, что делает невозможным его распараллеливание без **внесения изменений**.

Цикл, показанный ниже, заполняет массив таким же образом, но в нем уже отсутствуют зависимости данных.

```
1. #pragma omp parallel for
2.      for (i=0; i < NumElements; i++)
3.      {
4.         array[i] = StartVal + i; //зависимость отсут-
ствует
5.      }
```

Полученный код не на 100% совпадает с первоначальным кодом, т. к. отсутствует приращение переменной **StartVal**. В результате, после завершения параллельного цикла, эта переменная будет иметь значение, отличное от того, которое получается в последовательной версии. Если значение **StartVal** используется после цикла, требуется дополнительный оператор, как показано ниже:

```
1. #pragma omp parallel for //Этот фрагмент идентичен
последовательной версии
2. for (i=0; i < NumElements; i++)
3. {
4. array[i] = StartVal + i;
5. }
6. StartVal += NumElements;
```

Таким образом, учитывая эти простые примеры, можем заклбючить, что для больших объемов данных весьма желательно провести сравнительную характеристику распараллеливания потоков данных с последующим анализом их эффективности для алгоритмов, как выполняемых разными методиками интеллектуальной обработки (например, задача коммивояжера в нейронных сетях или генетических алгоритмах), так и выполняемых в разных технологиях: OPENMP и MPI (там, где есть возможность проверить работу в разных технологиях).

В частности, при знакомстве с темой «Нейронные сети» наглядный пример улучшения оригинального алгоритма Хопфилда с точки зрения быстродействия дает распараллеливание по данным (с учетом зависимости и независимости и разделения на соответствующие наборы), т. к. процесс обучения сети можно распараллелить, учитывая количество примеров для обучения и количество используемых ядер процессора.

Литература

1. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2004. 602 с.

2. *Немнюгин С., Стесик О.* Параллельное программирование для многопроцессорных вычислительных систем. СПб.: БХВ-Петербург, 2002. 200 с.

3. *Корнеев В. В., Гареев А. Ф., Васютин С. В., Райх С. В.* Базы данных. Интеллектуальная обработка информации. М.: Нолидж, 2001. 352 с.

4. *Федорова Н. Н., Терехов С. А.* Параллельная реализация алгоритмов обучения нейронных сетей прямого распространения с использованием стандарта MPI. Российский Федеральный Ядерный Центр — Всероссийский НИИ Технической Физики. E-mail: sta@nine.ch70.chel.su

5. *Стюарт Рассел, Питер Норвиг.* Искусственный интеллект: современный подход (Artificial Intelligence: A Modern Approach). 2-е изд. М.: Вильямс, 2006. 1424 с.

06. *Тархов Д. А.* Нейронные сети. Модели и алгоритмы. М.: Радиотехника, 2005. 256 с.

7. *Комарцова Л. Г., Максимов А. В.* Нейрокомпьютеры. 2-е изд. М.: МГТУ им. Баумана, 2004. 400 с.

8. http://www.neuropro.ru/

9. http://www.intuit.ru/catalog/hpct (Каталог курсов «Суперкомпьютерные технологии и высокопроизводительные вычисления»).

10. http://www.intuit.ru , в частности: 1) *М. В. Якобовский.* Введение в параллельные алгоритмы. 2) *В. П. Гергель.* Введение в методы параллельного программирования. 3) *В. П. Гергель.* Теория и практика параллельных вычислений. 4) *А. Б. Барский.* Архитектура параллельных вычислительных систем. 5) *С. А. Немнюгин.* Модели и средства программирования для многопроцессорных вычислительных систем, и многие другие курсы этого каталога.

.

ПРОГРАММИРОВАНИЕ И ОПТИМИЗАЦИЯ МЕТОДА РЕЛАКСАЦИИ РЕШЕНИЯ СЛАУ

М. П. Винник

Научный руководитель:
И. Г. Бурова

Санкт-Петербургский Государственный Университет

Решается и оптимизируется задача нахождения приближенных решений СЛАУ методом релаксации. В отличие от точных методов решения СЛАУ метод не требует громоздких вычислений и изначально предназначался для вычислений вручную.

Изложение метода

Система линейных уравнений (1) приводится к виду (2):

$$\begin{cases} a_{11}\cdot x_1 + ... + a_{1n}\cdot x_n = b_1, \\ a_{21}\cdot x_1 + ... + a_{2n}\cdot x_n = b_2, \\ ... \\ a_{n1}\cdot x_1 + ... + a_{nn}\cdot x_n = b_n, \end{cases} \quad (1)$$

$$\begin{cases} -x_1 + b_{12}\cdot x_2 + ... + b_{1n}\cdot x_n + c_1 = 0, \\ b_{21}\cdot x_1 - x_2 + ... + b_{2n}\cdot x_n + c_2 = 0, \\ ... \\ b_{n1}\cdot x_1 + b_{n2}\cdot x_2 + ... - x_n + c_n = 0, \end{cases} \quad (2)$$

где $b_{ij} = -\dfrac{a_{ij}}{a_{ii}}$ $(i \neq j)$ и $c_i = \dfrac{b_i}{a_{ii}}$.

Пусть $x^{(0)} = (x_1^{(0)} ... x_n^{(0)})$ — это начальное приближение решения системы. Далее находятся невязки:

$$\begin{cases} R_1^{(0)} = c_1 - x_1^{(0)} + \sum\limits_{j=2}^{n} b_{1j}\cdot x_j^{(0)}, \\ R_2^{(0)} = c_2 - x_2^{(0)} + \sum\limits_{j=1,\, j\neq 2}^{n} b_{2j}\cdot x_j^{(0)}, \\ ... \\ R_n^{(0)} = c_n - x_n^{(0)} + \sum\limits_{j=1}^{n-1} b_{nj}\cdot x_j^{(0)}. \end{cases}$$

Наша цель на каждом шаге — обратить в ноль максимальную по модулю невязку. Чтобы это осуществить величине $x_s^{(0)}$ дадим приращение $\delta x_s^{(0)} = R_s^{(0)}$ и мы будем иметь $R_s^{(1)} = 0$ и $R_s^{(1)} = R_i^{(0)} + b_{is} \cdot \delta x_s^{(0)}$ при $i \neq s$. Процесс закончится когда все невязки будут равны нулю с заданной точностью. После этого получаем приближенное решение $x = (x_1, ..., x_n)$ по формуле: $x_i = x_i^{(0)} + \sum_k \delta x_i^{(k)}$. Метод сходится к решению если матрица коэффициентов системы положительно определена и является матрицей с диагональным преобладанием.

Описание программы

Программа представляет собой консольное приложение написанное на языке C++ с применением **Open MP.** Пользователь задает размер матрицы, имя файла, допустимую погрешность и вектор начального приближения(одним числом или вручную вводит все компоненты). Матрица вместе с присоединенным к ней справа вектором констант должна быть записана в текстовый файл специальным образом, а именно «в одну строку», по порядку элементов: «$a_{1,1}$, $a_{1,2}$, ..., $a_{1,n}$, $a_{2,1}$» и т. д. (обязательно должно быть по одному пробелу между элементами). Матрицы реализованы с помощью двумерных динамических массивов. Open MP не позволяет распараллелить цикл с неизвестным числом итераций. Поэтому перед началом подсчетов для построения приблизительного количества итераций используются эмпирические оценки числа итераций.

Оценка числа итераций

Для получения оценок были проведены тесты с помощью последовательной программы (далее она же использовалась для подсчета ускорения) на матрицах размеров 100×100, 150×150, 200×200, …, 400×400 при нулевом векторе начального приближения. Было подсчитано количество итераций для каждой из этих матриц для точностей от 0.01 до 0.0000001. Матрицы меньших размеров не тестировались из-за нецелесообразности применения на них распараллеливания (разделение вычислений на потоки очень ресурсоемкая команда). Матрицы генерировались случайным образом в математическом пакете Maple 11. Для обеспечения существования решения матрица системы выбиралась с диагональным преобладанием, при этом диапазон случайных чисел брался на диагонали от $n \cdot 5$ до $n \cdot 5 + 100$, а вне диагонали от 1 до 5. Компоненты вектора правой части генерировались в диапазоне от –200 до 200. Записать полученную матрицу в файл нетрудно: достаточно после ее генерирования в Maple вставить команду записи матрицы в текстовый файл

```
> f1:=open(«C:\\mapdata.txt»,WRITE);
for i from 1 to n do
 for j from 1 to n+1 do
  fprintf(f1,»%d «,C[i,j],» «);
   end do;
end do;
close(f1);
```

Матрица будет записана в файл в нужной форме, останется только удалить квадратные скобки с помощью команды «**Заменить на...**» и заменить запятые между элементами на пробелы.

Распараллеливание

На данный момент проведено распараллеливание только цикла **for** основного хода метода. Далее предполагается включить распараллеливание работы с матрицами (приведение к нужному виду). Программа находится в завершающей стадии, оптимизация еще не полна, но тем не менее результат, связанный с ускорением, уже получен м представлен в таблице. Сейчас при одинаковом количестве итераций для последовательного и параллельного варианта во втором случае получается решение с значительно большей точностью за примерно одно и то же время работы. В таблице приведены результаты предварительных экспериментов, проводившихся при погрешности 0.0001 и нулевом векторе начального приближения:

Размер матрицы	Время обычной программы	Время программы с распараллеливанием	Число итераций для последовательной программы	Число итераций для программы с распараллеливанием
120×120	87	88	236	280
170×170	179	176	351	380
230×230	329	328	466	530
270×270	471	432	501	590
310×310	576	570	534	700

Итог

Даже с грубыми оценками и с неполной оптимизацией, программа дает существенное ускорение, что еще раз подчеркивает всю мощь **Open MP** как средства оптимизации и языка **C++,** как средства реализации объемных математических расчетов

Литература

1. *Демидович Б. П., Марон И. А.* Основы вычислительной математики. М., 1960.

РАСПАРАЛЛЕЛИВАНИЕ ЗАДАЧИ ИЗ ТЕОРИИ ЧИСЕЛ

М. А. Тверьянович, А. Р. Ханов

Научный руководитель:
И. Г. Бурова

Для того чтобы распараллелить вычислительную задачу, написанную на языке C или C++, на одном многопроцессорном компьютере можно использовать технологию OpenMP. Она позволяет сделать это с минимальными трудозатратами. Все, что нужно сделать: это расставить директивы распараллеливания в наиболее трудоемких для вычисления местах. Возникает вопрос о том, каково будет изменение производительности при различных способах расстановки директив.

Рассмотрим такую задачу: необходимо провести операцию побитового «исключающего или» для всех девятизначных чисел. В обычном последовательном варианте это будет простая программа из девяти вложенных циклов «for» с одной операцией внутри — накопление результата. Для распараллеливания необходимо не только указать OpenMP на то, что нужно разбить цикл, но и то, что доступ к переменной, в которой производится накопление результата, будет последователен для всех потоков исполнения. В целом вся директива выглядит так:

(1) `#pragma omp parallel for reduction(^:Result)`

Попробуем ставить ее последовательно перед каждым циклом и проследить изменение скорости работы. Результат показан на графике 1.

В каждом случае время работы распараллеленной программы меньше времени работы последовательной, кроме случая, когда распараллеливаются наиболее вложенные циклы (начиная с 7 вложенного цикла). В этом случае эффект от этой директивы получается обратным. Однако кроме этого наиболее хороший результат по времени получается при использовании директивы (1) перед одним из первых трех циклов и время работы во всех этих трех случаях практически не отличается.

Теперь попробуем провести распараллеливание по-другому. Ключевое слово reduction в директиве (1) делает следующее: для каждого потока исполнения заводится свой экземпляр переменной, внутри потока проводится операция, после чего указанная в директиве операция проводится между всеми экземплярами переменных из потоков исполнения.

Попробуем сделать то же самое, но разделив вручную один из вложенных циклов на два независимых цикла с половиной от общего числа итераций этого цикла с помощью директивы (2).

(2) `#pragma omp parallel sections`

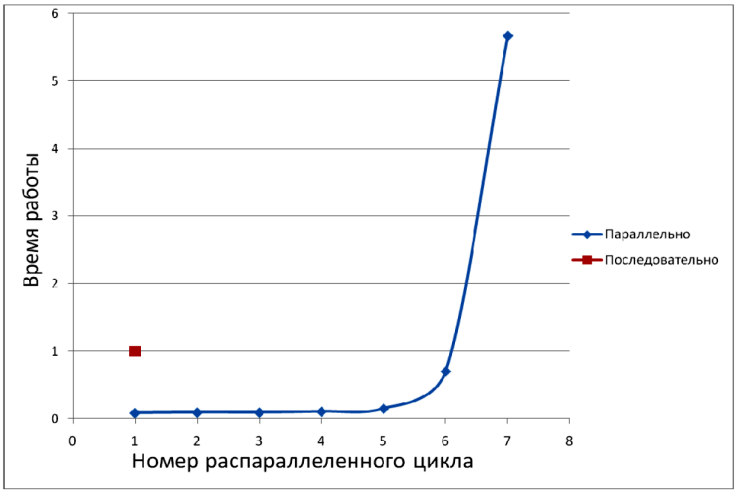


График 1. Зависимость времени, затраченного на вычисление задачи, от номера цикла перед которым поставлена директива (1) с “parallel for”

Каждая из частей разбитого на пополам цикл включается в отдельный блок section и в каждой из этих двух частей идет суммирование результата в отдельную переменную, после выполнения обоих секций их отдельные результаты сводятся в результирующую переменную. Результат в сравнении с предыдущим способом показан на графике 2.

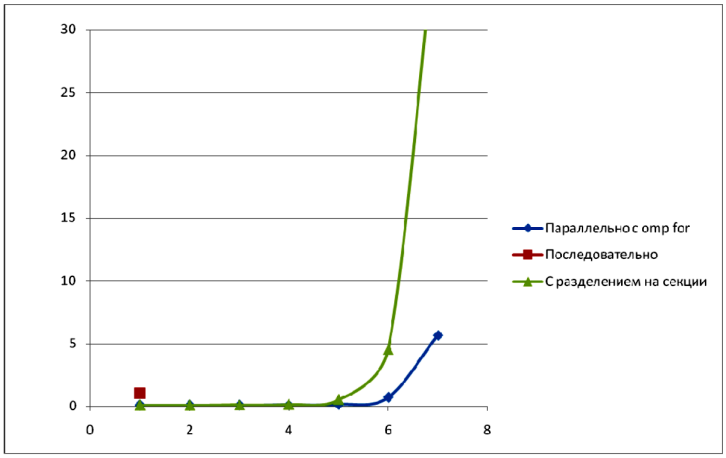


График 2. Зависимость времени, затраченного на вычисление задачи, от номера цикла перед которым поставлена директива (1) с “parallel for” или проведено разбиение на секции с помощью директивы (2)

По графику 2 видно, что при распараллеливании с помощью разбиения на секции на внешних циклах ускорение практически не отличается от того, которое получается при использовании директивы (1) с parallel for. Однако с распараллеливанием внутренних циклов заметно лучше справляется директива (1).

Из показанного выше следует следующий вывод: при наличии задачи с большим числом вложенных циклов имеет смысл использование директив только перед внешними циклами. Кроме того работу по разбиению цикла лучше всего доверить самому OpenMP посредством директивы parallel for, ведь улучшения при использовании разбиения на секции не наблюдалось.

Вычислительная геометрия

Вяткина
Кира Вадимовна

к.ф.-м.н.
доцент кафедры системного программирования СПбГУ

ЭВРИСТИЧЕСКИЙ АЛГОРИТМ *S** ДЛЯ ЗАДАЧИ ШТЕЙНЕРА НА ОРИЕНТИРОВАННЫХ ЕВКЛИДОВЫХ ГРАФАХ

Д. А. Ейбоженко

Санкт-Петербургский государственный университет

Задача Штейнера широко используется в проектировании интегральных схем, телекоммуникациях, биологии и других областях науки и техники. В максимально общей постановке, на ориентированных графах, она формулируется следующим образом:

Рассмотрим ориентированный граф $G(M, N)$ с заданной на дугах функцией $d: N \to \mathbb{R}_+$, начальной вершиной b и множеством терминальных вершин $E \subset M$. Любое ориентированное дерево с корнем в b, связывающее все терминалы, называется *деревом Штейнера*. *Задача Штейнера* состоит в поиске дерева Штейнера наименьшей длины.

Как известно из [1], задача Штейнера на графах является NP-сложной. Существует точный метод, основанный на принципе динамического программирования и уравнении Беллмана, представленный в [2]. В нем решение строится одновременно для всех вершин и для всех подмножеств множества терминалов. Процесс всегда находится в некотором состоянии (m, E_p), где $m \in M$, $E_p \in 2^E$. На каждом шаге выбирается одно из двух решений:

- Перейти по какой-либо дуге, начинающейся в m, в другую вершину;
- Разветвить путь, выбрав некоторое разбиение терминального множества $E_P = E_{P_1} \cup E_{P_1}$, и для каждого состояния $(m,\ E_{P_i})$ решить такую же задачу.

В то же время, многие задачи из реальной практики могут быть сформулированы с использованием евклидового графа, у которого для каждой вершины $m \in M$ заданы евклидовы координаты (x_m, y_m), а длины дуг определяются евклидовым расстоянием между их концами.

Общая идея алгоритма состоит в том, чтобы уменьшить количество рассматриваемых методом динамического программирования подмножеств терминальных вершин, исходя из данных о взаимном расположении вершин. Рассматриваются три разных способа построения множества подмножеств терминальных вершин.

- Простейший метод, основанный на упорядочивании терминалов по полярному углу относительно начальной вершины.
- Метод «концентрических окружностей», учитывающий также удаленность терминалов от начальной вершины.
- Обобщенный метод, рассматривающий в качестве базовой вершины для упорядочивания любую вершину из M.

Производится сравнение их точности и времени вычисления между собой, а также с другими известными методами решения задачи Штейнера

в общем виде — алгоритмом Такахаши [3], жадным эвристическим алгоритмом [4] и др.

Доказана полиномиальная сложность алгоритма при всех трех методах построения множества терминальных подмножеств, которая составляет, соответственно, $O(e^2 n \log m)$ для первых двух методов и $O(e^2 nm \log m)$ для обобщенного метода.

Литература

1. *М. Гэри, Д. Джонсон.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

2. *И. Романовский.* Дискретный анализ. СПб.: Невский Диалект; БХВ-Петербург, 2003.

3. *H. Takahashi* and *A. Matsuyama.* An approximate solution for the Steiner problem in graphs, Math. Japonica, 1980.

4. *A. Zelikovsky.* The last achievements in Steiner tree approximations, tempo.math. md, vol. 1, 1993, pp. 35–44.

ВЫЧИСЛЕНИЕ РЕДАКЦИОННОГО РАССТОЯНИЯ МЕЖДУ ДЕРЕВЬЯМИ НА ОСНОВЕ СТЯГИВАНИЯ ВЕРШИН

М. Е. Гладких

Санкт-Петербургский государственный университет

Для решения задач распознавания образов и компьютерного зрения полезно иметь легко вычислимую метрику для контуров, согласно которой похожими являются те контуры, которые близки с точки зрения человеческого восприятия. Это позволяет структурировать и выполнять запросы к базам данных изображений. Вот пример подобной задачи: дано изображение здоровой кости и кости пациента; требуется определить, насколько они различаются.

Подходящая метрика может быть определена на основе скелетов контуров (см., например, [2, 3]), которые, в свою очередь, могут быть интерпретированы в терминах филогенетических деревьев. В данном контексте имеются в виду филогенетические деревья без выделенного корня с помеченными листьями и непомеченными внутренними узлами. Без ограничения общности будем полагать, что в качестве меток используются целые числа от 1 до n. Два таких дерева с одинаковым числом вершин и листьев называются изоморфными, если существует биекция между их вершинами, сохраняющая метки.

Расстояние между филогенетическими деревьями (T_1, T_2) определяется как минимальное суммарное количество допустимых операций, применением которых к деревьям T_1 и T_2 можно получить из каждого из них одно и то же дерево T*. При задании классической метрики допустимой операцией модификации является стягивание ребра (см., например, [4]). Мы же предлагаем расширить множество допустимых операций, включив в него также операцию стягивания вершин, соединенных путем длины 2.

Для удобства можно ввести в рассмотрение операции, обратные к допустимым. Нами было показано, что любая последовательность операций, модифицирующих дерево T, может быть преобразована в последовательность операций той же длины, приводящей к тому же результату, в рамках которой вначале выполняются все допустимые операции, а затем — все операции, обратные к допустимым. Этим обусловлена, в частности, корректность приведенного выше определения расстояния между деревьями, а также существование переборного алгоритма для вычисления последнего.

Разработанный нами метод представляет собой обход с отсечениями графа состояний в ширину[5] с применением техники meet-in-the-middle [1]. Алгоритм работает следующим образом: поддерживаются очереди деревьев достижимых, соответственно, из T_1 и T_2, а также хеш-таблицы, в которых хранятся расстояния от T_1 и T_2 до этих деревьев. На каждом шаге алгоритма из очереди выбирается следующее дерево, достижимое из T_1 (на нечетном шаге) или T_2 (на четном шаге), и к нему применяется некоторое множество

допустимых операций (которое формируется на основе специальных правил). Если результирующее дерево ранее было получено из T_2, данный факт фиксируется; в противном случае, мы проверяем, было ли такое дерево уже получено из T_1 другим способом, и если ответ отрицательный, оно добавляется в конец очереди. Алгоритм завершается, когда очередь оказывается пуста.

При вычислении хеш-значений для деревьев использовалось полиномиальное хеширование матриц расстояний между листьями по модулю максимального значения знакового 64-битного типа. При этом не возникало никаких коллизий во время стресс-тестирования, что объясняется хорошими перемалывающими свойствами полиномиального хеширования и значительным запасом хеш-значений по сравнению с количеством рассматриваемых матриц.

Описанное выше переборное решение и несколько других решений, а также предложенный нами алгоритм генерации случайного дерева с заданным количеством вершин и листьев, алгоритм стресс-тестирования и несложный алгоритм визуализации были реализованы на языке Java.

Алгоритм стресс-тестирования использовался для анализа гипотез о преобразованиях деревьев. Его суть заключается в последовательной генерации очередной пары случайных деревьев и вычислении точного расстояния между ними, с последующим вычислением расстояния исследуемым методом и проверке полученных значений на равенство. Концептуально это эквивалентно продолжительному поиску контрпримера. Если он не найден за длительное время, то с большой вероятностью гипотеза верна.

Алгоритм генерации случайного дерева с заданным количеством внутренних вершин и листьев работает следующим образом. Вначале рассматривается дерево, состоящее из одной вершины. Далее на каждой итерации к случайной вершине текущего дерева присоединяется цепь случайной длины (взятой из некоторого распределения).

С помощью описанного выше метода был экспериментально подтвержден ряд результатов, предварительно полученных теоретическими методами.

Литература

1. *W. Diffie, M. Hellman*. “Exhaustive Cryptanalysis of the NBS Data Encryption Standard”. IEEE Computer 10 (6) : 74–84, 1977.

2. *P. N. Klein, T.B. Sebastian, B.B. Klimia.* “Shape matching using edit-distance: an implementation”. In Proc. 12th Annu. ACM-SIAM Symp. on Discrete Algorithms, 781–790, 2001.

3. *P. N. Klein, S. Tirthapura, D. Sharvit, B. B. Klimia.* “A tree-edit distance algorithm for comparing simple, closed shapes”. In Proc. 11th Annu. ACM-SIAM Symp. on Discrete Algorithms, 696–704, 2000.

4. *T. J. Warnow.* “Tree Compatibility and Inferring Evolutional History”. Journal of Algorithms 16: 388–407, 1994.

5. *Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн*. «Алгоритмы: построение и анализ». 2-е изд. Изд-во Вильямс, 2011.

СКЕЛЕТЫ МНОГОГРАННИКОВ И ИХ СЕЧЕНИЯ

В. И. Гориховский

Санкт-Петербургский государственный университет

Введение

В настоящее время одним из главных направлений в компьютерной графике является 3d-моделирование. Поскольку использование полных моделей требует больших вычислительных ресурсов, активно исследуются менее затратные способы представления 3d-объектов. Одним из подобных представлений являются 3d-скелеты.

В работах [1–3] скелеты определялись как след граничных элементов многогранника (ребер или вершин) при равномерном стягивании его границы. Под стягиванием понималось равномерное движение граней или ребер в направлении внутренней нормали. Алгоритм построения 3d-скелета, предложенный в [3], имеет временную сложность $O(n^4 \log n)$.

В нашей работе рассматриваются 3d-скелеты, их плоские сечения, а также некоторые свойства проекций граней на секущую плоскость.

Процессы построения скелета и его сечения

Определим процесс распространения линейного фронта, порожденного многогранником P. В начальный момент времени фронт $\mathrm{SCP}(P, 0)$ идентичен границе P; далее все грани начинают движение внутрь P с одинаковой постоянной скоростью. В ходе распространения структура фронта меняется; в конечном счете, все компоненты фронта исчезнут. Линейный фронт в момент времени t будем обозначать через $\mathrm{SCP}(P, t)$.

Рассмотрим многогранник P и его сечение некоторой плоскостью S, не содержащей вершин P. Очевидно, S является секущей и по отношению к линейному фронту в период времени $(0, Дt)$, для некоторого $Дt > 0$. Его сечение представляет собой множество многоугольников, изменяющихся со временем: ребра каждого из них движутся внутрь со скоростями, индуцированными скоростями граней. Таким образом, понятие линейного фронта и его распространения применимо и к сечению, однако каждое ребро сечения в общем случае перемещается со своей скоростью. Для любого момента времени $t \geq 0$, будем обозначать сечение линейного фронта $\mathrm{SCP}(P, t)$ плоскостью S через $\mathrm{SCP}(P \cap S, t)$.

В работе показано, что, используя информацию о сечении $\mathrm{SCP}(P \cap S, t)$, мы можем построить часть сечения скелета.

Очевидно, что в большинстве случаев мы не сможем таким образом построить все сечение скелета, поскольку в какой-то момент секущей плоскости может достигнуть еще одна грань.

Лемма. Пусть P — многогранник, S — секущая плоскость, не содержащая вершин P. Найдется $\Delta t > 0$, такое, что для любого момента времени $t \in (0, \Delta t)$ выполняется:

$$\mathrm{SCP}(P, t) \cap S = \mathrm{SCP}(P \cap S, t).$$

Из данной леммы следует, что, учитывая влияние граней, не пересекающих секущую плоскость в начальный момент времени, мы сможем построить сечение скелета многогранника, не строя весь скелет.

Построение сечения многогранника: выпуклый случай

Будем строить сечение скелета выпуклого многогранника плоскостью, не параллельной ни одной из граней. Для этого будем хранить информацию о внешних гранях в виде движущихся прямых — пересечений плоскостей, содержащих грани, с секущей. Затем строим скелет при помощи модифицированного алгоритма Фелкеля и Обдржалека [4], добавляя во фронт ребра в те моменты, когда некоторые прямые достигают вершин обрабатываемого многоугольника.

Алгоритм.

1. Строим пересечения плоскостей содержащих грани и секущей плоскости.
2. Вычисляем для каждой прямой время, за которое она достигнет многоугольника, и упорядочиваем прямые в соответствии с полученными значениями.
3. Повторяем, пока внутренность многоугольника непуста: строим скелет по стандартной схеме до того момента, пока какая-то прямая не достигнет многоугольника в некоторой его вершине; при наступлении такого момента заменяем данную вершину ребром нулевой длины, параллельным и движущимся со скоростью рассматриваемой прямой; обновляем расстояния.

Проведём оценки времени и памяти, необходимой для исполнения алгоритма. Шаги 1 и 2 выполняются за время $O(n \log n)$, где n — количество граней исходного многогранника. Построение скелета также требует $O(n \log n)$ времени, так как общее число ребер, появляющихся в границе многогранника в процессе распространения линейного фронта, не превосходит n.

Теорема. Построение скелета сечения многогранника требует $O(n \log n)$ времени и $O(n)$ памяти

Мы говорили о сечениях плоскостями, не параллельными граням. Можно уточнить алгоритм таким образом, чтобы не было необходимости в указанном ограничении. Для этого нужно найти все грани, параллельные секущей

плоскости, определить, в какой момент они достигнут сечения, и добавить в этот момент грань к скелету. Это уточнение не повлияет на оценки временной и емкостной сложности алгоритма.

Построение сечения многогранника: простой невыпуклый случай

Реальные объекты крайне редко бывают выпуклыми, поэтому при моделировании необходимо уметь обрабатывать невыпуклые случаи. Существуют различные к этому подходы. Например, в работе [3] подробно описывается построение скелета произвольного многогранника и разрешение противоречий, возникающих при обработке невыпуклых случаев.

Рассмотрим простой невыпуклый случай — клин в виде треугольной пирамиды, приближающийся к плоскости сечения. Будем считать, что этот клин пересечет сечение.

Подобное событие обрабатывается следующим образом. Определить время касания клина и плоскости сечения и добавить в сечение треугольную «дыру» — многоугольник, внутренность которого не принадлежит сечению, со сторонами, имеющими соответствующие скорости. В момент касания это многоугольник вырожден в точку.

Возникает вопрос, каким образом можно идентифицировать подобные случаи. Один из вариантов — использование заметающих плоскостей, параллельных секущей и проходящих через вершины P. Если ребра, инцидентные рассматриваемой вершине, лежат по одну сторону от заметающей плоскости, но во фронте есть и вершины, лежащие по другую ее сторону, то это и есть вершина клина.

Таким мы образом мы получаем алгоритм с той же асимптотической временной и емкостной сложностью, что и в выпуклом случае.

Дальнейшие перспективы

Дальнейшими направлениями развития данной работы является построение сечений многогранников в более сложных случаях, чем те, что были рассмотрены выше. Также возможно проведение анализа взаимосвязей между скелетами и диаграммами Вороного.

Другой целью является построение плоской структуры, по которой можно восстановить 3d-скелет многогранника. Вероятно, это позволит создать алгоритм построения 3d-скелета, работающий за лучшее время, чем существующие на сегодняшний день алгоритмы.

Литература

1. *Aichholzer, O., Aurenhammer, F., Alberts, D., Gärtner, B.* A novel type of skeleton for polygons. J. of Universal Computer Science 1 (12), 752–761 (1995).

2. *Aichholzer, O., Aurenhammer, F.* Straight skeletons for general polygonal figures in the plane. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, Springer, Heidelberg, 117–126. (1996).

3. *Gill Barequet, David Eppstein, Michael T. Goodrich, Amir Vaxman*. Straight Skeletons of Three-Dimensional Polyhedra. In Proc. 16th Annu. European symp. on Algorithms, LNCS, vol. 5193, Springer-Verlag, 148–160 (2008).

4. *Felkel, P., Obdržálek, Š.* Straight skeleton implementation. L.Szirmay-Kalos (ed.), Proc. Spring Conf. on Computer Graphics, Budmerice, Slovakia, 210–218 (1998).

ВЗВЕШЕННЫЕ СКЕЛЕТЫ ДЛЯ ВЫПУКЛЫХ МНОГОГРАННИКОВ

И. В. Макеев

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Понятие прямолинейного скелета было впервые введено в рассмотрение для случая простых многоугольников [1]; позже было предложено его обобщение на случай многогранников [2].

Рассмотрим многогранник в трехмерном пространстве. Представим, что граница многогранника движется внутрь, причем скорости всех граней постоянны и равны, а грани движутся, оставаясь параллельными самим себе.

В каждый момент времени границу многогранника(-ов) будем называть линейным фронтом. В ходе распространения, линейный фронт невыпуклого многогранника может разделиться на две или более частей.

Процесс стягивания продолжается до тех пор, пока линейный фронт не исчезнет.

Для многогранника введем набор весов $W = (w_1, w_2, ..., w_n)$, сопоставляющий каждой грани ее вес w_i, где все w_i неотрицательны. Вес определяет скорость движения грани в процессе распространения линейного фронта.

Определение. Взвешенный скелет многогранника — это объединение множества поверхностей, заметаемых ребрами многогранника(-ов) в процессе распространения линейного фронта, в котором каждая грань f_i движется со своей скоростью w_i.

Прямолинейный скелет имеет приложения в архитектуре [4] и может быть использован при восстановлении поверхностей по контурам [3].

Построение взвешенного скелета

В 2008 году четырьмя авторами был совместно разработан алгоритм построения скелета для многогранника [2].

Мною предложена и реализована модификация этого алгоритма для случая построения взвешенного скелета выпуклого многогранника. В качестве языка программирования была выбрана Java.

Программа позволяет рисовать многогранник, сохранять его в файл и загружать из файла, назначать граням многогранника веса и строить взвешенный скелет.

Всюду в дальнейшем будем предполагать, что рассматриваемый многогранник не имеет параллельных граней, и в каждой его вершине сходятся три ребра.

Когда в процессе распространения линейного фронта исчезает ребро, будем говорить, что происходит событие. В этот момент четыре грани ли-

нейного фронта оказываются инцидентными общей вершине, и структура линейного фронта изменяется.

Алгоритм воспроизводит последовательность событий в процессе распространения линейного фронта.

События для ребер многогранника заносятся в очередь с приоритетами. Из очереди извлекается событие для ребра, которое исчезает раньше других. Затем, в соответствии с извлеченным событием, происходит обновление структуры линейного фронта, и создается грань скелета. В очередь помещаются события для ребер обновленного линейного фронта. Эта последовательность действий повторяется, пока очередь не опустеет.

Может оказаться, что событие, извлеченное из очереди на некотором шаге алгоритма, не соответствует текущей конфигурации линейного фронта. Тогда это событие произойти не может. Такое событие будем называть невозможным.

Алгоритм построения взвешенного скелета

1. Инициализация:
Поместить все возможные события в очередь Q.

2. While Q непуста do:

(a) Взять очередное событие I из Q. Если I невозможно, перейти к шагу 2.
(b) Создать узел скелета в точке, в которой произошло событие, и соединить ее с вершинами, задействованными в событии.
(c) В список граней скелета добавить грань, сформированную в результате выполнения шага 2 (b).
(d) Обновить структуру линейного фронта в соответствии с шагом 2 (b).
(e) Добавить в очередь события для вновь созданных ребер.

Типы событий:

1. Событие ребра.

Событие ребра происходит в тот момент, когда в процессе распространения линейного фронта исчезает его ребро. Пусть исчезает ребро e_i, вершинам которого инцидентны ребра e_1, e_2 и e_3, e_4. Ребро e_i стягивается в вершину v_e многогранника, степень которой равна четырем. Затем вершина v_e расщепляется, и возникает новое ребро e^*_i (рис. 1, *а*).

2. Событие грани.

Событие грани происходит в тот момент, когда в процессе распространения линейного фронта исчезает грань многогранника. В общем случае исчезают только треугольные грани.

Пусть в процессе распространения линейного фронта исчезает ребро e_i многогранника, и при этом грань f_1, содержащая ребро e_i, является треу-

гольником. В этом случае треугольная грань f_1 стягивается в точку, и ее вершины сливаются. Появляется новая вершина v^*, инцидентная ребрам e_1, e_2, e_3 (рис. 1, *b*).

Создание грани скелета.

Пусть происходит событие для ребра e_i. На шаге 2 (b) алгоритма создаются новые ребра скелета e_{i1} и e_{i2}, имеющие общую вершину v_i. Ориентируем ребра e_{i1} и e_{i2} от вершины v_i.

В список граней скелета нужно добавить грань f_i, сформированную после события для ребра e_i. Грань f_i будет содержать ребра e_{i1} и e_{i2}, а также другие, построенные ранее, ребра скелета, лежащие с ними в одной плоскости. Чтобы определить эти ребра, будем обходить ребра скелета, начав с ребер e_{i1} и e_{i2}, каждый раз переходя на ребро, лежащее в одной плоскости с ребрами e_{i1} и e_{i2}. Нельзя переходить на ребро, направление которого противоположно направлению обхода. В результате мы получим набор ребер грани f_i. Если происходит событие грани, то аналогичным образом создаются три грани.

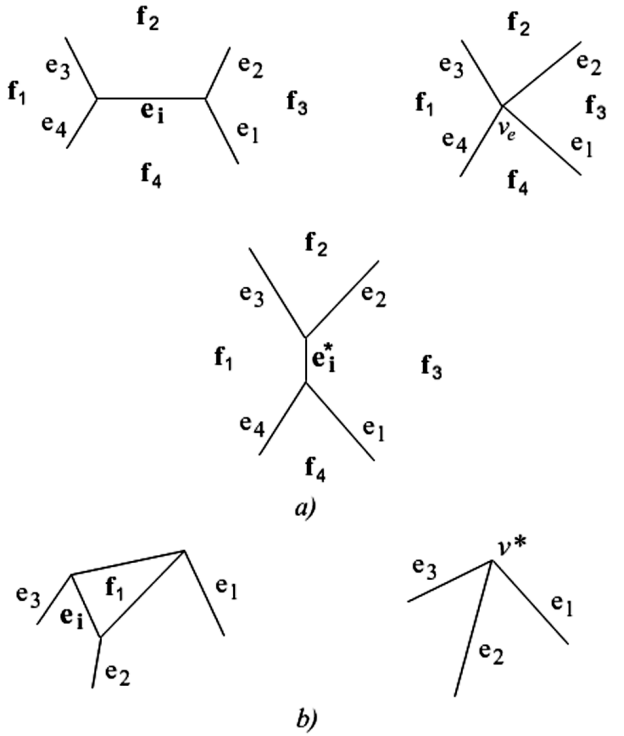


Рис. 1

Временная сложность алгоритма составляет $O(n^2 \log(n))$, где n — число граней многогранника.

Литература

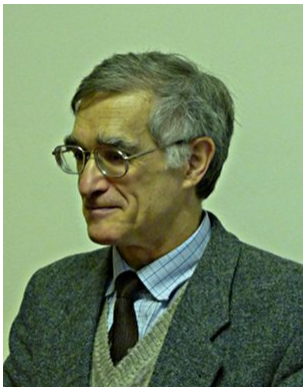
1. *Aichholzer, O., Aurenhammer, F., Alberts, D., Gärtner, B.* A novel type of skeleton for polygons. J. of Universal Computer Science 1 (12), 752–761, 1995.

2. G. Barequet, D. Eppstein, M.T. Goodrich, and A. Vaxman, Straight skeletons of three-dimensional polyhedra. Proc. 16th Ann. European Symp. on Algorithms (ESA), Karlsruhe, Germany, Lecture Notes in Computer Science, 5193, Springer-Verlag, 148–160, 2008.

3. *G. Barequet, M. T. Goodrich, A. Levi-Steiner*, and *D. Steiner*. Contour interpolation by straight skeleton. Graphical Models (GM), vol. 66 (4), 245–260, 2004.

4. *A. Recuaero* and *J. P. Gutierrez*. Sloped roofs for architectural CAD systems. Microcomputers in Civil Engineering 8: 147–159, 1993.

Методы хранения и поиска информации



Новиков
Борис Асенович

д.ф.-м.н.
профессор кафедры информатики СПбГУ

ИСПОЛЬЗОВАНИЕ ОБРАТНОЙ СВЯЗИ ПРИ ПОИСКЕ ИЗОБРАЖЕНИЙ

М. С. Шорникова (e-mail: mary.shornikova@gmail.com)

СПбГУ, математико-механический факультет

Разработка эффективных алгоритмов поиска изображений, при которых индексирование изображений выполняется без участия человека, является важной задачей. Решение этой задачи затрудняется многообразием и разнородностью коллекций изображений, в которых осуществляется поиск.

Один из способов — поиск изображений по текстовым аннотациям. В этом случае необходимо получить для каждого изображения множество описывающих его слов. Задача поиска изображения сводится к текстовому поиску, обладающему надежными алгоритмами. Однако такой подход представляется трудным с точки зрения как создания самого описания, так и субъективности текстового описания изображения.

Альтернативой поиску по текстовым аннотациям является поиск изображений по содержанию. Он не использует никакой дополнительной информации такой как текстовая аннотация, время или место создания изображения. Метод основан на анализе содержания изображений — численных характеристик составляющих его пикселей.

К характеристикам изображения относят цвет, текстуру и форму объектов на изображении. Для каждой характеристики существует значительное количество способов построения векторов признаков. Векторами признаков (или просто признаками) называют наборы численных параметров, описывающие конкретные изображения. Степень близости изображений определяется заданной на пространстве векторов признаков метрикой.

Близкие по метрике изображения могут оказаться далёкими с точки зрения семантики. Такое различие называется «семантическим разрывом». Одним из наиболее эффективных подходов для устранения этого различия является использование обратной связи при поиске изображений. Поиск с использованием обратной связи включает следующие действия:

1. По запросу пользователя система выдаёт ему результаты поиска;
2. Пользователь отмечает подходящие изображения или оценивает релевантность изображений по условной шкале;
3. Система учитывает оценку пользователя при следующей выдаче результатов.

Число алгоритмов, использующих обратную связь, велико [1] Широко применяются алгоритмы, комбинирующие обратную связь и кластеризацию. Например, для вычисления важных признаков изображения для запроса, авторы работы [2] кластеризуют признаки и пользовательские логи одновре-

менно. В работе [3] результаты поиска на каждой итерации упорядочиваются при помощи кластеризации.

Алгоритм, описанный в работе, отличается новым способом применения кластеризации. Он заключается в том, что база изображений предварительно разбивается на кластеры изображений, близких по некоторому заранее выбранному признаку. При поиске на каждой итерации выдаются изображения, близкие по другому признаку к релевантным изображениям. Изображения выбираются из кластеров, в которых находятся релевантные изображения.

Более подробно об алгоритме. Пусть F1, F2 — признаки изображений. Для каждого изображения в базе вычисляются значения обоих признаков. База изображений кластеризуется по значениям признака F1 методом «*K* средних». После кластеризации значения признака F1 не нуждаются в хранении.

На первой итерации поиска находятся *N* изображений, близких по признаку F2. На последующих итерациях показывают *N* изображений близких по признаку F2 из кластеров, к которым принадлежат релевантные изображения. Из каждого кластера выбирают *M* изображений, где *M* — отношение *N* к числу кластеров, в которых есть релевантные изображения. Если *M* больше числа непросмотренных изображений в кластере, то оставшиеся выбирают близкими по признаку F2 из всей базы.

При численных экспериментах в качестве признака F2 было выбрано число ненулевых столбцов двадцатисемицветной гистограммы, построенной по цветовой модели HSV [4]. Признак текстуры энергия [5] был выбран в качестве F1.

Эксперименты проводились на базе, превышающей 45 тысяч изображений млекопитающих 454 видов. Разделение на виды было использовано для моделирования обратной связи.

В качестве изображения-образца выбиралось произвольное изображение из базы. Релевантными считались изображения животных того же вида, что и на изображении-образце. Число итераций было ограничено 10. На каждой итерации система выдавала 50 изображений.

Для 239, 500 и 898 кластеров было проведено по 100 запросов. Для сравнения те же изображения-образцы отправлялись системе поиска без учёта кластеризации. На каждой итерации у обеих систем вычислялись точность и полнота поиска.

Лучшие результаты — увеличение на 5.98 % точности поиска и на 2.81 % полноты поиска — были получены при количестве кластеров, равном 239. С пятой итерации точность увеличивалась больше, чем на 7 %. Но на первых 4 итерациях метод давал уменьшение точности почти на 3 %. А при взаимодействии с пользователем важно получить лучшие результаты как можно раньше.

Лучшие результаты на начальных итерациях показал поиск с 898 кластерами (табл. 1).

Таблица

Итерация	Увеличение точности (%)	Увеличение полноты (%)
1	–0.61	3.17
2	1.6	3.88
3	2.66	2.62
4	4.26	2.93
5	6.91	6.56
6	7.45	6.65
7	7.98	6.98
8	7.89	1.31
9	8.42	1.63
10	10	2.68
Среднее	5.66	3.84

Примечание: Нет противоречия в том, что полнота поиска увеличилась, а точность — уменьшилась (как на первой итерации). Это означает, что алгоритм сработал лучше на запросе, для которого в базе содержалось мало релевантных изображений, и хуже — для которого много релевантных изображений.

Литература

1. *Zhou X. S., Huang T. S.* Relevance feedback in image retrieval: A comprehensive review. Multimedia Systems. 2003. Vol. 8, No. 6. Pp. 536–544.

2. *Rege M., Dong M., Fotouhi F.* Co-Clustering Image Features and Semantic Concepts. Proceedings of ICIP'06. Pp. 137–140.

3. *Chen Y., Wang J. Z., Krovetz R.* Content-Based Image Retrieval by Clustering. MIR'03 Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval. Pp. 193–200.

4. HSL and HSV. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/HSL_and_HSV

5. *Haralick R. M., Shanmugam K., Dinstein I.* Textural Features for Image Classification. IEEE Transactions on Systems. 1973. Vol. 3. No. 6. Pp. 610–621.

АНАЛИЗ ВЕРОЯТНОСТНЫХ АЛГОРИТМОВ МНОГОМЕРНОГО ИНДЕКСИРОВАНИЯ

А. В. Волохов

Санкт-Петербургский государственный университет

1. Введение

Известно, что вычислительная сложность задачи поиска точных ближайших соседей в многомерном пространстве экспоненциально возрастает с ростом размерности. Чтобы обойти эту проблему, решается схожая задача — поиск примерных ближайших соседей. Её специфика состоит в том, что при примерном поиске ответ на запрос радиуса R будет верным, если возвращаемые точки находятся на расстоянии, не превышающем $c \cdot R$ от точки запроса. В свою очередь, это позволяет использовать вероятностные алгоритмы индексирования многомерных данных.

2. Цели

Цель данной работы состояла в сравнении скорости и качества работы алгоритмов пространственно-чувствительного хеширования (Locality-Sensitive Hashing), предназначенных для поиска в пространствах с различными функциями расстояния. Методы индексирования сильно зависят от пространства, в котором этот поиск выполняется, поэтому было логично предположить, что разные алгоритмы поиска, запущенные на одном и том же наборе данных будут давать разные по качеству результаты.

В работе использовались алгоритмы для евклидовой и косинусной мер близости.

3. Алгоритмы и структуры данных

Оба алгоритма были реализованы в промышленной СУБД Oracle.

Объекты базы данных — многомерные вектора.

3.1. Описание алгоритмов

В обоих случаях, точки из набора данных хешируются точками из предопределённого набора случайных векторов (с координатами из нормального распределения)

3.1.1. Хеширование случайными гиперплоскостями

Здесь случайный вектор определяет нормальную гиперплоскость. Хеш семейство этого алгоритма выглядит следующим образом:

$$h_r(v) = \begin{cases} 1, v.p \geq 0 \\ 0, v.p < 0 \end{cases},$$

где p — предопределённый вектор, v — вектор базы данных.

Каждая такая хеш функция определяет полупространство, в котором находится вектор базы данных. Тогда угол между двумя векторами v и u можно определить по следующей формуле:

$$Pr\,[h_p(v) = h_p(u)] = 1 - \theta \frac{(u,\, v)}{\pi}.$$

Использование конкатенации нескольких хеш функций семейства позволяет повысить точность определения угла между векторами.

Поиск:

1. Вычислить значения всех хеш функций для точки запроса.
2. Получить все точки из корзины с данным хешем.
3. Если количество точек достаточно для удовлетворения запроса, закончить поиск.
4. Получить номер следующей корзины из списка ближайших корзин.
5. Повторить пункты 2–5 для полученной корзины.

3.1.2. Хеширование с использованием *s*-устойчивых распределений

В данном алгоритме, устойчивость нормального распределения позволяет использовать скалярные произведения предопределённых случайных векторов и точек базы данных для определения Гауссовой нормы разности двух векторов базы данных.

Хеш семейство в этом случае будет следующим:

$$h(v) = \left\lfloor \frac{a \cdot v + b}{w} \right\rfloor,$$

где a — случайный вектор, b — случайное смещение в пределах [0 ... w], w — размер корзины.

Чтобы обеспечить высокую вероятность удовлетворения запроса при низкой вероятности коллизии далёких точек, необходимо использовать конкатенацию из k хеш функций семейства и затем хешировать L композитными функциями.

Параметры k и L вычисляются по следующей формуле:

$$\delta = 1 - (1 - P_1^K)^L,$$

где δ — вероятность получения верного ответа, P_1 — вероятность коллизии близких точек.

Для каждой полученной таким образом пары параметров строятся хеш-таблицы. Полученная структура проверяется на тестовом множестве, после чего выбирается оптимальная по скорости пара параметров.

Поиск:
1. Прохешировать точку запроса первой комбинированной функцией.
2. Получить все точки из корзины с данным хешем.
3. Если количество точек недостаточно для удовлетворения запроса, повторить пункты 1–3 для следующей хеш функции.
4. Закончить поиск.

3.2. Структуры данных

В обоих случаях исходный набор данных хранится в таблице.

vectors

Sur_key	Number
Att_1	Number
...	...
Att_k	Number

Набор предопределённых случайных векторов хранится в таблице.

pivots

Sur_key	Number
Att_1	Number
...	...
Att_k	Number

Также реализована процедура, создающая обе эти таблицы для данных конкретной размерности.

3.2.1. Хеширование случайными гиперплоскостями

Таблица Hash_table содержит данные данные ключ-значение и список корзин, находящихся на единичном расстоянии от неё.

Sur_key	Number
Hash_value	Varchar2
Points	Nested table of number

Hyperlane_stat содержит статистику по работе алгоритма: среднее время ответа на запрос, среднее число угаданных точных ближайших соседей, отношение среднего расстояния до найденных точек к среднему расстоянию до точных ближайших соседей.

response_time	Number
precision	Number
mean	Number

3.2.2. Хеширование с использованием *s*-устойчивых распределений

В таблице pivots_hash_table содержится таблица соответствий между k случайными векторами и i-ой хеш функцией.

sur_key	Number
pivot_id	Number

L одинаковых таблиц hash_table_i содержат хеш таблицы для соответствующих функций.

value_1	Number
...	...
value_k	Number
Points	Nested table of number

Таблица sstable_stat содержит статистические данные, аналогичные хранящимся в соответствующей таблице для гиперплоскостного алгоритма.

response_time	Number
precision	Number
mean	Number

4. Эксперименты и анализ

Эксперименты проводились на данных, относящихся к image content data retrieving. База данных состояла из 25000 тысяч векторов размерности 41. В качестве тестового множества было взято 20 % исходного множества. На стадии препроцессинга были вычислены оптимальные параметры для каждого из алгоритмов — длина хамминговой строки в первом алгоритме была равна 17, параметры k и L во втором — 6 и 7 соответственно. В качестве критерия скорости работы алгоритма использовалось процентное посчитанных расстояний(обработанных точек) к общему количеству точек. Точность определялась как количество верно найденных точных ближайших соседей к общему количеству ближайших соседей. Эксперименты показали следующие результаты:

Количество обработанных точек:
Гиперплоскости: 2.98 %;
s-устойчивое распределение: 5.55 %.

Точность:
Гиперплоскости: 30.8 %;
s-устойчивое распределение: 33.3 %.

Отклонение расстояния до найденных ближайших соседей от расстояния до точных ближайших соседей:
Гиперплоскости: 0.07;
s-устойчивое распределение: 0.04.

5. Заключение

Полученные результаты позволяют заключить, что, несмотря на схожую точность, алгоритм для евклидового пространства даёт в среднем более релевантные результаты при худшей скорости работы.

Литература

1. *A. Andoni, M. Datar, N. Immorlica, P. Indyk,* and *V. Mirrokni.* Locality-sensitive hashing scheme based on p-stable distributions. Nearest Neighbor Methods for Learning and Vision, Neural Processing Information Series. MIT Press, 2005.

2. *D. Ravichandran, P. Pantel,* and *E. Hovy.* Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. ACL'05: Proceedidngs of the 43rd Annual Meeting on Association for Computational Linguistics. Pp. 622–629. 2005.

3. *P. Zezula, G. Amato, V. Dohnal, M. Batko*. Similarity Search — The Metric Space Approach. Springer, 2006.

ПРЕДСТАВЛЕНИЕ И ОБРАБОТКА СЛОЖНЫХ ЗАПРОСОВ В СИСТЕМАХ ПОИСКА ИЗОБРАЖЕНИЙ

Б. А. Новиков, А. С. Ярыгина

Современные задачи и методы поиска информации не ограничены поиском текстов и изображений. Необходимость обработки и поиска сложных объектов, а также реализации сложных сценариев поиска, требует построения и выполнения сложных поисковых запросов.

Задачи поиска, основанные на построении сложных запросов, можно разделить на два класса:

- связанные с адекватным отображением и описанием природы процесса поиска (явное уточнение запроса, обратная связь от пользователя);
- поиск сложных объектов (текстов с изображениями, объектов из бизнес аналитики).

Целью работы является построение средства, механизма описания сложных видов поиска на основе подобия. Предлагаемый инструмент должен представлять собой однородную систему способов и операций, предназначенную для формирования и декомпозиции поисковых запросов, не зависящую от конкретной задачи поиска. Необходимость построения общего инструмента для синтеза данных была обозначена в литературе [2, 4].

В рамках работы построена модель, позволяющая комбинировать различные типы подзапросов внутри одного запроса. Центральным понятием нашей модели является понятие *Q*-множества.

Q-множеством называется множество S вместе с определенной на нем функцией оценки $score : S \to [0,1]$.

Q-множество позволяет абстрагироваться от конкретного языка запросов, природы объектов в пространстве поиска, методов выполнения запросов, и включает в себя запрос и результат его выполнения.

Пользовательский запрос может представлять собой текст, изображение, некоторый объект с множеством атрибутов, но природа запроса, независимо от его структуры, с точки зрения поисковой системы всегда может быть описана некоторой функцией оценки. Отличие *Q*-множества от ответа на пользовательский запрос состоит в том, что ответ, во-первых, упорядочен, во-вторых, содержит не все объекты из множества. В рамках работы построен набор операций, основанный на представлении любого запроса в виде *Q*-множества независимо от природы объектов и метода построения их оценок.

Q-множество может быть естественным образом построено по запросу и функции расстояния: $score(e) = similarity(q, e)$, где *similarity* – функция подобия, $e, q \in S$.

Введенное выше понятие *Q*-множества по существу совпадает с понятием нечеткого множества, предложенного Заде. Операции теории нечетких множеств использованы в нашей модели и расширены дополнительными

операциями, поскольку обычных теоретико-множественных операций не достаточно при работе с Q-множествами.

В работе предложен набор операций над классом Q-множеств, который позволяет выражать высокоуровневые операции необходимые при обработке сложных объектов и сложных видов поиска. Набор операций представляет собой формальную систему методов калибровки и синтеза Q-множеств и позволяет описывать сложные поисковые запросы и методы их построения.

При анализе сложного объекта пользователь, как правило, оценивает все его признаки и атрибуты в совокупности, но при построении поисковой системы встает задача точного описания интуитивного процесса синтеза нескольких критериев поиска.

При решении задачи синтеза Q-множеств необходимо учитывать ряд интуитивных представлений пользователя о синтезе результатов поиска [2,3]. Первым свойством является «Эффект хора», основная идея которого заключается в том, что хорошие объекты по двум признакам должны быть в итоге лучше, чем хорошие только по одному. Вторым свойством является «Эффект снятия сливок», который состоит в том, что хорошие объекты в смысле хотя бы одного из синтезируемых множеств должны быть хорошими и в результате.

Синтезом двух Q-множеств называется функция $fusion : Scores(S) \times Scores(S) \rightarrow Scores(S) : \forall a, b \in Scores(S)\ fusion(a, b) = fusion(b, a)$, сохраняет упорядоченность (если $a(x) < a(y)$, $b(x) < b(y)$, то $fusion(a, b)(x) < fusion(a, b)(y)$).

В разных задачах поиска возникают те или иные информационные потребности и, следовательно, не существует универсального метода синтеза двух запросов. В зависимости от конкретной задачи алгоритмы и формулы, реализующие операцию синтеза могут по-разному учитывать «Эффект хора» и «Эффект снятия сливок». В рамках работы мы рассмотрели несколько реализаций операции синтеза: унаследованные из теории нечетких множеств и известный алгоритм CombMNZ, который считается наилучшим при синтезе разнородных признаков объектов при поиске изображений ([1]). Формулы, реализующие некоторые из операций синтеза, представлены в таблице 1. Использование операций super-union и super-intersect позволяет сохранять вероятностную интерпретацию оценок в случае, если комбинируемые Q-множества независимы.

При решении задачи синтеза Q-множеств очень часто возникает задача сопоставления оценок разной природы, например, если при поиске изображений синтезируются Q-множества на основе текстуры и цвета. Функции расстояния на соответствующих пространствах поисковых образов будут разные, и поэтому, как правило, получаемые оценки будут несопоставимы. На уровне нашей абстрактной модели мы рассматривали операции калибровки.

Операции калибровки могут учитывать ряд важных свойств:

- чувствительность к выбросам (оценка отдельного объекта внутри Q-множества не должна сильно влиять на результат калибровки);
- скос (высокие оценки внутри Q-множества сильнее влияют на результат калибровки, поскольку именно объекты с высокими оценками составляют результат);

- эффективность (разница в качестве Q-множеств, являющихся аргументами операции синтеза, принимается во внимание при калибровке).

В работе было рассмотрено несколько реализаций процедур калибровки на основе операций нормализации и усиления Q-множеств.

Операцией нормализации называется монотонная неубывающая функция $norm : Scores(S) \rightarrow Scores(S)$: не изменяет соотношения между оценками внутри Q-множества.

Операцией усиления называется неубывающая функция $strengthen : Scores(S) \times [0,1] \rightarrow Scores(S)$: увеличивает оценки выше заданного уровня, и уменьшает остальные.

На основе предложенной формальной системы операций представлено несколько спецификаций процедур калибровки и синтеза оценок объектов. Заложенные в различные реализации операций синтеза и калибровки параметры позволяют настраивать операции под специфические, конкретные задачи поиска. Например, операция усиления зависит от порога, отделяющего высокие оценки от низких, и параметра, определяющего степень усиления или ослабления Q-множества.

В рамках исследования выявлены и проанализированы общие принципы и механизмы работы с Q-множествами (синтез и калибровка), а также определены свойства предложенных операций и их взаимосвязь с потребностями пользователя и свойствами конкретных задач построения сложных запросов.

В качестве примера работы со сложными объектами рассмотрена задача поиска изображений по образцу на основе нескольких признаков: цветовых гистограмм, цветовых моментов и текстуры. Для улучшения качества поиска изображений за счет более полного анализа свойств объектов использовались подходы, основанные на комбинировании разных признаков.

Результаты были проанализированы и сопоставлены с результатами, полученными на основе известных подходов к комбинированию нескольких признаков при поиске изображений по содержанию.

В таблице 1 показано значение R-precision результатов синтеза Q-множеств на основе цветовых моментов и гистограмм.

Таблица 1

R-precision при синтезе Q-множеств на основе цветовых моментов и гистограмм

	without norm	**norm-avg**	**norm-maxmin**	**norm&strength**
CombMNZ $(s_1 + s_2) * R^1$	0,49	0,53	0,49	0,53
super-intersect $s_1 * s_2$	0,50	0,53	0,50	0,53
super-union $1 - (1 - s_1) * (1 - s_2)$	0,49	0,52	0,50	0,52

Результаты экспериментов показали, что качество синтезированного Q-множества в значительной степени зависит от точности первоначальных множеств, и выбранного метода калибровки функций оценки. Различные реализации функции синтеза привели к сопоставимым результатам. Тем не менее, детальный анализ свойств описанной модели позволяет предсказывать поведение операций, в зависимости от конкретных требований задачи поиска. Таким образом, в рамках работы был предложен подход к единообразному описанию сложных видов поиска на основе подобия.

Литература

1. *Н. Васильева, А. Дольник, И. Марков.* Поиск изображений. Синтез различных методов поиска при формировании результатов // Интернет-Математика 2007: Сборник работ участников конкурса. Екатеринбург: Изд-во Урал. ун-та, 2007. С. 46–55.

2. *J. Kludas, E. Bruno,* and *S. Marchand-Maillet.* Adaptive multimedial retrieval: Retrieval, user, and semantics // Information Fusion in Multimedia Information Retrieval. Springer-Verlag, Berlin, Heidelberg, 2008. Pp. 147–159.

3. *D. Lillis, F. Toolan, R. W. Collier, and J. Dunnion.* Probfuse: a probabilistic approach to data fusion // In E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Jarvelin, editors, SIGIR. ACM, 2006. Pp. 139–146.

4. *L. Valet, G. Mauris,* and *Ph. Bolon.* A statistical overview of recent literature in information fusion // Proceedings of the Third International Conference on Information Fusion, 1: MOC3/22 –MOC3/29, 2000.

КЛАССИФИКАЦИЯ ТЕКСТОВ ПО ВОЗРАСТНОМУ И ГЕНДЕРНОМУ ПРИЗНАКУ АВТОРА

К. С. Туманова

Санкт-Петербургский государственный университет

Одним из активно развивающихся в последнее время направлений в области обработки естественных языков (NLP, Natural Language Processing) является автоматическое профилирование автора (Author Profiling). Задачи, решаемые в рамках этого направления, заключаются в построении с помощью методов машинного обучения моделей, которые для любого входного текста будут возвращать информацию об авторе этого текста, некоторые его значимые характеристики, например, пол, возраст, принадлежность к той или иной социальной группе, страну или регион происхождения, психологический тип и др.

Большинство исследований в этой области проводилось для классификации англоязычных текстов и по отдельно взятому признаку, например, по полу[5], свойствам характера [3], принадлежности к той или иной идеологической организации [4] и т. п.

Но современными учеными-лингвистами было выявлено, что на речь человека все эти факторы оказывают влияние одновременно [1, 2]. Очевидно, что люди одного пола, но разных возрастов имеют разный словарный запас, используют разное количество речевых оборотов, и строят предложения различной сложности. Подобные связи можно обнаружить и между другими признаками, поэтому результаты выделения какой-то одной характеристики без учета другой могут оказаться не достаточно точными.

В данной работе будут предложены алгоритмы для автоматического профилирования автора текста, в которых учитываются возможные взаимосвязи характеристик личности. Алгоритмы будут описаны для случая двух характеристик, но эти модели могут быть естественным образом расширены.

Каждый признак, который необходимо извлечь из текста, будем называть измерением. Для классификации по двум измерения одновременно могут быть использованы следующие подходы:

1. *«Плоская» (Flat) классификация.*

Имея в первом измерении N групп, а во втором M групп, можно определить $N \cdot M$ новых классов, которые представляют собой всевозможные сочетания этих групп.

Основные недостатки данного алгоритма в том, что при построении модели на каждый класс приходится относительно небольшой объем данных, а также при больших числах N и M просмотр и анализ результатов становится

не очень удобным. Но в этом случае благодаря тому, что все комбинации исходных групп учтены, можно ожидать высокую точность классификации.

2. *Иерархическая классификация.*

Данный подход заключатся в том, что сначала классификация производится по одному признаку, а затем для получившихся классов независимо друг от друга проводится классификация по второму признаку. При этом в случае двух измерений возможны два варианта алгоритма, в зависимости от того, по какому признаку классификация производится в первую очередь.

Особенность этого метода заключается в том, что на первом уровне на каждый класс приходится больший объем данных, чем на втором, что может сказаться на точности.

3. *Многомерная классификация.*

В данном случае классификация текстов производится по каждому измерению независимо, и далее результаты экспериментов совмещаются для каждого текста базы.

Основное достоинство данного подхода в том, что на каждый класс приходится достаточное количество входных данных, даже в том случае, когда исходная база текстов относительно мала. Но при этом важно, чтобы выбранные для классификации характеристики давали хорошую точность для обоих признаков.

Работа данных алгоритмов оценивалась для классификации текстов по возрасту и полу автора. В качестве текстов были использованы русскоязычные блоги (интернет-дневники). Для каждого автора был создан текстовый файл, в котором содержалось от 20 до 1000 записей из его дневника. Все авторы были разделены на 4 возрастные группы. Информация о количестве блогов для каждой группы указана в таблице 1.

Таблица 1

	Меньше 18	20 — 27	30 — 37	Больше 40	Всего
Мужчины	14	48	64	69	195
Женщины	55	50	30	21	156
Всего	69	98	94	90	351

Текст каждого автора представлялся в виде вектора из 129 характеристик, которые можно разделить на следующие группы: частота использования знаков пунктуации; частота использования различных частей речи и их сочетаний; частота использования речевых оборотов и фразеологизмов; частота использования смайликов; информация о длине предложений и слов, и о словарном запасе автора.

Данные характеристики были выбраны на основании различий в речи людей разного возраста и пола, выявленных лингвистами.

Для классификации использовались реализации Байесовских сетей и SMO алгоритма для Support Vector Machnes в Weka tools [7]. Подробную информацию об этих алгоритмах можно найти в [6,7]. Для каждого алгоритма была построена модель на основе половины текстов базы, а на второй половине она была протестирована.

Для подтверждения исследований лингвистов сначала были проведены эксперименты по классификации текстов только по полу автора в каждой возрастной группе независимо и для всех групп вместе. Точность классификации для каждого случая указана в таблице 2.

Таблица 2

	Меньше 18	20 — 27	30 — 37	Больше 40	Все группы
SMO	76,8 %	63,2 %	74,5 %	78,9 %	71,5 %
Bayesian Network	76,8 %	64,2 %	70,2 %	73,3 %	68,6 %

Как видно из результатов эксперимента, точность классификации внутри возрастных групп, несколько выше, чем точность классификации для всех групп. Это подтверждает предположение о том, что возраст оказывает влияние на речь человека и должен учитываться при автоматическом профилировании. Но в данном случае ввиду особенностей текстов и выбранного векторного представления, это зависимость не такая явная.

В ходе проведения основных экспериментов были получены следующие результаты. GA и AG — обозначения двух видов иерархической классификации: Gender-Age и Age-Gender.

	Точность классификации по полу и возрасту
Плоская	
SMO	30,5 %
Bayesian Network	28,2 %
Иерархическая (GA)	
SMO	31,8 %
Bayesian Network	31,2 %
Иерархическая (AG)	
SMO	33,3 %
Bayesian Network	24,1 %
Многомерная	
SMO	34,5 %
Bayesian Network	35,6 %

Точность алгоритмов оказалась не так высока. Это объясняется, во-первых, относительно небольшим размером использованной базы, и, во-вторых, тем, что выбранное векторное представление текстов основывается на различиях, выявленных между мужской и женской речью, различия между людьми разных возрастов используются в гораздо меньшей степени. Этот факт, оказал значительное влияние на точность классификации по возрастному признаку, а это в свою очередь отразилось на точности алгоритмов в целом.

При этом, как и ожидалось, плоская классификация показала самые низкие показатели по причине небольшой базы. Иерархическая классификация оказалась немного точнее, при этом вариант классификации Age-Gender оказался лучше, что опять же подтверждает предположение о влиянии возрастного признака на письменную речь. Многомерная классификация показала самые высокие показатели.

Но несмотря на незначительные отличия в работе предложенных алгоритмов, можно сделать вывод о том, что многомерная классификация наиболее подходит для задач автоматического профилирования автора.

Литература

1. *Гетте Е. Ю.* Речевое поведение в гендерном аспекте: Дис. ... канд. филол. наук, 10.02.19. Воронеж, 2004. 248 с. РГБ ОД, 61:05-10/35

2. *Горошко Е. И.* Гендерные особенности русскоязычного Интернета. http://www.textology.ru/article.aspx?aId=22

3. *S. Argamon, S. Dawhle, M. Koppel* and *J. Pennebaker* (2005). Lexical Predictors of Personality Type, Proceedings of Classification Society of North America, St. Louis MI, June 2005.

4. *M. Koppel, N. Akiva, E. Alshech* and *K. Bar* (2009). Automatically Classifying Documents by Ideological and Organizational Affiliation, Proc. of IEEE Intelligence and Security Informatics, Dallas TX, June 2009.

5. *M. Koppel, S. Argamon* and *A. Shimoni* (2003). Automatically categorizing written texts by author gender, Literary and Linguistic Computing 17 (4), November 2002. Pp. 401–412.

6. *J. Platt*. Fast training of support vector machines using sequential minimal optimization, In Advances in Kernel Methods — Support Vector Learning, MIT Press, 1998.

7. *Ian H. Witten* and *Eibe Frank*. Data Mining: Practical machine learning tools with Java implementations, Morgan Kaufmann, San Francisco, 2000.

О ДВУХ МЕТОДАХ ИСПОЛНЕНИЯ ЗАПРОСОВ ТИПА «ЗВЕЗДА»

К. К. Смирнов, Г. А. Чернышев

Санкт-Петербургский государственный университет

Введение

Среди приложений баз данных принято выделять два типа нагрузок: OLTP и OLAP. On-Line Transaction Processing, или оперативная обработка транзакций, описывает систему, в главные задачи которой входит обработка транзакций. В свою очередь в задачи OLAP или оперативной аналитической обработки входят создание, сопровождение и анализ данных и выдача отчетов [1]. Данные типы нагрузок являются диаметрально противоположными по характеру и, соответственно, требуют различных подходов в реализации систем. Первый тип нагрузок может быть охарактеризован большим количеством пользователей и изменений в данных. Для второго типа важными задачами являются обеспечение богатого инструментария, быстроты анализа большого объема данных при отсутствии изменений. В данной работе мы рассмотрим некоторые вопросы построения систем второго типа, а именно: каким образом следует выстраивать иерархии планов выполнения запросов в данных системах.

Планом выполнения запроса (Query Execution Plan) называется набор алгебраических операций подлежащих выполнению, необходимые ограничения, затрагивающие их порядок выполнения (логический план), а так же потенциальные процедуры реализации данных операций (физический план). План запроса принято изображать в виде дерева операций, в котором листья изображают операции с исходными таблицами.

В OLAP системах часто встречается тип запросов, называемый star join. Он появляется в результате применения подхода к построению хранилища данных с использованием схемы «звезда». Суть этой схемы заключается в следующем: имеется большая (по количеству записей) таблица фактов и набор небольших таблиц измерений, каждая из которых связана с таблицей фактов с помощью внешнего ключа. Основные запросы в данной схеме будут выражаться как набор соединений таблиц измерений с таблицей фактов и дальнейшей агрегацией по заданным атрибутам. При этом, фильтрация по предикатам происходит только для таблиц измерений.

В настоящей работе мы рассмотрим вопрос выбора операторов для построения дерева запроса такого типа.

Возможные решения

Для решения данной задачи мы будем рассматривать решения в терминах классической итераторной модели. Данная модель впервые была пред-

ложена в работе [4] и применена в системе Volcano. Основные предложения данной модели можно представить как предоставление унифицированного интерфейса для всего набора операторов плана запроса. При этом речь идет не только о логическом плане, но и о физических операторах. Интерфейс состоит из следующих методов: открыть/закрыть итератор, получить запись. Иногда добавляются и другие методы, такие как *reset*, который обрывает выполнение запроса и начинает выполнять работу заново. Обычно данная модель реализуется с помощью механизма наследования ООП.

Рассмотрим два возможных подхода к построению планов запросов (рис. 1):

1) Создать последовательность соединений таблицы фактов с таблицами отношений, затем к последнему в иерархии оператору применить оператор агрегирования.
2) Использовать единственный оператор мультисоединения, который будет реализовывать всю функциональность соединений, затем применить оператор агрегирования. Здесь важно отметить, что для данного типа запроса характерен малый размер таблиц измерений по сравнению с таблицей фактов.

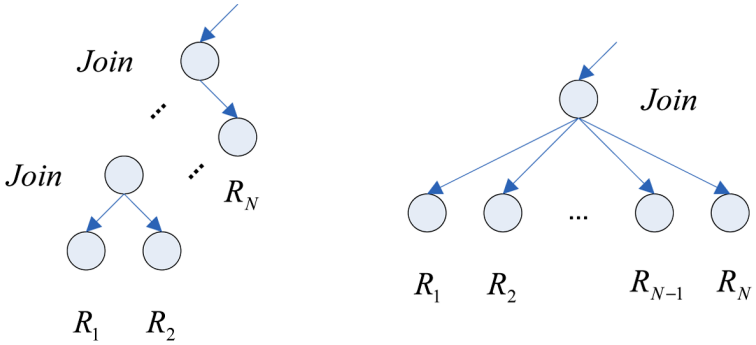


Рис. 1. Возможные подходы к построению плана запроса

Каждый из этих подходов имеет свои достоинства и недостатки. К достоинствам использования специализированного оператора можно отнести снижение накладных расходов на передачу данных между операторами (требуется меньшее количество буферов), высокий потенциал настраиваемости. Снижение расходов на передачу данных между операторами может быть очень значительным даже при условии реализации нацеленной на минимизацию количества межоператорных передач. Однако, степень снижения, как будет показано в части экспериментов, зависит от данных. Данный подход может быть рассмотрен как альтернатива построению древовидного плана запроса с минимальной или около минимальной стоимостью. Однако для его использования требуется дополнительный анализ запроса для выявления описанного шаблона. Другой серьезный недостаток — ограниченность воз-

можностей параллелизации запроса, а именно невозможность применения межоператорного параллелизма.

Эксперименты

Мы провели эксперименты с использованием нашего прототипа исполнителя SPJ запросов. Данная система была разработана авторами и заняла третье место на соревновании ACM SIGMOD в 2010 году [3]. В предыдущих частях мы не рассматривали какую-то конкретную реализацию операторов соединения, однако тип реализации может быть фактором, влияющим на выбор оператора. Существует несколько [5] способов реализации физического оператора соединения, каждый из которых имеет свои достоинства при определенных условиях.

Наша реализация использовала блочный алгоритм вложенных циклов с сортировкой и кешированием для реализации соединений. Другие особенности реализации нашей системы описаны в данной статье [2].

Эксперименты проводились на следующем оборудовании: Intel(R) Pentium(R) D CPU 3.0 Ghz, 3GB RAM.

Было проведено две группы экспериментов, первая из которых рассматривала вопрос влияния размера таблицы фактов на соотношение производительностей выбранных двух методов. Вторая группа экспериментов была связана с селективностями операций соединения. В данной работе мы определяем селективность операции соединения таблицы фактов с таблицей измерений как отношение количества записей, полученных в результате соединения, к количеству записей в таблице фактов. В качестве измеряемого параметра рассматривали время выполнения запроса.

Наш запрос состоял из трех таблиц измерений, соединенных с таблицей фактов. Из таблиц измерений осуществлялась выборка на основе набора предикатов. При этом суммарный размер таблиц измерений были выбран таким, чтобы он помещался в память.

Были получены следующие результаты:

1) На таблицах фактов длины от нескольких мегабайт до 2 гигабайт (это был максимальный рассматривавшийся размер), в условиях одинаковой селективности всех таблиц измерений подход единственного оператора стабильно давал выигрыш в 3–4 %.
2) В условиях высокой селективности одной из таблиц измерений, мы получали выигрыш в 30 % в предложенном подходе, при условии, что высокоселективный оператор находился на вершине плана.

Заключение

В данной работе рассматривался вопрос о перспективах использования отдельного оператора для star join запросов с целью упрощения иерархии.

Данный оператор может рассматриваться как альтернатива использованию стоимостной оптимизации. Преимущества предложенного подхода заключается в простоте реализации, которая проявляется в том, что отпадает необходимость в стоимостной оптимизации при выполнении данных запросов. При этом, производительностью остается на высоком уровне. Проведенные эксперименты показали целесообразность использования оператора мультисоединения в условиях запросов рассматриваемого типа.

Литература

1. *К. Дж. Дэйт*. Введение в системы баз данных. 8-е изд. Вильямс, 2005.

2. *Смирнов К. К., Чернышев Г. А.* Сетевые и многопоточные аспекты архитектуры распределенных СУБД // Программные продукты и системы. № 1. 2011.

3. *C. Genzmer* et al. The SIGMOD 2010 Programming Contest: A Distributed Query Engine. SIGMOD Record, 39 (2), pp. 61–64, June 2010.

4. *G. Graefe.* 1994. Volcano, An Extensible and Parallel Query Evaluation System. IEEE Trans. on Knowl. and Data Eng. 6, 1 (February 1994), 120–135.

5. *G. Graefe.* 1993. Query evaluation techniques for large databases. ACM Comput. Surv. 25, 2 (June 1993), 73–169.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АРХИТЕКТУР ДЛЯ СИСТЕМ ВЕРТИКАЛЬНОГО ПОИСКА

Д. И. Качмар

Санкт-Петербургский государственный университет

Вертикальные поиски, или поиски по структурированным объектам с заранее определенной семантической моделью, позволяют решать задачи поиска по структурированной информации, например, с помощью них задача фильтрации объектов по необходимому атрибуту, или задача поиска объектов по нужному диапазону значений уже не представляется такой сложной, в отличии от такой же задачи в полнотекстовых поисковых системах.

Также стоит отметить, что в контексте работы со структурированной информацией вертикальный сервис имеет схожесть с базой данных, в связи с однозначностью трактовки запроса и наличием структурированных данных.

Важной проблемой, о которой необходимо помнить, является построение и проектирование поиска по структурированной информации и проблема масштабирования данной системы при увеличении количества данных и количества запросов к ней.

Осуществление поиска по структурированным данным находящимся в хранилище — довольно абстрактная задача, требующая гораздо большего углубления в проблему. Рассмотрим несколько способов поиска по данным, с требованием, что каждый из этих способов должен позволять осуществлять как фильтрацию данных по конкретному атрибуту и значению, так и диапазону значений.

- Реляционная база данных, и использование SQL запросов в качестве системы поиска
- Предварительное индексирование суррогатных сущностей, полученных из структурированных данных
- Распараллеливание запросов для поиска по непересекающимся массивам объектов.

Рассмотрим каждый из этих способов более подробно и проведем для каждого способа набор экспериментов, вычисляющих среднее время ответа в зависимости от количества данных и количества запросов, приходящих в систему.

Для проверок будем использовать одинаковый набор запросов и одни и те же объекты. В качестве шаблонов данных будем использовать реальные объявления о продаже автомобилей и реальные запросы, задаваемые пользователями к вертикальному поиску по объявлениям автомобилей — http://auto.yandex.ru.

Результаты экспериментов в таблицах представлены в миллисекундах, показано среднее время ответа через 5 минут после начала работы системы. Прочерки показывают, что через пять минут очередь запросов только продолжала расти и система переставала отвечать.

Использование реляционной базы данных и SQL запросов

В качестве реляционной базы данных будем использовать Oracle 11g, каждый объект хранится кортежем, все атрибуты, по которым осуществляем поиск, проиндексированы.

Количество данных в тыс. объектов / Количество запросов к системе	20	120	700
300	11	70	90
3000	32	78	–
30 000	122	–	–

Масштабирование поисковых возможностей данного подхода осуществляется чаще всего внутренними средствами реляционной базы данных, такими, например, как репликация базы на некоторый кластер серверов. Но, к сожалению, готовые базы данных, поддерживающие репликацию (такие, как Oracle), стоят довольно дорого, поэтому их использование и эксперименты с кластерной архитектурой довольно сложны в исполнении.

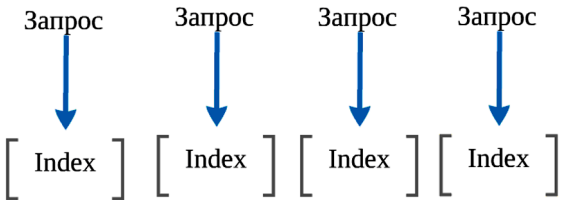
Применение заранее подготовленного индекса

В качестве системы индексации и поиска будем использовать Lucene версии 3.0.3, с проиндесированными суррогатными сущностями, полученными из объектов.

Количество данных в тыс. объектов / Количество запросов к системе	20	120	700
300	7	12	56
3000	15	45	111
30 000	40	–	–

Подход с заранее проиндексированной информации в среднем работает быстрее. Стоит заметить, что при данном подходе приходится платить возможностью управляемости данных, так как поддержка изменений поисковых атрибутов сущностей без переиндексации очень трудоемка.

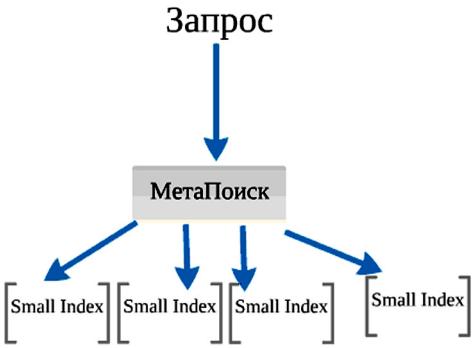
Также проведем эксперимент с простым поиском по проиндексированной информации, когда количество серверов в кластере 10, и запросы равномерно распределяются между серверами.



Количество данных в тыс. объектов/ Количество запросов к системе	20	120	700
300	1	8	17
3000	3	10	19
30 000	5	13	23

Заметим, что масштабирование по количеству запросов практически линейно, чего и стоило ожидать. Но данная система, к сожалению, так и не поддерживает увеличение количества данных в еще 10 раз, так как физически не хватает памяти для её размещения.

Шардирование данных и использование метапоисковой архитектуры



В качестве «базовых» поисков для метапоиска были использованы поиски с предварительной индексацией, описанные ранее. Сервер, обеспечивающий работу метапоиска был один, количество базовых поисков — 10.

Количество данных в тыс. объектов/ Количество запросов к системе	20	120	700
300	15	18	23
3000	17	20	30
30 000	21	28	55
300 000	48	55	109

Итак, в ходе экспериментов с метапоисковой архитектурой удалось добиться работы на большем количестве данных. Также к успешным достижениям стоит отнести, что результаты работы, полученные при небольшом количестве данных не сильно отличаются от результатов работы поиска с предварительной индексацией.

Использование каждого из подходов оправдано в зависимости от решаемых задач, которые определяются нагрузкой на поиск, объемом данных, по которым необходимо искать и управляемостью.

Литература

1. *S. Brin* and *L. Page*. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Stantford Univercity, 1998.

2. *Z. Nie, J. R. Wen, W. Y. Ma.* Object-level Vertical Search, Third Biennial Conference on Innovative Data. 2007.

3. http://www.oracle.com/

4. http://auto.yandex.ru

5. http://lucene.apache.org

6. *L. A. Barroso, J. Dean.* Google cluster architecture, Micro, IEEE, 2003.

.

АЛГОРИТМ ВЫДЕЛЕНИЯ ИМЕНОВАННЫХ СУЩНОСТЕЙ НА ОСНОВЕ ВИКИПЕДИИ

М. В. Ткаченко

Санкт-Петербургский государственный университет

Выделение в тексте названий фестивалей, людей, продуктов, биологических объектов, протеинов, и т. д. и определение их соответствующего типа называется задачей выделения именованных сущностей (named entity recognition). Термин именованная сущность был введен на Message Understanding Conference (MUC-6), под влиянием исследований в области извлечения информации. Было замечено, что изначальная разметка в тексте имен людей, названий населенных пунктов и организаций а так же некоторых числовых величин — есть необходимый элемент для последующих задач обработки текста. Интерес к проблеме не угасает и по сей день как с академической стороны, так и со стороны индустрии, а компонента выделения сущностей остается ключевым элементом для таких приложений как обнаружение событий (event detection), анализ мнений (sentiment analysis), создание вопросно-ответной системы (question-answering system) и т. д.

В самой общей форме на вход алгоритму подается текст, на выходе выдается информация о местоположении и типах выделенных имен. В данной работе рассматриваются четыре типа сущностей: имена людей (PERSON), названия населенных пунктов (LOCATION), названия организаций (ORGANIZATION), и тип разное (MISCELLANEOUS), который включает в себя именованные сущности, не попадающие в перечисленные три класса, это события, национальности и т. д.

Сложности в задаче возникают по ряду причин: большинство именованных сущностей встречаются в текстах редко и о них невозможно собрать информацию о словоупотреблении, также названия могут быть многозначны, например, в разных контекстах слово «Вашингтон» может означать фамилию человека, так и город в Соединенных Штатах.

Для решения задачи, как правило применяют методы машинного обучения [4]. Однако у данного подхода есть существенный минус: системы такого типа требуют огромного количества размеченных текстов, которые как правило создаются вручную специалистами в конкретной предметной области. Таким образом, аннотированные тексты имеют обыкновенно маленький объем и требуют дополнения коллекции с течением времени. Поэтому, чтобы повысить качество итоговой системы используют дополнительные источники информации, которые могут быть получены порой при минимальном участии человека, такие как словари сущностей, списки редко или часто встречаемых слов и т. д.

В качестве такого дополнительного ресурса автор предлагает использовать Википедию* (Wikipedia). Википедия — общедоступная многоязычная интернет-энциклопедия, содержащая более 18 млн. статей (около 3,5 на английском языке)**, которые постоянно добавляются и дорабатываются добровольцаим со всего мира.

Каждая статья Википедии описывает как правило один определенный объект, это может быть человек, организация, книга, событие. Структура страницы также как и структура самой энциклопедии содержит множество элементов, которые могут пригодиться при анализе текста. Статья может быть помечена одной или несколькими категориями, или отнесена к той или иной странице-списку. Оба этих элемента используются для облегчения навигации по Википедии. Например, статья «Bill Gates» помечена категорией «American philanthropists», и ссылка на эту старницу присутствует на странице «List of richest people». Как видно из примера категории могут быть полезны для определения класса страницы.

Следует также упомянуть страницы-перенаправления (redirect pages), которые отражают явление синонимии терминов. Например, на страницу «Bill Gates» можно перейти через перенаправление «William Gates, III».

Первый этап проделанной работы — классификация Википедии по классам именованных сущностей. В качестве набора классов была взята разметка в 16 классов (таблица 1). В качестве классов имен были взяты типы наиболее широко распространенные в научной литературе, чтобы полученная разметка могла послужить для дальнейших исследований в данной области. Отметим, что класс OTHER_PAGE не описывает класс имен и был введен для того, чтобы отсеять страницы, которые описывают обычные слова, или именованные сущности, слишком бедно представленные в Википедии.

Для тренировки и оценки качества классификатора было вручную размечено 2966 статей Википедии. Данные были разбиты на три набора в соотношениях 50/25/25, первый набор использовался для тренировки системы, второй для настройки параметров, третий для проверки качества системы. Сравнивались два метода классификации: наивный байесовский классификатор (NB) и метод опорных векторов (SVM) [2].

Базовый набор признаков состоял из токенов текста статьи и заголовка. Итоговый набор был дополнен токенами категорий, шаблонов, таблиц описывающих таксономию видов (taxobox), в вектор признаков также были включены заголовки страниц-списков, которые содержали данную статью. Из набора были исключены числа и стоп-слова. Полученный вектор взвешивался при помощи IDF и приводился к единичному L_2 нормой. Результаты работы классификаторов на тестовом множестве указаны в таблице 2.

---

* www.wikipedia.org

** http://meta.wikimedia.org/wiki/List_of_Wikipedias

Таблица 1

Классы именованных сущностей, примеры

Класс	Пример
PERSON	Barak Obama
GEO POLITICAL LOC	Paris
GEO REGION	Nile (River)
ASTRAL BODY	12 Victoria
FACILITY	Statue of Liberty
ORGANIZATION	Apple Computers
VEHICLE	BMW 30
WORK OF ART	Eugene Onegin
GAME	Go
SUBSTANCE	Deiodinase
SOFTWARE	Visual Studio 2010
EVENT	Kerrville Folk Festival
PLANT	Matricaria chamomilla
INSECT	Scarabaeus
ANIMAL	Megalodon
OTHER PAGE	sleep, ball, …

Таблица 2

Оценка качества классификации на тестовом наборе

	Базовая система		Итоговая система	
	NB	SVM	NB	SVM
Микро F-мера	65,5	71,0	71,4	79,8
Макро F-мера	54,5	58,1	71,1	72,0
Точность по ИС	68,6	80,2	82,6	94,5

В качестве оценки были использованы макро и микро F-меры по всем классам и, дополнительно, точность (A) по именованным сущностям. $A = T/(A - D_{\text{true\_other}} A - K)$, где T — правильно определенные именованные сущности, A — число страниц в коллекции, K — число именованных сущностей, определенных как OTHER_PAGE, $D_{\text{true\_other}}$ — число правильно определенных страниц класса OTHER_PAGE. Повышению последней характеристики было уделено особое внимание, так как она важна при создании точного словаря именованных сущностей.

Второй этап работы — создание системы выделения имен из текста. Для этого была использована модель случайных условных полей (Conditional Random Fields) [1].

На вход системе поступает текст разделенный по токенам, на выходе система выдает информацию об именованных сущностях. Рассмотрим пример размеченного системой текста:

President/O Obama/U-PER stepped/O carefully/O into/O the/O historic/O dispute/O between/O Turkey/U-LOC and/O Armenia/U-LOC…

Тэгом «O» размечаются слова отличные от именованных сущностей. Префиксы «U-», «B-», «I-», «L-» перед именем класса означают соответственно, что текущее слово является именованной сущностью, открывает, находится или завершает именованную сущность состоящую из нескольких слов.

Признаки используемые базовой и итоговой системой описаны в таблице 3. Отдельного упоминания требует алгоритм получения формы слова и признаки полученные с помощью Википедии.

Таблица 3

Признаки используемые для системы выделения сущностей*

Признаки
Слова (w_{-2}, w_{-1}, w_0, w_1, w_2, $w_{-1}:w_0$, $w_0:w_1$)
Части речи (t_{-1}, t_0, t_1, $t_{-1}:t_0$, $t_0:t_1$)
Формы слов (s_{-2}, s_{-1}, s_0, s_1, s_2, $s_{-1}:s_0$, $s_0:s_1$)
N-граммы текущего слова длинны от двух до шести символов
Префикс и суффикс текущего слова в 3 символа
Является ли слово началом предложения
*Есть ли слово в википедийном словаре (d_{-2}, d_{-1}, d_0, d_1, d_2, $d_{-1}:d_0$, $d_0:d_1$)

Форма слова получается из самого слова путем замены в нем всех алфавитных символов, цифр и всех остальных символов на символы «x» («X» для заглавных букв), «0» и «–» соответственно. Далее подпоследовательности одинаковых символов преобразуются в последовательность из двух. Например, слово «iPhone-3g» имеет форму «xXxx-0x».

На первом этапе работы был получен словарь содержащий следующие пары: (Заголовок статьи, Класс). Считаем, что заголовок страницы не отличим от заголовка перенаправления ведущего на эту страницу. Далее в качестве признаков для классификатора, используется изначальная разметка полученная следующим образом: все вхождения заголовков статей Википедии в тексте помечаются соответствующим классом. Для разметки использова-

* Индекс указывает позицию слова считая с текущего, через двоеточие обозначаются биграммы признаков. Через «*» помечен признак, который использовался только в итоговой системе.

лись только релевантные задаче типы: PERSON, GEO_POLITICAL_LOC, GEO_REGION, FACILITY, ORGANIZATION и EVENT.

Таблица 4 содержит результаты тестирования на эталонном тестовом множестве CoNLL-2003 [3]. Как видно из таблицы использование Википедии как дополнительного ресурса дает прирост F-меры по всем классам.

Одним из перспективных направлений работ в данной тематике может стать разрешение неоднозначности терминов при разметке текста классами Википедии т. е. разделение значений слов «Вашингтон» как город и «Вашингтон» как человек.

Таблица 4

Качество систем на тестовом множестве (F-мера)

Класс	Базовая система	Итоговая система
PER	86,6	90,3
LOC	86,7	89,7
ORG	77,5	82,3
MISC	76,4	77,8
ALL	82,7	86,2

Литература

1. *Andrew McCallum* and *Wei Li*. Early results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-Enhanced Lexicons. In Proceedings of CoNLL-2003. Edmonton, Canada, 2003. Pp. 188–191.

2. *Christopher D. Manning, Prabhakar Raghavan* and *Hinrich Schütze*. Introduction to Information Retrieval, Cambridge University Press. 2008.

3. *E. F. Tjong Kim Sang.* Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*. 2002. Pp. 155–158.

4. *Nadeau, David* and *Satoshi Sekine*. A survey of named entity recognition and classification. Linguisticae Investigationes. 2007. 30(1): 3–26.

АЛГОРИТМЫ УПРОЩЕНИЯ ГРАФА ДЕ БРЮИНА В ЗАДАЧЕ ГЕНОМНОГО АССЕМБЛИРОВАНИЯ

С. Ю. Нурк

Математико-механический факультет СПбГУ,
Лаборатория Алгоритмической Биологии СПбАУ РАН

В геноме закодирована практически вся информация о развитии и функционировании живых организмов. Именно поэтому задача определения последовательности нуклеотидов («букв») молекулы ДНК является одной из основных задач биоинформатики и имеет огромное количество практических применений: от разработки новых лекарств и диагностики генетических заболеваний до выявления эволюционных закономерностей. Несмотря на значительный прогресс за последние 30 лет, в ней остается огромное количество нерешенных проблем, прежде всего связанных с несовершенством применяемых алгоритмов.

Технологически, процесс разделяется на два этапа:

1) Биологический этап — секвенирование. На этом этапе большое количество копий одной молекулы ДНК разрезаются случайным образом на подстроки приблизительно равной длины, для концов которых определяются последовательности нуклеотидов. Множество полученных пар строк (их называют парными риды, paired reads) являются выходом секвенатора;
2) Вычислительный этап — ассемблирование (сборка). На этом этапе по набору парных ридов, полученных на первом этапе, с помощью специализированных алгоритмов и программных продуктов, производится попытка восстановить исходную последовательность генома.

С тех пор как наибольшее распространение получили технологии секвенирования под общим названием NGS (Next Generation Sequencing), которые, с одной стороны, значительно дешевле и быстрее аналогов, а с другой, имеют большее количество ошибок и меньшую длину ридов, в ассемблировании доминирует алгоритмический подход, основанный на графах де Брюина.

Граф де Брюина определяется следующим образом: для некоторого фиксированного k извлечем из ридов все подстроки длины k и поместим их в вершины графа. Две вершины v и w соединяются ребром, если в ридах встречается подстрока, длины $k + 1$, начинающаяся с подстроки, соответствующей v, и заканчивающаяся подстрокой, соответствующей w.

В модельной ситуации, исходная строка генома соответствует некоторому циклу в этом графе.

После построения графа, происходит его сжатие, при котором каждый «неразветвленный» участок графа заменяется одним «длинным» ребром.

Из-за различных типов ошибок секвенирования, помимо вершин и ребер, которые должны войти в итоговую сборку, в графе присутствует большое «побочных» элементов. Эти элементы могут сильно усложнить структуру графа и значительно повлиять на всю его последующую обработку. По этой причине необходимы способы их устранения.

Ситуация осложняется тем, что для получения качественной сборки, помимо самого сжатого графа де Брюина, необходима еще некоторая информация, ассоциированная с его элементами. Например, из исходных парных ридов, можно, для некоторых пар ребер, извлечь информацию о расстоянии, на котором они должны находиться в итоговой сборке.

Нами были предложены и реализованы эффективные алгоритмы устранения основных типов структурных аномалий графа («tips», «bulges», «erroneous connections»), которые, в соответствии с семантикой производимых операций, поддерживают логическую целостность дополнительной информации, используемой на последующих этапах сборки.

Теоретический анализ сложности алгоритмов затруднителен, но на реальных примерах, время их работы составляет $O(E \times \log E)$, где E — количество ребер в сжатом графе де Брюина, что позволяет им не являться «узким местом» при работе ассемблера.

Литература

1. Genome Reconstruction: A Puzzle with a Billion Pieces. P. Compeau, P. Pevzner, 2010.

2. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. D. R. Zerbino, E. Birney. Genome research, 2008.

3. An Eulerian path approach to DNA fragment assembly. Pevzner PA, Tang H., Waterman MS. Proc Natl Acad Sci USA. 2001 Aug 14;98(17): 9748–53.

4. Fragment assembly with double-barreled data. Pevzner PA, Tang H. Bioinformatics. 2001; 17 Suppl. 1: S. 225–33.

Кибернетика и робототехника

Фрадков
Александр Львович

д.т.н.
профессор кафедры теоретической кибернетики СПбГУ
заведующий лабораторией
«Управление сложными системами» ИПМаш РАН

Лучин
Роман Михайлович

ст.преподаватель кафедры теоретической кибернетики СПбГУ
научный сотрудник лаборатории теоретической кибернетики

АДАПТИВНОЕ УПРАВЛЕНИЕ РОБОТОМ-ВЕЛОСИПЕДОМ

А. А. Вакушкин

Санкт-Петербургский государственный университет

Каждый, кто садился за руль велосипеда, знает, что это средство передвижения само по себе неустойчиво. Чтобы не упасть, особенно при езде на небольшой скорости, необходимо постоянно крутить руль, таким образом, восстанавливая равновесие. Однако никто не задумывается, с какой скоростью и силой надо поворачивать — это получается само собой после нескольких поездок. Значит, человек приспосабливается, адаптируется. А как научить делать то же самое робота?

Задача управления роботом-велосипедистом уже стала классическим примером задачи управления в условиях неопределённости и возмущений. Цель данной работы — реализовать один из известных алгоритмов, получивший название «Полоска».

Пусть $y(t)$ — функция отклонения велосипеда от нормали (т. е. угол между нормалью и плоскостью велосипеда). Это значение можно измерить, например, гироскопом, что и было сделано в данной работе. Основная задача тогда — путём выбора управления сделать эту функцию ограниченной по модулю неким наперёд заданным числом: $|y(t)| < C_y$.

Пусть теперь $y(t)$ — функция времени, равная углу поворота руля, V — скорость движения велосипеда (считаем её постоянной), l, h — координаты центра тяжести, c — база велосипеда, g — ускорение силы тяжести. Также считаем, что управление производится с некоторой задержкой T. Тогда уравнение динамики велосипеда имеет следующий вид:

$$\ddot{y}(t) - \alpha_1 y(t) = \alpha_2 u(t-T) + \alpha_3 \dot{u}(t-T) + v(t), \tag{1}$$

где $\alpha_1 = \dfrac{g}{h}$, $\alpha_2 = \dfrac{V^2}{hc}$, $\alpha_3 = \dfrac{lV}{hc}$, $v(t)$ — возмущение, которое не превосходит по модулю некоторой заранее заданной константы $C_v > 0$: $|v(t)| < C_v$. Чтобы перейти к дискретному уравнению системы, будем считать, что $u(t)$ постоянно на отрезках времени длины δ. Обозначим за y_t значения функции при $t = 0, 1, \ldots$ Аналогично u_t — значение управления, v_t — возмущения. Тогда уравнение (1) примет вид:

$$y_t + a_1 y_{t-1} + a_2 y_{t-2} = b_1 u_{t-k} + b_2 u_{t-k-1} + v_t, \tag{2}$$

где $k = 1 + \dfrac{T}{\delta}$, $a_1 = -2ch\gamma$, $a_2 = 1$, $b_1 = \left[\alpha_2(ch\gamma - 1) + \alpha_3\sqrt{\alpha_1}sh\gamma\right]\alpha_1^{-1}$, $b_2 = \left[\alpha_2(ch\gamma - 1) - \alpha_3\sqrt{\alpha_1}sh\gamma\right]\alpha_1^{-1}$, $\gamma = \delta\sqrt{\alpha_1}$.

В новых обозначениях цель управления имеет вид

$$|y_t| \le C_y. \tag{3}$$

В работах В. А. Якубовича (см., например, [1]) предложены алгоритмы адаптивного управления, в том числе алгоритмы управления велосипедом. В одном из них («Полоска») управлении строится по следующей формуле:

$$u_t = u_{t-1}\tau_1 + y_t\tau_1 + y_{t-1}\tau_1, \tag{4}$$

где коэффициенты τ_i подбираются в процессе адаптации по следующей схеме:

- Если в момент времени t цель управления достигается, то коэффициенты τ_i остаются без изменений;
- Если же цель управления не достигается, то τ_i изменяются по следующим формулам:

$$\tau_1 = \tau_1 - u_{t-1}\xi;\ \tau_2 = \tau_2 - y_t\xi;\ \tau_3 = \tau_3 - y_{t-1}\xi, \tag{5}$$

где

$$\xi = \left(1 - \frac{C_v}{C_y}\right) \cdot \frac{y_{t+1}}{u_{t-1}^2 + u_t^2 + y_{t-1}^2 + y_t^2}.$$

Начальные данные можно выбрать произвольным образом. В книге [1] доказывается, что этот метод является конечно-сходящимся, т. е. обеспечивает достижение цели (3) в системе (2), (4), (5) за конечное число шагов. В работе [2] описана реализация этого алгоритма средствами LEGO Mindstorms NXT и пакета RWTH [3].

Приведённый алгоритм был реализован на языке программирования C, после чего программа была протестирована на роботе LEGO Mindstorms NXT с использованием средств [4, 5]. Конструкция робота отлична от [2]. В настоящее время производится отладка робота.

Литература

1. *Фомин В. Н., Фрадков А. Л., Якубович В. А.* Адаптивное управление динамическими объектами. М.: Наука, 1981.

2. *Селиванов А. А.* Адаптивное управление роботом-велосипедистом // Материалы докладов 12-й конференции молодых ученых по навигации и управлению движением. СПб.: ГНЦ РФ ОАО «Концерн «ЦНИИ «Электроприбор», 2010.

3. RWTH Mindstorms NXT Toolbox. http://www.mindstorms.rwth-aachen.de/

4. EcrobotC API. http://lejos-osek.sourceforge.net/ecrobot_c_api_frame.htm

5. Samples — http://lejos-osek.sourceforge.net/nxtway_gs.htm

РОБОТОТЕХНИЧЕСКАЯ СИСТЕМА ПЕРЕХВАТА ЦЕЛИ

Г. П. Облапенко, А. А. Семакова

Санкт-Петербургский государственный университет

В пионерских работах В. А. Якубовича по теории адаптивных систем [1, 2] были предложены модельные примеры адаптивных робототехнических систем, решающих задачи управления движением и преследованием. В настоящей работе описывается попытка реализации одной из теоретических схем, описанных в [1, 2].

Предположим, что имеется два объекта — перехватчик и цель. При этом цель обладает существенно большей мобильностью и скоростью перемещения, чем перехватчик. Также известно, что цель движется по заранее заданной траектории (окружность, эллипс, парабола и т. д.). Задача перехватчика заключается в том, чтобы отследить траекторию цели и на основе полученных данных осуществить ее перехват. Рассматривается частный случай задачи: цель движется по окружности, в центре которой находится перехватчик.

Реализация происходила с помощью конструктора Lego Mindstorms NXT и программного пакета RWTH для MATLAB [3, 4]. Роботы оснащены двумя моторами, к которым прикреплены передние колеса — с помощью них осуществляется перемещение цели и перехватчика; третье колесо, прикрепленное сзади обеспечивает устойчивость модели. Главное отличие цели от перехватчика заключается в том, что у последнего над управляющим блоком расположен третий мотор, к которому прикреплен ультразвуковой датчик расстояния (сонар); цель же должна представлять собой объект, который хорошо отражает ультразвук, поэтому над ее управляющим блоком мы прикрепили плоский металлический предмет.

Рассмотрим подробнее условия, относительно которых был реализован проект. Имеется радар с узконаправленным лучом, который может сканировать пространство (плоскость) в диапазоне 360°. Начальному положению радара отвечает угол поворота на 0°. Поворот против часовой стрелки естественно принять за увеличение значения угла, поворот по часовой стрелке — за уменьшение. До момента появления цели в зоне видимости, перехватчик производит сканирование пространства в заданном диапазоне. При обнаружении цели он начинает отслеживать ее траекторию — засекать отдельные точки, фиксируя расстояние до цели, время и угол поворота, на котором произошло отслеживание. При этом полагаем, что фиксирование следующей точки должно происходить не ранее, чем через 0.4 с после фиксирования предыдущей. Всего за время отслеживания фиксируется 14 точек. В силу технических особенностей конструктора Lego, диапазон обнаружения цели меняется от 15 до 50 см, а радиус окружности равен 35 см.

Сначала определяется скорость перехватчика. Для этого совершается один оборот колес вперед и равносильный ему поворот назад:

$$V = \frac{4 \cdot \pi \cdot R}{t}.$$

С помощью ультразвукового радара определяется среднее расстояние до цели, после чего вычисляется время, необходимое перехватчику, чтобы пересечь окружность: $t = V/R$. Рассмотрим подробнее вычисление угловой скорости цели. Фиксируем две точки и промежуток времени между ними. Если угол поворота второй меньше, чем первой, то

$$\omega = \frac{|\varphi_1 - \varphi_2|}{t},$$

в противном случае:

$$\omega = \frac{360 - |\varphi_1 - \varphi_2|}{t}.$$

Теперь необходимо определить время ожидания между фиксированием последней точки и перехватом. Существуют два возможных случая, а именно:

1. Если $\varphi - \omega \cdot t > 0$, то время ожидания равно

$$\frac{\varphi}{\omega} - t.$$

2. Если $-360 < \varphi - \omega \cdot t < 0$, то время ожидания равно:

$$\frac{360 + \varphi}{\omega} - t.$$

После фиксирования последней точки перехватчик ожидает вычисленное время и осуществляет перехват.

Практическая реализация алгоритма дала успешные результаты — в большинстве случаев перехватчик достигает цель.

В дальнейшем планируется уделить особое внимание минимизации погрешностей в уже реализованном алгоритме, а также разработать алгоритм, позволяющий перехватывать цель вне зависимости от ее траектории и начального положения перехватчика.

Литература

1. *Якубович В. А.* Адаптивные системы с многошаговыми целевыми условиями // ДАН СССР. 1968. Т. 183. № 2. С. 303–306.

2. *Якубович В. А.* Об одной задаче самообучения целесообразному поведению // Автоматика и Телемеханика. 1969. № 8. С. 119–139.

3. LEGO Mindstorms home page. http://mindstorms.lego.com

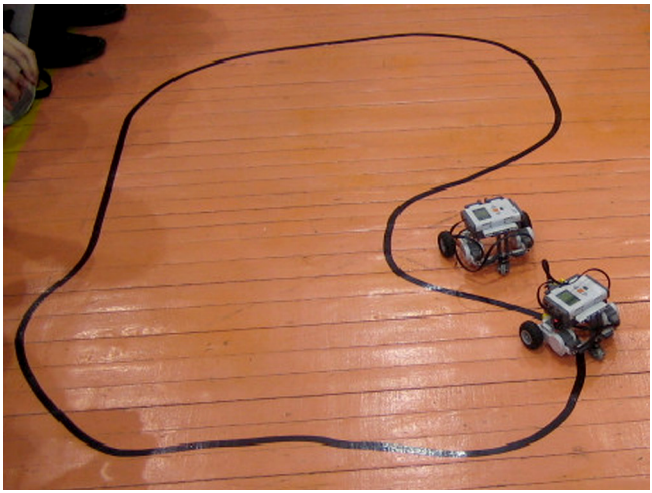
4. RWTH Toolbox. http://www.mindstorms.rwth-aachen.de

ОСОБЕННОСТИ РЕАЛИЗАЦИИ СИНХРОННОГО ДВИЖЕНИЯ МОБИЛЬНЫХ LEGO-РОБОТОВ

Е. В. Усик

Санкт-Петербургский государственный университет

В настоящий момент активно происходит автоматизация различных процессов, затрагивающих с каждым разом всё большее количество аспектов повседневной жизни. Всё чаще для выполнения некоторых задач, например уборка мусора, исследование труднодоступных территорий, проведение каких-либо работ в неблагоприятных для человека условиях, используются роботизированные системы (роботы). С увеличением темпов и областей использования роботизированных систем возникает необходимость координации их действий, при этом роботы должны выполнять такую координацию, по возможности, без участия человека. В частности, задачу координации действий между роботами можно рассматривать как задачу обеспечения их синхронного движения при некоторых условиях.



В настоящей работе выполнялось решение задачи синхронного движения мобильных роботов на основе схемы «Ведущий—Ведомый». Эта схема описывается таким образом:

1) в группе мобильных роботов выбирается робот — «Ведущий», которому известны все необходимые данные об окружающей обстановке;
2) остальные мобильные роботы называются «Ведомыми», они могут собирать информацию об окружающей обстановке с помощью установленных датчиков и обмениваться данными с «Ведущим».

При реализации вышеприведённого алгоритма в качестве программной платформы для LEGO-роботов [3] использовалась прошивка LEJOS NXJ (Java for LEGO Mindstorms) [1], которая представляет собой реализацию простейшей Java-машины, способной запускаться на LEGO NXT.

В настоящей работе использовались два LEGO-робота, один из них есть ведущий, а другой ведомый. Был использован так называемый принцип слепого, когда только один робот знает точно свою траекторию движения. Ведомый же получает нужные данные через протокол Bluetooth [2].

Мобильные LEGO-роботы поддерживают беспроводной способ обмена данными с помощью встроенного Bluetooth-модуля. Каждый робот одновременно может поддерживать беспроводное подключение с тремя устройствами, а в каждый момент времени производить обмен данными только с одним из них. Мобильный робот может устанавливать беспроводное подключение, как с другими мобильными роботами, так и с устройствами, которые могут быть настроены на использование команд из набора LEGO MINDSTORMS NXT Communication Protocol и поддерживающими Serial Port Profile (SPP). Для более выгодного представления настоящей работы, ведущий робот двигался по черной линии, с помощью светового датчика, что позволило его сделать более независимым.

Активация датчика происходит с помощью стандартного класса в lejos:

```
final LightSensor light = new LightSensor(SensorPort.S1);
```

Движение по линии, было реализовано с помощью ПД регулятора [4].

```
while (true){
            if(light.readValue() > 40){
                  Motor.A.stop();
                  Motor.C.stop();
                  Motor.A.forward();
                  else {
                  Motor.A.stop();
                  Motor.C.stop();
            Motor.C.forward();}
```

Ведомый робот, зная тип движения, также начинает осуществлять нужное действие.

```
while (true){
int n = dis.readInt();
            if(n==1){ Motor.A.stop();
                  Motor.C.stop();
                  Motor.A.forward();}
```

```
else { Motor.A.stop();
       Motor.C.stop();
Motor.C.forward();}
```

Таким образом, были изучены особенности управления мобильными Lego-роботами, а также различные спецификации протокола Bluetooth. И с учетом всех этих факторов была разработана и реализована основа для синхронизации мобильными роботами.

Lego-роботы — это отличная возможность видеть, как работают сложнейшие математические алгоритмы. Они позволяют малыми затратами разработать и оптимизировать новые технологии управления. Сегодняшний мир требует предельной экономии и наилучшей оптимизации расходами. Таким образом, все это в нынешних экономических условиях очень актуально и востребовано.

Литература

1. LEGO Mindstorms home page, http://mindstorms.lego.com

2. *Valera A., Weiss M., Valles M., Diez J. L.* Bluetooth-networked trajectory control or autonomous vehicles // 8th IFAC Symposium on Cost Oriented Automation. 2007. Vol. 8. Part 1.

3. *Filippov S., Fradkov A.* Cyber-Physical Laboratory Based on LEGO Mindstorms NXT — First Steps. 3rd IEEE Multiconference on Systems and Control. St. Petersburg, 8–10 July, 2009. Pp. 1236–1241.

4. *Бесекерский В. А., Попов Е. П.* Теория систем автоматического регулирования. СПб.: Профессия, 2004.

РАЗРАБОТКА АЛГОРИТМОВ УПРАВЛЕНИЯ АВТОНОМНЫМ ТРАНСПОРТНЫМ РОБОТОМ НА ОСНОВЕ МЕТОДА *I*-УНИВЕРСАЛЬНЫХ РЕГУЛЯТОРОВ

А. А. Мельников

Санкт-Петербургский государственный университет

Введение

В настоящее время очень широко распространены задачи автоматического управления различными системами, за функционирование которых в прошлом отвечал человек. Основной проблемой в процессах такого рода является наличие огромного числа факторов, которые влияют на происходящее. И, построение систем управления, которые могли бы учитывать все эти факторы, является одним из приоритетных направлений современной кибернетики и робототехники.

Одной из таких задач является задача движения некоторого транспортного средства вдоль заданного маршрута. Маршрут может быть задан различными способами: разметкой на поверхности дороги, с помощью магнитных маяков, системой глобального позиционирования и др. Основным фактором неопределенности в этой задаче является то, что мы не знаем наперед, по какому маршруту нужно двигаться, но в то же время необходимо гарантированно по нему пройти. Для того, чтобы гарантировать это требование, и будут применяться *I*-универсальные регуляторы. Все, что сделано в мире по этому направлению на данном этапе теоретически не гарантирует безошибочного движения вдоль маршрута.

Целью данной работы является разработка идеи, алгоритма управления транспортного робота в задаче движения вдоль траектории, которая задаётся в виде некой разметки на поверхности, по которой перемещается робот (к примеру, разметка на территории портовых складов, по которой движутся транспортные платформы). Ключевым отличием этой работы от других является идея применения теории *I*-универсальных регуляторов [1] к поставленной задаче. Преимуществом такого подхода является инвариантность системы управления относительно внешнего возмущения, т. е. цель управления (движение вдоль разметки) гарантированно достигается во всех случаях, когда характеристики разметки содержатся в некотором классе, обусловленном физическими параметрами транспортного средства. Ранее, такого применения этой теории не осуществлялось.

Методы исследования

Для разработки алгоритма в первую очередь необходимо построить математическую модель управляемого процесса. Будем считать, что линейная

скорость транспортного средства постоянна. Важным значением является отклонение транспортного средства от заданного маршрута. Требуется построить такую математическую модель, которая описывала бы изменение этой величины. Рассмотрим следующий рисунок:

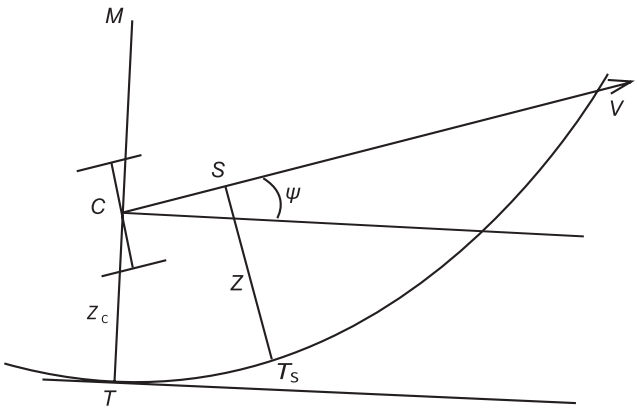


Рис. 1

Здесь M — центр касательной окружности к кривой в точке T, z_C — расстояние между C и T, точка S — расположение датчика и z — расстояние между точками S и T_S. В результате построения модели движения, имеем:

$$\begin{cases} \dot{z} = V\psi + l\omega, \\ \dot{\psi} = \omega - VR_T, \end{cases} \tag{1}$$

где R_T — кривизна траектории в точке T.

Применение теории I-универсальных регуляторов

Сформулируем цель управления. Необходимо построить такой регулятор, чтобы выполнялось следующее условие:

$$\lim_{t \to +\infty} z(t) = 0, \quad \forall R_T(t) \in \Theta, \tag{2}$$

где τ — класс возможных возмущений.

Для начала перейдем от модели в пространстве состояний к модели вида вход—выход:

$$A(\lambda) z = B(\lambda)\omega + F(\lambda) R_T,$$

где

$$A(\lambda) = \lambda^2, \quad B(\lambda) = l\lambda + V, \quad F(\lambda) = -V^2.$$

Теперь задача сведена к той форме, которая рассматривается в теории *I*-универсальных регуляторов.

Регулятор, на котором достигается цель управления, находится следующим образом:

$$D(\lambda)\omega = C(\lambda)z + G(\lambda)R_T,$$

где

[Sorry. Ignored `\begin{gather} ... \end{gather}`]

Тут r — произвольный многочлен, R — произвольный гурвицев многочлен.

Результаты

Для моделирования данного алгоритма управления использовалась среда MatLab.

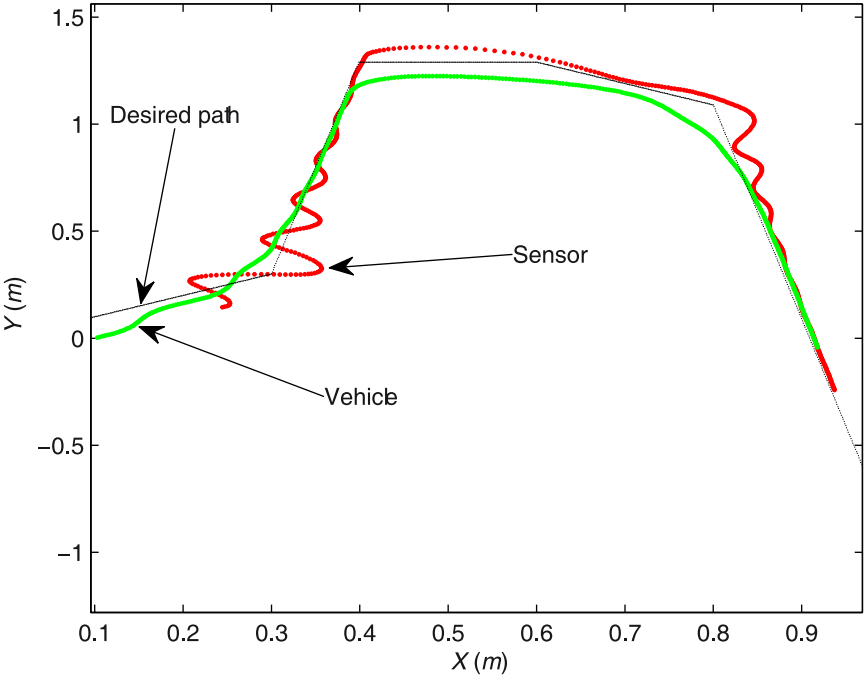


Рис. 2

С использованием пакета Control sistem toolbox произведены расчеты, необходимые для проведения моделирования данной системы управления. Результат можно увидеть на рис. 2.

Литература

1. *Проскурников А. В., Якубович В. А.* Задача об инвариантности системы управления // Доклады академии наук. 2003. Т. 389. № 6. С. 742–746.

2. MATLAB User's Guide. The Mathworks Inc. www.mathworks.com

3. LEGO Mindstorms home page. http://mindstorms.lego.com

4. *Мельников А. А., Лапко С. В., Сорока А. В.* Учебно-лабораторный комплекс для исследования систем управления движением автономных транспортных средств // 2-я международная конференция «Научно-техническое творчество — путь к обществу, основанному на знаниях». Сборник научных докладов. Москва, 2010.

5. *Gusev S. V., Paromtchik I. E., Makarov I. A. and Yakubovich V. A.* Adaptive motion control of nonholonomic vehicle // Proc. of the IEEE Int. Conf. on Robotics and Automation. Belgium, May 16–20, 1998.

6. nxtOSEK. lejos-osek.sourceforge.net

7. *Мельников А. А.* Алгоритмы управления движением автономных транспортных средств // 3-я мультиконференция по проблемам управления, 7-я научно-техническая конференция «Мехатроника, автоматизация, управление». Сборник материалов. СПб., 2010.

Прикладные вопросы теории алгебраического кодирования

Фёдоров
Андрей Рюрикович

генеральный директор компании Digital Design

ВНЕДРЕНИЕ ШИФРОВАНИЯ В СИСТЕМУ ХРАНЕНИЯ ДАННЫХ ВЫСОКОЙ ПРОИЗВОДИТЕЛЬНОСТИ

А. А. Овчинников
(E-mail: anton.ovchi2nikov@gmail.com)

Санкт-Петербургский Государственный Университет, математико-механический факультет, 3 курс

Целями работы являются исследование особенностей шифрования накопителей в системах хранения данных (СХД), а также поиск оптимального варианта встраивания механизма шифрования в конкретную систему на примере СХД Avrora.

Описание требований

Системы хранения данных являются специализированными средствами для достижения целей хранения и передачи информации. Они нуждаются в эффективных способах реализации и внедрения шифрования, но имеют свою специфику по сравнению с шифрованием каналов связи.

Но одно дело — изначально создавать систему хранения с поддержкой шифрования, и другое — правильно организовать интеграцию этого механизма с уже существующей и успешно функционирующей СХД. Попытка серьезного изменения архитектуры системы может повлечь существенную трату времени и материальных ресурсов.

Каким еще свойствам должна удовлетворять Система Хранения с шифрованием?

1. Конфиденциальность: данные на дисках должны храниться в зашифрованном виде, причем иметь к ней доступ может только ограниченный круг лиц.

К основным угрозам в данном контексте относятся кража (или изъятие) всей СХД, длительное наблюдение за «сырыми» данными, прослушивание канала от инициатора к СХД, а также попытки «внутреннего» проникновения (то есть попытка получить доступ с инициатора). В конечном итоге, все приведенные варианты не должны ставить под угрозу хранимую информацию.

2. Производительность: считывание и запись информации с/на диск должны выполняться быстро, причем вне зависимости от местоположения данных, то есть достижение произвольности доступа не должно отразиться на производительности.

3. Рациональность использования: способ шифрования не должен использовать слишком места на дисках**. Это** означает, что размер зашифрованных данных вместе с такой информацией, как ключи и необходимые

для алгоритмов шифрования другие данные не должны значительно превышать размер шифруемого открытого текста.

4. Прозрачность для пользователя.

Режим XTS

Как известно, при использовании блочного алгоритма шифрования над блоком данных произвольного размера необходимо использовать алгоритм в каком-то определенном режиме.

Самые известные режимы — ECB, CBC, CFB, OFB — являются непригодными для применения в Системах Хранения, так как имеют серьезные уязвимости в контексте нашей специфичной задачи. Примеры таких атак: watermark-атаки, copy&paste.

Из-за существенных уязвимостей «классических» режимов шифрования стало необходимым разработать специализированные режимы, предназначенные для использования в накопителях. Одним из первых таких режимов стал режим LRW (Liskov, Rivest, Wagner)[1]. Его создатели представили целый класс «настраиваемых шифров», которые используют специальный битовый вектор (размер которого обычно совпадает с размером блока для шифрования) — твик (tweak). Он служит скорее не для усиления криптостойкости, а для рандомизации выходного криптотекста при одинаковых открытых текстах. Успешной реализацией этой концепции стал режим XEX, предложенный в 2006 году, но сейчас более известной является его модификация: XTS (**X**EX-based **T**weaked encryption mode with ciphertext **S**tealing). Этот режим уже несколько лет успешно применяется для шифрования накопителей в таких программных продуктах, как Truecrypt, BestCrypt, dm-crypt и др.

СХД Avrora

В дальнейшем будем опираться на конкретный пример: СХД Avrora. Это российская СХД, ориентированная на высокую производительность и низкую стоимость за счет использования стандартных компонентов. Avrora позволяет создавать RAID уровней 0, 10 и 6, а эффективная реализация последнего является одним из главнейших достоинств этой СХД.

Возникла задача внедрения шифрования в Avrora, причем упор делался на RAID-6. Так как одним из важнейших преимуществ данной СХД является высокая производительность, главным моментом было не допустить деградации скорости.

В качестве алгоритма шифрования был выбран алгоритм AES (Rijndael), поскольку он является современным и хорошо проанализированным алгоритмом, а также из-за отличной аппаратной поддержки со стороны процессоров Intel: новый набор инструкций AES-NI показывает в среднем четырехкрат-

ный выигрыш в производительности по сравнению с программной реализацией. В скором времени поддержку этих инструкций будут осуществлять и процессоры AMD.

Режимом шифрования было решено сделать уже упомянутый XTS, причем областью действия ключа (keyscope) делается целый LUN (Logical Unit Number), что в какой-то степени аналогично шифрованию одного раздела в программах вроде Truecrypt, BestCrypt. Связано это с тем, что обычно пользователям СХД нужно обращаться к конкретному LUN. В каждом LUN — собственная адресация LBA (logical block address), что позволяет независимо вычислять номера сегментов (некоторое количество последовательных блоков, которые объединяются для проведения операций в режиме XTS) и блоков внутри сегментов.

Управление ключами

Данная тема сама по себе является довольно обширной, потому будет изложены только базовые концепции. В целом, подход к управлению ключами зависит от конкретной задачи и сферы применения СХД.

Как уже было упомянуто, каждый LUN шифруется отдельным ключом, то есть для доступа к каждому из шифрованных LUN необходимо пройти авторизацию.

Для этого необходимо обеспечить хранение специального метазаголовка для каждого LUN, в котором будет содержаться информация об используемом ключе, алгоритме, режиме шифрования, длине ключа и т.п., а также механизмы доступа к ним.

До недавнего времени не было единого стандарта, который бы описывал эти метаданные, и каждая программа использовала свой формат метазаголовка.

Но в 2009 году появилась первая версия спецификации LUKS (Linux Unified Key Setup), которая может вскоре стать единым стандартом для шифрования накопителей. LUKS предоставляет документированную, платформонезависимую спецификацию, описывающую формат мета-заголовка, а также процедуры установки, смены и удаления ключей. Также LUKS позволяет создавать до 8 паролей для одного шифруемого раздела. Если этого окажется мало, данная спецификация допускает масштабируемость.

Хранить метаданные можно либо в начале (в заголовке) каждого LUN, или же отвести под них отдельный LUN. Решено было использовать второй вариант, и на то есть несколько причин. Во-первых, это централизация хранения ключей, что позволяет быстро их изменять и удалять (это немаловажно при попытке нежелательного вскрытия системы). Во-вторых, не надо принимать дополнительные меры для предотвращения изменения заголовка.

Также к проблеме управления ключами можно отнести протокол аутентификации, то есть как поэтапно будет происходить общение пользователя

с системой. В работе описаны основные подходы в этом вопросе, но, в конце концов, главной задачей является именно организация некоего протокола, который во многом будет зависеть от сферы и условий применения СХД.

Уровень встраивания

Вкратце рассмотрим архитектуру СХД Аврора. Основной частью является модуль hyperraid_mod, который осуществляет поддержку RAID6: вычисление 2х синдромов (разных контрольных сумм, необходимых для восстановления информации в случае выхода из строя до 2х дисков). «Над» ним находится SCSI-драйвер scst_nr, который отвечает за отображение LUN на RAID и создает абстракцию для работы по разным аппаратным интерфейсам: FibreChannel, Infiniband… Выше находятся драйвера-связки с определенными физическими интерфейсами.

Можно выделить два принципиальных подхода при встраивании механизма шифрования:

1. Встраивание в RAID-Модуль (в нашем случае — hyperraid_mod), то есть шифрование будет происходить прямо перед записью или сразу после чтения. Важно учесть: шифрование должно происходить ДО вычисления синдромов, так как иначе возникает некоторая зависимость между зашифрованными данными, что позволяет применить разные техники криптоанализа. К тому же, достигается выигрыш в производительности: нужно меньше шифровать. А главным достоинством этого подхода является именно выигрыш в скорости, пока, правда, теоретический. Имеет место локальность обращения к памяти, Велика вероятность попадания в кэш нужных для последующих вычислений данных.
2. SCSI-модуль. Как только приходит SCSI-команда (Чтение, Запись), происходит извлечение данных (из команды при записи, с диска при чтении), которые затем шифруются/расшифровываются. Этот подход проще реализовать, так как нет необходимости писать реализацию под каждый RAID-уровень. Но есть и недостатки: по сути, происходит несколько «пробегов» по данным, первый раз — при шифровании, второй — при вычислении синдромов.

Результаты тестирования

Было проведено тестирование, в ходе которого проверялись зависимость потери производительности расчетов RAID-6 от размеров блоков страйпов, от количества дисков, а также от места встраивания при наличии шифрования AES-XTS (ключ 256 бит). Испытания проводились на тестовой машине с процессором Intel Core i5 520UM с поддержкой набора инструкций AES-NI, операционная система Red Hat Enterprise Linux 6.0.

В первую очередь тестировалась производительность при расчете синдромов P и Q для RAID-6, так как новые, поступившие в СХД данные обязаны пройти через эту процедуру. Было реализовано два подхода: шифрование всего страйпа перед вычислением синдромов и шифрование блока прямо перед вычислением синдромов. Результаты оказались интересными: ожидалось, что второй вариант покажет более высокую скорость (по той же причине, почему более эффективным считается шифрование на RAID-уровне, а именно из-за локальности обращения), но на деле выигрыш получался очень незначительным, а в некоторых случаях более эффективным был первый подход. Произошло это, скорее всего, из-за оптимизаций со стороны кэша. Стоит также учитывать, что тестирование проводилось в user-mode, в то время как реальные драйвера будут работать в kernel-mode.

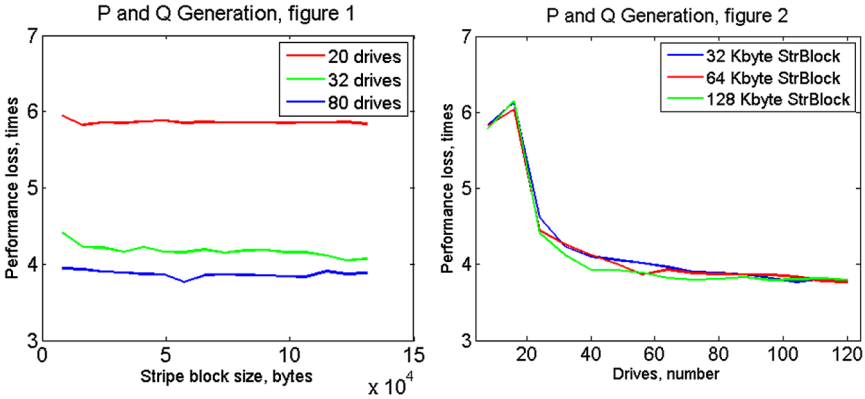
Из-за незначительности различий, будут использоваться результаты для первого подхода.

Были протестированы: размеры блоков страйпов — с 4 Кб по 128 Кб, количество дисков — от 4 до 120, замеры проводились в тиках (tick) процессора.

Тесты продемонстрировали следующее:

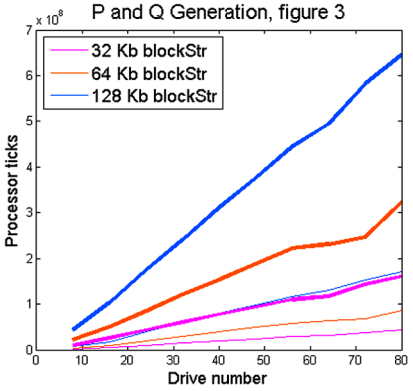
На падение скорости практически не влияет размер блоков страйпов, что хорошо видно из графиков 1 и 2, где падение показано «в разах».

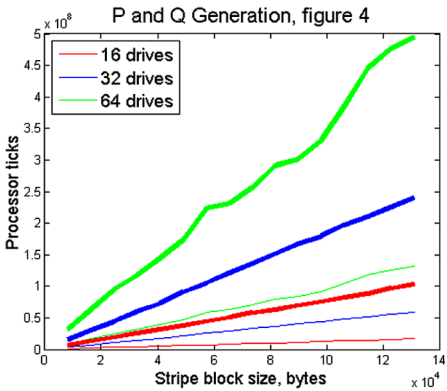
Можно видеть, что в зависимости от количества дисков падение производительности может быть различным.



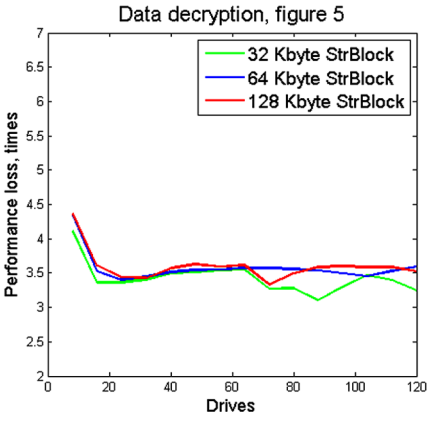
Когда дисков немного (до 20), вычисление синдромов происходит до 6 раз медленнее. При увеличении количества дисков это число уменьшается примерно до 3,7. Связано это с тем, что два сравниваемых процесса с определенными допущениями имеют вид линейных функций вида $(ax + b)$ и $(cx + d)$, у которых $(a \cdot d)! = (b \cdot c)$. То есть графики этих двух функций не пересекаются в нуле, что видно на графике 3: там показаны абсолютные значения вычисления синдромов с шифрованием (более объемные линии)

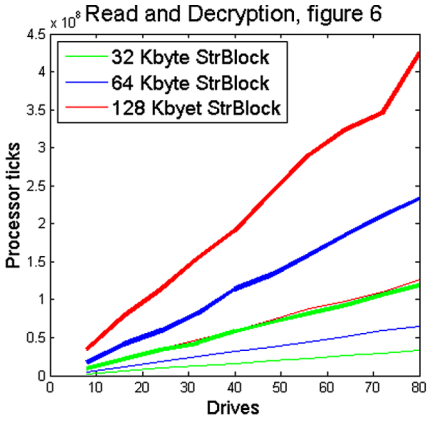
и без при фиксированных размерах блоков страйпов. В итоге мы получаем приближенную гиперболу на графике 2.





Также было протестировано расшифровывание, которое сравнивалось с обычным чтением. Результаты получились похожие, что видно из графика 5. Наблюдается снижение (незначительное, правда) коэффициента падения производительности при увеличении количества дисков. На графике 6 приведены абсолютные значения.





Таким образом, тестирование показало, что использованное шифрование не должно существенно сказаться на производительности. Вычисление синдромов является довольно незначительной в плане влияния на быстродействие операцией, самым узким местом в СХД обычно являются сами накопители, и многое зависит от алгоритмов кэширования, реализованных там. Поэтому увеличение времени при вычислении синдромов даже на порядок не должно показать серьезное падение производительности. Больше

на данный момент сказать трудно, решающим станет финальное тестирование на реальной системе, так как есть много факторов, которые трудно учитывать на стадии моделирования: опять же поведение кэша, скорость чтения/записи на диск и т. д.

Заключение и выводы

Были проанализированы возможные варианты внедрения шифрования данных в СХД Avrora.

Тестирование показало, что операции (рас)шифрования при эффективной реализации не окажут существенного влияния на производительность всей системы. Также тестирование продемонстрировало незначительное отклонение производительности при двух методах внедрения. Если эти ожидания действительно оправдаются, то встраивание можно будет проводить там, где это проще сделать — на уровне SCSI.

Следующим этапом работы будет внедрение и тестирование на реальной рабочей системе.

Литература

1. IEEE 1619P Standard “Narrow Block Encryption” (2007).
2. *Bruce Schneier*. “Applied Cryptography” (1996).
3. *Liskov, Rivest, Wagner*. “Tweakable Block Ciphers” (Advances in Cryptography, 2002).
4. “Intel Advanced Encryption Standard (AES) Instructions Set” White Paper (January 2010).
5. Linux Unified Key Setup (LUKS): http://code.google.com/p/cryptsetup/
6. Компания Avroraid http://avroraid.com/

АЛГОРИТМЫ, ИСПРАВЛЯЮЩИЕ ОШИБКИ, В СХД УРОВНЯ RAID 6

А. С. Солозобов

Санкт-Петербургский государственный университет

В настоящей статье рассматриваются оптимизации, применимые в СХД с программно реализованным RAID-6, который позволяет строить СХД на «обычных» компьютерных комплектующих без использования специальных RAID контроллеров.

Общая модель построения RAID 6

Для того, чтобы иметь возможность восстанавливать данные в системе при выходе из строя сразу двух физических носителей, необходимо для каждого стрипа данных хранить два независимых синдрома. Мы рассмотрим алгоритм организации RAID-6, основанный на подсчёте синдрома чётности и полиномиального синдрома Рида—Соломона.

Каждый диск разделяется на блоки по 128 бит и каждый блок рассматривается как набор коэффициентов из GF(2) многочлена 127 степени. Обычным образом вводятся операции сложения и умножения многочленов. Так как коэффициенты многочленов из GF(2), то операция сложения будет эквивалентна операции XOR. Полученное кольцо многочленов факторизуется по неприводимому многочлену 128-ой степени и получается поле многочленов, в котором каждый ненулевой элемент обратим.

Под многочленами стрипа будем понимать многочлены, соответствующие блокам указанного стрипа, и все операции, производимые над ними, будут производиться в указанном поле.

Первый синдром получается как сумма многочленов в стрипе. Для получения второго синдрома производится суммирование многочленов стрипа, домноженных каждый на x^i, где i — номер диска, которому соответствует блок в стрипе. В данном случае от того, с какого числа мы будем нумеровать диски, принцип работы алгоритма не меняется, но, чтобы не выполнять лишних операций умножения многочленов, лучше нумеровать их начиная с нуля.

Алгоритмические оптимизации

Так как RAID у нас программный, то, помимо выбора алгоритмически хороших решений, нам надо научиться наиболее эффективно использовать те вычислительные ресурсы, которые нам предоставляет процессор, его кэш и оперативную память компьютера.

На этапе проектирования алгоритма очень важным является создание максимально линейного кода программы, не содержащего каких-либо ветвлений. Это даёт возможность оптимально использовать процесс конвейеризации обработки команд в процессоре.

Для ускорения операции нахождения остатка от деления многочлена делитель должен иметь старший член 128 степени, а остальные члены должны быть не старше 63 степени. Это позволяет проводить факторизацию, используя два умножения и два сложения многочленов. Мы будем использовать многочлен

$$x^{128}+x^7+x^2+x+1.$$

Многочлен 1

В общем случае восстановление в стрипе двух потерянных блоков B_a и B_b с номерами a и b соответственно по полиномиальному синдрому Q и синдрому чётности P сводится к решению системы двух алгебраических уравнений:

Система 2

$$\begin{cases} B_a = P + B_b; \\ B_b = \dfrac{P \cdot x^{-a} + Q}{x^{b-a}+1}. \end{cases}$$

При её решении используется довольно медленная операция деления многочленов. Так как при инициализации системы мы уже знаем максимальное количество дисков в ней, то будет разумно предподсчитать все возможные знаменатели этой формулы. Поиск обратного многочлена рационально проводить алгоритмом Евклида.

Команды intrinsics и работа с кэшем

В современных процессорах существует многоуровневая система кэшей, которые заполняются из памяти целыми кэш-линиями. Таким образом, если мы начали восстанавливать систему после сбоя одного или двух дисков, то при проходе по страйпу мы на самом деле получим в кэше данные сразу из нескольких страйпов за счёт того, что размер кэш-линии в несколько раз больше, чем размер нашего блока, и у современных процессоров составляет 64 байта (в четыре раза больше размера одного блока в 128 бит). Но чтобы этот процесс работал, блоки должны быть выровнены на границу 64 байт и кратны размеру кэш-линии, иначе они будут кэшироваться кусками, что приведёт к снижению производительности.

Также современные процессоры поддерживают массу расширений набора команд, в частности, наборы SSE, SSE2, SSE4 и CLMUL. Эти команды очень удобны в использовании и при написании кода на языке C вызываются так называемыми intrinsics командами, которые, как утверждает компания

Intel, позволяют производить дополнительные оптимизации на этапе компиляции кода программы.

Мы выяснили, что при восстановлении одного или нескольких дисков, когда уже произведён проход по одному страйпу и запрошены данные с соответствующих дисков, мы можем сравнительно быстро провести обработку ещё нескольких страйпов, оказавшихся в кэше. Так как мы знаем, какие страйпы собираемся обрабатывать дальше, то за время текущей обработки хорошо было бы подгрузить из памяти в кэш необходимые данные следующих страйпов. Для этого есть специальная команда _mm_prefetch из набора SSE, которая позволяет подгрузить одну кэш-линию начиная с указанного адреса в ближайший к процессору кэш. Причём помимо адреса у этой команды есть ещё один параметр, указывающий, какого рода операции мы намерены дальше совершать с этими данными. В частности, от него зависит, будут ли эти данные при вытеснении из кэша помещены в более дальний кэш или сразу в память, не «пачкая» его (не вытесняя из него уже лежащие там актуальные данные).

Когда же мы занимаемся восстановлением системы, то при записи восстановленных по синдромам данных на диск они будут кэшироваться, тем самым «пачкая» кэш. Понятно, что пользователю данные одного или двух дисков вряд ли нужны и поэтому кэшировать их не стоит. В расширении SSE есть команды _mm_stream_pi и _mm_stream_ps, которые позволяют записывать данные «мимо» процессорного кэша. Понятно, что не стоит применять эти команды при пересчёте синдромов страйпа, связанном с изменением данных. Синдромы часто используемых страйпов как раз стоит кэшировать. И здесь уже встаёт решаемый эмпирически очень важный вопрос выбора оптимального количества дисков в системе и размера дискового страйпа, чтобы его данные и синдромы хорошо ложились в кэш.

Умножение многочленов

Очень важным оказывается использование команды PCLMULQDQ из набора CLMUL, которая позволяет перемножать 64-битные половины 128-битных переменных типа _m128i (именно поэтому был выбран соответствующий размер дискового блока и соответствующий многочлен факторизации). При этом умножение в процессоре проходит с выключенными цепями переноса, т.е. при переполнении разряда он сбрасывается на ноль без увеличения ближайшего старшего разряда. Эта команда полностью соответствует описанному в нашем поле умножению и тем самым позволяет гораздо быстрее его выполнять.

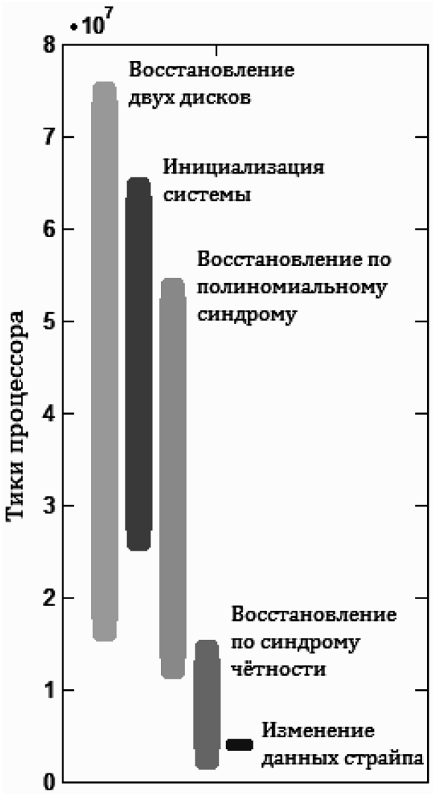
Замеры производительности алгоритма

Было проведено тестирование производительности алгоритма, написанного с учётом всех вышеперечисленных оптимизаций.

Наиболее часто используемая операция записи данных в страйп является наименее трудоёмкой и всегда выполняется за время, не зависящее от количества дисков в системе.

Операции восстановления диска в страйпе по синдрому Q оказывается сопоставимой по трудоёмкости с операцией генерации синдромов для страйпа в целом, хотя алгоритмы их работы очень похожи и отличаются тем, что первый из них дополнительно решает систему (2).

Восстановление двух дисков в страйпе оказывается на 5 % дольше, чем суммарная восстановление диска в страйпе отдельно по синдрому чётности и отдельно по полиномиальному синдрому, хотя первому из них опять же приходится дополнительно решать систему (1).



Диапазоны количества тиков при различных операциях в системах с 8–32 дисками по 128 KB

Направления оптимизации

Наиболее целесообразным направлением оптимизации алгоритма кажутся:

1) Ускорение самой часто используемой операции при полиномиальном кодировании — операции перемножения многочленов.
2) Введение более быстрой операции последовательной записи нескольких блоков на один стрип, чем запись соответствующих блоков по отдельности.
3) Переход к использованию неприводимого многочлена 64-ой степени вместо многочлена (1), что может привести к ещё более быстрому выполнению операций с многочленами.

Литература

1. *Фёдоров А. Р.* Задача восстановления утерянных дисков в массиве с использованием арифметики конечных колец.

2. Intel® 64 and IA-32 Architectures Software Developer's Manual.

http://www.intel.com/products/processor/manuals/

3. Intel® 64 and IA-32 Architectures Optimization Reference Manual.

http://www.intel.com/products/processor/manuals/

4. Intel® Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode.

http://www.intel.com/products/processor/manuals/

5. Intel® 64 and IA-32 Architectures Application Note TLBs, Paging-Structure Caches, and Their Invalidation.

http://www.intel.com/products/processor/manuals/

6. Porting and Optimizing Multimedia Codecs for AMD64 architecture on Microsoft® Windows®.

http://download.microsoft.com/download/5/b/5/5b5bec17-ea71-4653-9539-204a672f11cf/MMCodec_amd64.doc

АЛГОРИТМЫ КЭШИРОВАНИЯ В СХД

Р. Колобов

Санкт-Петербургский Государственный Университет

В основе современных систем хранения данных по-прежнему лежат RAID-массивы на базе жёстких дисков, которые не могут обеспечить достаточной пропускной способности без применения кэширования. Поэтому задача подбора алгоритма кэширования остаётся важным вопросом при проектировании СХД. При разработке системы также необходимо учитывать область её применения; примерами могут служить два противоположных по требованиям приложения: базы данных и обработка видео. Соответственно, для оценки производительности системы необходимо проанализировать паттерны запросов предполагаемой области использования (пример подобного анализа см. в [1]) и протестировать на них кэш.

В случае видеозаписи обработка данных идёт параллельными потоками, и паттерн достаточно прост: последовательное чтение или запись. В случае же с базами данных во-первых, доля обращений по случайным (или псевдослучайным — например, записи в таблице, равномерно расположенные через большие промежутки) адресам высока, а во-вторых, при транзакциях обращение к одному и тому же блоку данных может происходит как минимум дважды, с временной задержкой (до 3–5 секунд), в течение которой при неправильной политике кэша данные могут быть вытолкнуты как подпадающие под паттерн произвольного доступа (подробное обсуждение проблемы correlated references см. в [2]).

Обработка случайных запросов и борьба с блокировками при параллельной обработке достаточно хорошо описаны в [3] и предшествующих ей работах. Наиболее часто встречающиеся решения — это выстраивание блоков в LRU-очереди, разделение данных на последовательные и произвольные и группировка запросов на запись в кэше с целью последующего выталкивания на диск большими порциями. Однако, комбинирование различных оптимизаций может давать иногда неожиданные эффекты, поэтому любой хорошо проработанный теоретически алгоритм при встраивании в систему нуждается в комплексном тестировании.

В качестве примера для тестирования и поиска оптимизаций была выбрана СХД Avrora от Avroraid. Тестирование проводилось с помощью Iometer, который с помощью гибкой системы настроек позволяет воспроизводить паттерны доступа различных приложений.

В первую очередь были протестированы последовательные паттерны, на которых производительность системы оказалась незначительно хуже производительности RAM, в которой размещался её кэш, что обеспечивается хорошим алгоритмом read ahead и write back. Однако, при внесении доли произвольных запросов оказалось, что производительность системы начинает быстро падать (см. рис. 1).

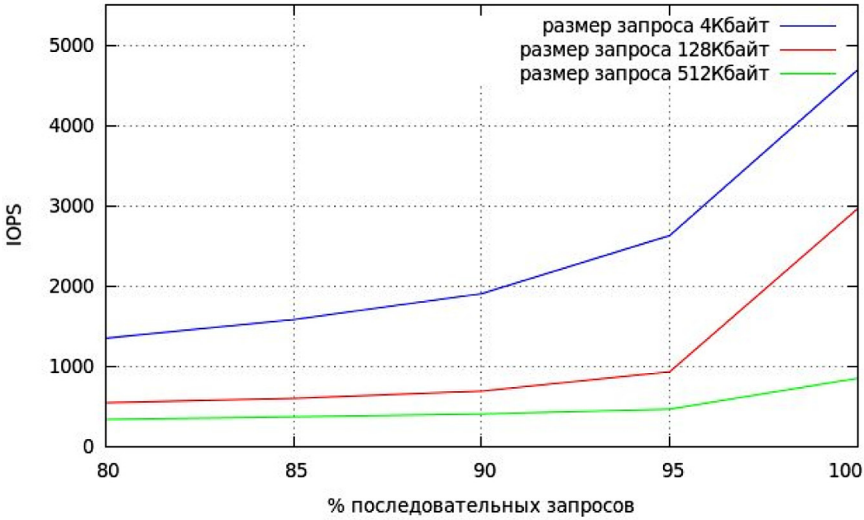


Рис. 1. Зависимость производительности от доли случайных запросов в тесте

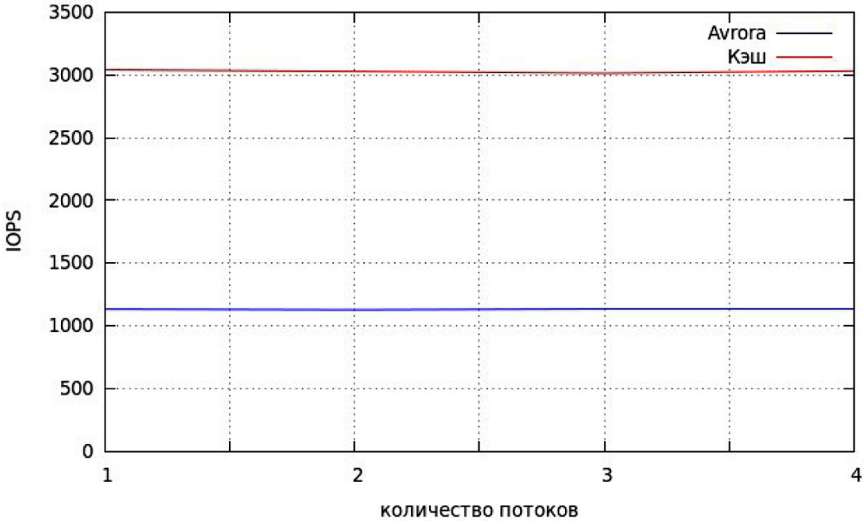


Рис. 2. Производительность при разном количестве потоков при произвольном доступе

Было сделано предположение, что это происходит из-за наличия общей очереди для последовательных и параллельных запросов, в результате чего произвольные данные могут засорять кэш и препятствовать его наполнению последовательными данными.

При многопоточном чтении со 100 % произвольным паттерном система работала всего лишь в 3 раза медленнее кэша (см. рис. 2), что в совокупности с данными о последовательных паттернах позволило сделать вывод о достаточной оптимизации для однородных последовательностей запросов.

Однако, на этих же тестах очередь write back заполнялась до предела (ср. рис. 3 и 4), установленного настройками по умолчанию, и приходилось вручную их менять для ускорения работы. В этом случае, возможно, пригодилось бы наличие адаптивности (соответствующие техники описаны в [3, 4]).

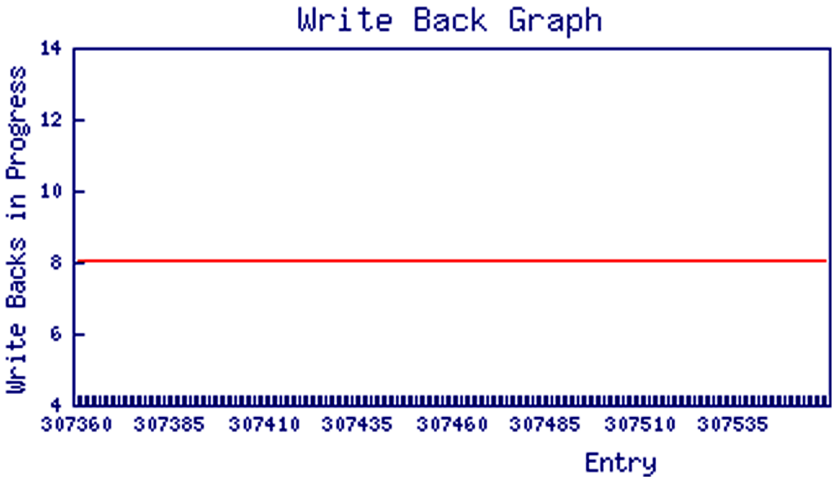


Рис. 3. Заполнение очереди write back

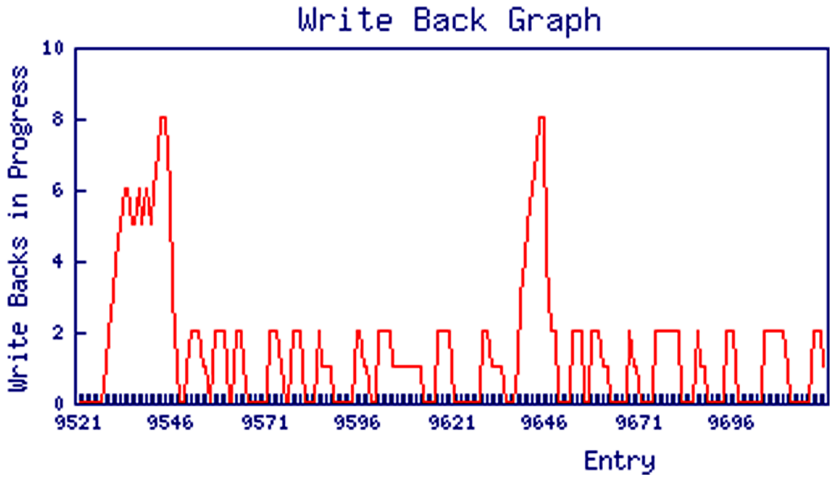


Рис. 4. Наполнение очереди write back при неинтенсивной нагрузке

Также при многопоточной работе производительность чтения была иногда ниже ожидаемой, так как блокировки в системе устанавливаются не зависимо от типа запроса, а многопоточное чтение можно осуществлять и без блокировок.

По итогам замеров были сделаны выводы, что ускорить работу системы могут:

1. более строгий механизм разделения последовательных и параллельных запросов;
2. защита от блокировки запросов на чтение при работе в несколько потоков;
3. добавление алгоритму адаптивности.

Поскольку принципы, использованные при разработке кэша Avrora, используются и в других системах (например, упорядочивание по схеме LRU, весьма часто применяющееся в кэшах СХД), можно сделать вывод, что указанные выше оптимизации можно успешно применить (если они ещё не применены) для ускорения работы различных СХД, а не только к исследованной в данной работе Avrora.

Литература

1. *Asit Dan, Philip S. Yu, Jen-Yao Chung.* Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability (1994).

2. *Elizabeth J. O'Neil, Patrick E. O'Neil.* The LRU-K Page Replacement Algorithm For Database Disk Buffering (1993).

3. *Binny S. Gill, Biplob Debnath, Michael Ko, Wendy Belluomini.* STOW: A Spatially and Temporally Optimized Write Caching Algorithm (FAST 2009).

4. *Nimrod Megiddo, Dharmendra S. Modha.* ARC: A Self-Tuning, Low Overhead Replacement Cache (FAST 2003).

Мультиагентные технологии и их приложения в информатике

Тимофеев
Адиль Васильевич

д.т.н.
профессор
заведующий лабораторией информационных технологий
в управлении и робототехнике СПИИРАН

МУЛЬТИАГЕНТНЫЕ ТЕХНОЛОГИИ И ИХ ПРИЛОЖЕНИЯ В РОБОТОТЕХНИКЕ И НЕЙРОИНФОРМАТИКЕ

А. В. Тимофеев (E-mail: tav@iias.spb.su)

Санкт-Петербургский институт информатики и автоматизации, Санкт-Петербургский Государственный университет

1. Интеграция и интеллектуализация систем управления и навигации роботов

В последние годы важное значение приобретает развитие принципов интеллектуального управления роботами, интеграции систем управления движением, навигации и функциональной диагностики, а также создание теории мультиагентных робототехнических систем (МАРС) на базе информационных и телекоммуникационных технологий.

Принципы интеллектуального управления роботами основываются на использовании элементов искусственного интеллекта (ИИ) [1–5]:

- моделирование среды (препятствий) на основе локальной сенсорной информации или средств виртуальной реальности;
- планирование кратчайших маршрутов и траекторий движения в среде с известными или неизвестными препятствиями;
- распознавание образов и функциональная диагностика.

Одним из наиболее эффективных средств распараллеливания процессов обработки информации и управления являются нейронные сети (НС). Предложенные в [6–12] методы интеллектуального и нейросетевого управления нашли применение в различных областях.

2. Мультиагентные робототехнические системы

На практике роботы обычно интегрируются в мультиагентные робототехнические системы (МАРС) для совместного достижения общих целей и решения сложных задач. При этом возникают новые проблемы группового управления и коммуникации, связанные с организацией «коллективного» поведения роботов. Традиционно эти проблемы решались на основе централизованного или децентрализованного управления.

Компромиссный подход заключается в сетевом управлении и групповой навигации МАРС. В этом случае роботы рассматриваются как интеллектуальные мехатронные агенты МАРС, локальные БД и БЗ и могут оперативно обмениваться информацией [4, 5].

Принцип действия МАРС основывается на декомпозиции общей задачи на ряд локальных задач, возлагаемых на агентов-роботов, распределении этих задач между ними, планировании коллективного поведения агентов, ко-

ординации взаимодействия агентов на основе кооперации, реконфигурации, коммуникации и разрешения конфликтных ситуаций [4, 5].

Задачи стратегического уровня обычно возлагаются на специального агента-координатора, а задачи тактического уровня параллельно решаются роботами как мехатронными агентами. В результате мультиагентного управления и групповой навигации значительно увеличивается надёжность, адаптивность и быстродействие МАРС в изменяющейся среде с препятствиями. В последние годы были разработаны основы теории управления МАРС и информационная технология мультиагентной бесконфликтной навигации коллектива роботов, функционирующих в изменяющейся среде с препятствиями [4–6]. В частности, были предложены методы групповой навигации и управления коллективным движением роботов-агентов и разрешения конфликтов (предотвращения столкновений).

3. Мультиагентное управление информационными потоками в МАРС

Совершенствование МАРС связано с развитием методологии автоматизации, адаптации и интеллектуализации систем сетевого управления информационными потоками на базе динамических моделей телекоммуникационных сетей (ТКС) как сложных объектов управления с переменной структурой, методов оптимизации процессов маршрутизации информационных потоков и принципов адаптивного и интеллектуального управления трафиком с использованием мультиагентных технологий и протоколов нового поколения (IPv6 и др.). На этом новом пути возможен как учет реальной динамики ТКС, т. е. фактического состояния или изменения структуры (топологии) и параметров (весов каналов связи) ТКС в реальном времени, так и адаптация к различным факторам неопределенности на основе мониторинга и функциональной диагностики ТКС [4–6].

Основные функции обработки информации, самоорганизации и управления информационными потоками по запросам внешних агентов распределяются между внутренними агентами, роль которых выполняют сетевые или нейросетевые агенты. Архитектура этих внутренних сетевых агентов аналогична архитектуре ТКС. В этом проявляется фрактальность сетевых и нейросетевых агентов по отношению к ТКС и ее подсетям.

Нейросетевые агенты предназначены прежде всего для параллельной передачи и обработки сложных мультимедийных сигналов и образов (2D- или 3D-изображения и т. п.). В результате обучения по множеству прецедентов из обучающей БД осуществляется настройка архитектуры (топологии сетевых нейронов) и параметров (синаптических весов) нейронных агентов на решаемую задачу [4–6, 10–12]. В последнее время разработаны модели нейросетевых агентов для адаптивной маршрутизации (агенты-маршрутизаторы) и автоматической классификации WEB-сайтов на естественном (русском)

языке (агенты-классификаторы). Программная реализация и имитационное моделирование этих агентов свидетельствует об их эффективности и преимуществах по отношению к традиционным информационным технологиям.

Важное значение для эффективного распознавания образов и диагностики состояний в реальном времени представляют гетерогенные полиномиальные (ПНС) с самоорганизующейся архитектурой, которые аккумулируют «нейрообразы» и решающие (классифицирующие и идентифицирующие) правила и обеспечивают массовый параллелизм, хорошую экстраполяцию и высокое быстродействие при принятии оптимальных или субоптимальных решений [6–10].

Коллективное использование гетерогенных ПНС в качестве НС-агентов позволяет дополнительно распараллелить и распределить между локальными НС-агентами процессы решения сложных (глобальных) задач распознавания образов, анализа изображений и сцен, расширенной диагностики состояний и адаптивной маршрутизации информационных потоков.

4. Логические методы и нейросетевые технологии распознавания сложных изображений и сцен

Использование логических описаний образов позволяет свести задачи распознавания сложных изображений и интеллектуального анализа сцен к поиску логического вывода в исчислении предикатов. При этом в МАРС решаются три типа задач распознавания образов [9–10]:

- идентификация объекта из заданного класса на сложном изображении или 3D-сцене;
- классификация всех объектов на сложном изображении или 3D-сцене;
- анализ и локализация всех объектов на сложном изображении или 3D-сцене.

Для увеличения эффективности поиска предлагается иерархический способ формирования таких логических описаний. Нейросетевое представление иерархических логических описаний образов и решающих правил обеспечивает массовый параллелизм и высокое быстродействие при распознавании сложных изображений и интеллектуальном анализе сцен [9–12].

Предложенные логические методы и нейросетевые технологии успешно использовались для интеллектуального зрения роботов в МАРС при решении следующих прикладных задач [2, 3, 20, 25]:

- оценка потенциальной террористической опасности людей и предметов на вокзалах;
- распознавание и оценка террористической опасности транспортных средств, движущихся вблизи газопроводов [9–11].

Работа выполнена при поддержке грантов РФФИ № 09–08–00767-а, РФФИ № 08-08-12183-офи и РФФИ-ГФЕН Китая 10-08-91159 и Проекта № 1.6 Программы № 13 Президиума РАН.

Список литературы

1. *Тимофеев А. В.* Роботы и искусственный интеллект. М.: Наука, 1978. 192 с.

2. *Тимофеев А. В.* Адаптивные робототехнические комплексы. Л.: Машиностроение, 1988. 332 с.

3. *Timofeev A. V.* Intelligent Control Applied to Non-Linear Systems and Neural Networks with Adaptive Architecture // Journal of Intelligent Control, Neurocomputing and Fuzzy Logic. 1996. V. 1. No. 1. Pp. 1–18.

4. *Тимофеев А. В.* Мультиагентное и интеллектуальное управление сложными робототехническими системами // Юбилейный сборник «Теоретические основы и прикладные задачи интеллектуальных информационных технологий», посвящённый 275-летию РАН и 20-летию СПИИ РАН. СПб.: СПИИРАН, 1999. С. 71–81.

5. *Тимофеев А. В.* Методы высококачественного управления, интеллектуализации и функциональной диагностики автоматических систем. Часть I, Часть II. — Мехатроника, автоматизация, управление. 2003, № 5. 2004, № 2.

6. *Каляев А. В., Тимофеев А. В.* Методы обучения и минимизации сложности когнитивных нейромодулей супер-макро-нейрокомпьютера с программируемой архитектурой. — Доклады АН, 1984, т. 337, № 2, с. 180–183

7. Тимофеев А. В. Методы синтеза диофантовых нейронных сетей минимальной сложности // Доклады АН. 1995. Т. 345. № 1. С. 32–35.

8. *Тимофеев А. В., Сырцев А. В.* Модели и методы маршрутизации потоков данных в телекоммуникационных системах с изменяющейся динамикой. Москва: Изд-во «Новые технологии», 2005. С. 85.

9. *Timofeev A. V.* Parallel Structures and Self-Organization of Heterogeneous Polynomial Neural Networks for Pattern Recognition and Diagnostics of States // Pattern Recognition and Image Analysis. 2007. Vol. 17. No. 1. Pp. 163–169.

10. *Timofeev A. V , Azaletskiy P. S. , Myshkov P. S, Kesheng Wang*. Neural Network System for Knowledge Discovery in Distributed Heterogeneous Data // Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing, and Management. Vol. 207. 2006. Pp. 144–151.

11. *Косовская Т. М., Тимофеев А. В.* Иерархическое описание классов и нейросетевое распознавание сложных образов // Нейрокомпьютеры: разработка и применение. 2007. № 6. С. 30–33.

12. *Амбарян Т. Р., Тимофеев А. В.* Модели квантовых и нейронных вычислений в задачах обработки информации // Известия вузов. Приборостроение. 2005. Т. 48. № 7. С. 35–40.

НЕЙРОСЕТЕВЫЕ АЛГОРИТМЫ ПРИБЛИЖЕНИЯ РЕШЕНИЯ ОБРАТНОЙ ЗАДАЧИ КИНЕМАТИКИ РОБОТА ВЕРТИКАЛЬНОГО ПЕРЕМЕЩЕНИЯ (РВП)

А. В. Тимофеев, В. В. Титов

Мобильные роботы в настоящее время используются как в промышленности, так и в местах и средах, трудно доступных или опасных для человека (например, в космическом пространстве, под водой, а также в условиях высокой температуры и радиации). Одним из относительно новых направлений в развитии мобильной робототехники является создание роботов вертикального перемещения (РВП). Подобные роботы способны выполнять операции различного рода, находясь на поверхности с произвольным углом наклона.

Данное направление робототехники начало активно развиваться в 90-х годах в таких странах, как Япония, США, Англия, Германия, а также в России. В начале 90-х были выпущены многочисленные статьи, посвященные созданию или описанию уже действующих прототипов роботов вертикального передвижения.

Исследования в этом направлении используют для достижения поставленных задач разнообразные механизмы, материалы и принципы движения. В частности, широко используются вакуумные механизмы (присоски) [1–2], новейшие материалы в аппаратах крепления к поверхности (как правило, это различного рода адгезионные поверхности с микроструктурным и наноструктурным материалом) [8, 9] и принципы передвижения, заимствованные из природы (*bio-inspired hardware*) [2–7].

Разработанный в ЦНИИ РТК робот вертикального перемещения (рис. 1) имеет пять степеней подвижности и использует две вакуумные присоски (стопы) для перемещения по плоским вертикальным и поверхностям произвольной ориентации.

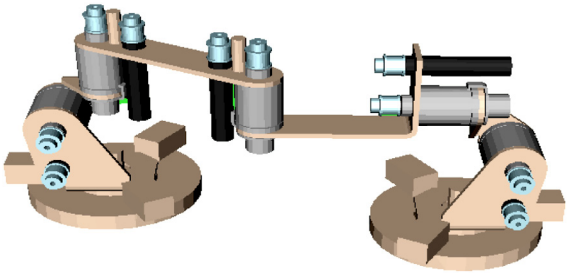


Рис. 1. Стопоходящий робот вертикального перемещения

В ходе экспериментальных работ и исследований по моделированию была подтверждена необходимость в решении обратной задачи кинематики данного механизма, применяемая в задачах контроля положения, планирова-

ния и калибровки. Из множества подходов к решению данной задачи [10–13] был выбран подход, связанный с численной минимизацией функционала

$$Z(q) = Q_{\text{зад}} - F(q), \tag{1}$$

где q — углы шарниров, $F(q)$ — решение прямой задачи кинематики, $Q_{\text{зад}}$ — целевое положение и ориентация в декартовых координатах.

Как известно, при использовании градиентных и других методов поиска минимумов этого функционала имеются известные проблемы, связанные со скоростью сходимости алгоритмов оптимизации функционала (1), зависящие от начальных приближений q_0.

В качестве альтернативы был рассмотрен подход, основанный на нейросетевой аппроксимацией обратной задачи кинематики. Данный подход столкнулся с рядом трудностей, связанных с невозможностью достаточно точно аппроксимировать все пространство допустимых конфигураций робота нейронной сетью приемлемой сложности. Время обучения сети и генерации обучающих выборок в рамках такого подхода оказалось велико.

Однако использование нейросетевого приближения для последующего численного решения задачи минимизации функционала (1) оказалось более выгодным по сравнению с табличным хранением начальных приближений. Помимо этого был исследован подход, при котором нейросеть приемлемой сложности (размерности) обучается на заранее заданной (или спланированной) траектории движения звеньев механизма с учетом конструктивных ограничений на углы поворота шарниров при этом движении.

Совокупность таких предобученных нейронных сетей является решением обратной задачи кинематики для всего пространства конфигураций. Она может быть использована как с алгоритмом численного уточнения, так и без него.

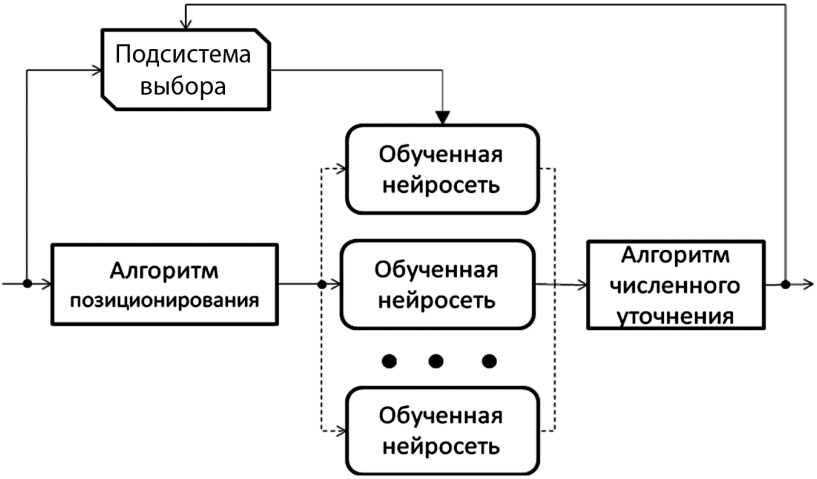


Рис. 2. Схема функционирования множества сетей

Решения обратной и прямой задачи кинематики играют важную роль при планирования движений любого робота. Как показали модельные эксперименты, решение обратной задачи кинематики при плоскостном хождении робота смысла не имеют ввиду того, что те классы препятствий (ступени не выше 5 см), которые робот способен преодолевать, легко преодолеваются за счет трех механических датчиков определения углов наклона присоски к поверхности (они же являются детекторами касания поверхности). При переходе на стену обратная задача кинематики, решаемая с помощью нейросетевого приближения с численным уточнением, в дальнейшем применялась при генерации промежуточных точек в алгоритме поиска траектории перехода на стену, а также для визуализации положения звеньев механизма при ручном управлении в сложно проходимых местах.

В дальнейшем планируется рассмотреть возможность автоматического синтеза походки всего робота, как, например, это было сделано в работе [15].

Работа выполнена при частичной поддержке гранта РГНФ-ГФЕН Китая № 10-08-91159-ГФЕН-а «Исследование научных проблем интеллектуального управления магистральными транспортными средствами и мобильными роботами».

Литература

1. *Градецкий В. Г., Рачков М. Ю.* Роботы вертикального перемещения, М.: Тип. Мин. Образования РФ, 1997. 223 с.

2. *Jizhong Xiao* and *Ali Sadegh*. City-Climber: A New Generation Wall-climbing Robots, IEEE International Conference on Robotics and Automation, 2007. Pp. 2764–2765.

3. *Daltorio K. A., Horchler A. D., Gorb S., Ritzmann R. E.* and *Quinn R. D.* (2005) "A Small Wall-Walking Robot with Compliant, Adhesive Feet". Int. Conf. on Intelligent Robots and Systems (IROS'05), Edmonton, Canada.

4. *Daltorio K. A., Gorb S., Peressadko A., Horchler A. D., Ritzmann R. E.* and *Quinn R. D.* (2005). "A Robot that Climbs Walls using Micro-structured Polymer Feet". International Conference on Climbing and Walking Robots (CLAWAR), London, U.K., Sept. 13–15, 2005.

5. *Kathryn A. Daltorio, Terence E. Wei, Stanislav N. Gorb, Roy E. Ritzmann, Roger D. Quinn* (2007). "Passive Foot Design and Contact Area Analysis for Climbing Mini-Whegs", Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (ICRA'07), Roma, Italy, 10–14 April 2007.

6. *Kathryn A. Daltorio, Timothy C. Witushynsky, Gregory D. Wile, Luther R. Palmer, Anas Ab Malek, Mohd Rasyid Ahmad, Lori Southard, Stanislav N. Gorb, Roy E. Ritzmann, Roger D. Quinn* (2008). "A Body Joint Improves Vertical to Horizontal Transitions of a Wall-Climbing Robot", Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (ICRA'08), Pasadena, CA, U.S.A., May 19–23, 2008.

7. *Gregory D. Wile, Kathryn A. Daltorio, Eric D. Diller, Luther R. Palmer, Stanislav N. Gorb, Roy E. Ritzmann,* and *Roger D. Quinn* (2008). "Creenbot: Walking Inverted Using Distributed Inward Gripping", Int. Conf. on Intelligent Robots and Systems (IROS'08), Nice, France.

8. *Luther R. Palmer III, Eric D. Diller, Roger D. Quinn* (2009). “Design Aspects of a Climbing Hexapod”. Int. Conf. on Climbing and Walking Robots (CLAWAR’09), Istanbul, Turkey.

9. *Matthias Greuter, Gaurav Shah, Gilles Caprari, Fabien Tâche, Roland Siegwart, Metin Sitti.* Toward Micro Wall-Climbing Robots Using Biomimetic Fibrillar Adhesives, CONFERENCE PAPER, LSA-CONF-2005-023.

10. http://www.nr21.com/

11. *Тимофеев А. В., Экало Ю. В.* Системы цифрового и адаптивного управления роботов: учебн. пособие. СПб.: Изд-во СПбГУ, 1999. 248 с.: ил. ISBN 5-288-01081-1.

12. *Тимофеев А. В.* Адаптивные робототехнические комплексы // Л.: Машиностроение, 1988. 332 с.

13. *Фу К., Гонсалес Р., Ли К.* Робототехника: пер. с англ. М.: Мир, 1989. 624 с., ил. ISBN 5-03-000805-5.

14. *B. Siciliano, L. Sciavicco, L. Villani, G.* Oriolo Robotics Modelling, Planning and Control 2009 Springer-Verlag London Limited ISBN 978-1-84628-641-4.

15. *Peter Dittrich, Andreas Bürgel, Wolfgang Banzhaf.* Learning to move a robot with random morphology, CONFERENCE PAPER,Conf. First European Workshop on Evolutionary Robotics 1998. P. 165–178.

ДИНАМИЧЕСКАЯ АУТЕНТИФИКАЦИЯ НА ОСНОВЕ КЛАВИАТУРНОГО ПОЧЕРКА

А. Торегожин
(аспирант СПбГУ, «Ланит-Терком»,
E-mail: arstan.toregozhin@lanit-tercom.com)

В обычной жизни мы узнаем друг друга в лицо, если знакомы. Если не знакомы — по паспорту или аналогичному документу с фотографией. «Опознать» же человека, сидящего за компьютером по ту сторону «Сети», несколько сложнее — это требует достаточно специфичных методов.

Аутентификация пользователей обычно выполняется неким программным модулем, находящимся непосредственно на компьютере, на который пользователь пытается получить прямой или удаленный доступ.

Информация, по которой опознается пользователь, бывает трех видов:

- Пользователь знает нечто уникальное и демонстрирует компьютеру это знание. Такой информацией может быть, например, пароль.
- Пользователь имеет предмет с уникальным содержимым или с уникальными характеристиками.
- Аутентификационная информация является неотъемлемой частью пользователя. По этому принципу строятся системы биометрической аутентификации, использующие в качестве информации, например, отпечаток пальца.

Цель данной работы предложить подход к построению биометрических систем, позволяющий упростить и повысить их удобство.

Такая биометрия будет использовать только самое распространённое оборудование, такое как клавиатура. Биометрическая система будет состоять только из программного обеспечения, способного функционировать на большинстве компьютеров и работать быстро и незаметно для пользователя. Она будет удобной, то есть для аутентификации не нужно вводить пароль, нужно просто работать, для настройки системы под пользователя опять же не надо ничего делать, просто работать, система сама обучится и настроится.

Искусственные нейронные сети, благодаря их адаптивности, обучению и способности к обобщению функциональных зависимостей, широко применяются в задачах анализа статистических наблюдений сложной структуры. Благодаря этому они были выбраны как основа для достижения целей данной работы.

Таким образом, постановка задачи состоит в следующем:

- Провести анализ клавиатурного почерка с целью выявить характеризующие его параметры.
- На основе полученной информации о клавиатурном почерке разработать и реализовать архитектуру обучающейся нейронной сети.

- Разработать систему, использующую нейронную сеть для аутентификации пользователя во время его работы незаметным для него способом.

Проведённые исследования выявили следующие параметры, характеризующие пользователя:

- временные промежутки между нажатиями клавиш;
- время удержания клавиши нажатой;
- ритм печати;
- время дня — проведённые исследования показывают, что наиболее быстро пользователь работает в середине рабочего дня, чуть медленнее утром, и гораздо медленнее — вечером;
- последовательности букв, что обусловлено как близостью клавиш на клавиатуре, так и персональными особенностями рук пользователя;
- приложение, в котором работает пользователь, например, в приложении Word, пользователь будет печатать быстрее, чем в приложении калькулятор;
- день недели: это связано с психологическим состоянием пользователя, во время трудовой недели пользователь испытывает психологическую нагрузку, которая может накапливаться (например, врачи, дежурящие в больнице по несколько дней, к концу смены истощены), что сказывается на их клавиатурном почерке.

Первое, что надо сделать при создании приложений на основе нейронных сетей, это определить основные требования, которым должна удовлетворять проектируемая нейронная сеть. В данной работе нейронная сеть построена по следующим принципам:

- обучение методом «обучение с учителем», потому, что нейронная сеть должна обучаться на клавиатурном вводе пользователя;
- рекурсии и возвраты в проектируемой нейронной сети могут привести к неопределённо долгому процессу принятия решения, что неприемлемо в динамических системах аутентификации;
- нейронная сеть работает с параметрами, описанными на прошлом слайде. Данное требование обусловлено тем, что нейронная сеть должна предсказывать клавиатурный ввод пользователя (именно в этом и состоит новизна решения), длина слова, подаваемая на вход нейронной сети, не должна быть меньше трёх, так как это минимальное число букв, необходимое для замера клавиатурного ритма. Данное утверждение следует из того, что ритм описывается пропорциями временного промежутка, например, если пользователь ввёл слово длиной в 3 символа с промежутками между буквами $А$ и $Б$, то ритм ввода данного слова будет равен $А/(А+Б)$ и $Б/(А+Б)$. Отсюда видно, что ритм слова длиной в две буквы всегда будет равен единице.

В процессе работы над архитектурой системы был разработан механизм поиска оптимальной нейронной сети для определённых выше требований

и его реализация в виде дополнения к общедоступной библиотеке работы с нейронными сетями NeuronDotNet.

Для механизма нужно три набора тестовых данных:

- «обучающие» данные — примеры, записанные пользователем для обучения нейронной сети,
- «хорошие» данные — примеры, записанные пользователем для тестирования нейронной сети,
- «отрицательные» данные — примеры предназначены для тестового прогона нейронной сети, записанные другим человеком.

Данные нужны для замера качества сгенерированной нейронной сети с целью выбора наиболее оптимальной.

В виду того, что специфика задачи предполагает «обучение с учителем», генерация каждой новой нейронной сети происходит последовательным перебором всевозможных архитектур, ограниченных возможностями NeuronDotNet.

Для сгенерированной нейронной сети был разработан механизм распознавания пользователя по клавиатурному почерку представленный на рисунке.

Механизм сбора данных собирает всю нужную информацию незаметным для пользователя способом и вне зависимости от того, в каком приложении он сейчас работает.

Анализатор проводит аутентификацию при помощи нейронной сети и обрабатывает результаты в зависимости от текущих настроек точности.

Менеджер принятия решений ведёт статистику работы анализатора, управляет политиками аутентификации и сообщает пользователю о провале аутентификации, если таковая имеет место.

Система реализована на платформе .NET компании Майкрософт и работает на операционной системе Windows.

Назовём результаты работы:

- Проведён анализ параметров, характеризующих клавиатурный почерк;
- Предложен механизм поиска нейронных сетей по входным данным и его реализация;
- Разработана биометрическая система аутентификации пользователя по клавиатурному почерку, которая:
  - Состоит только из программного обеспечения;
  - Предоставляет возможность обучения;
  - Аутентификация ведётся незаметным для пользователя способом.

Литература

1. *Кобиелус Д.* Информационная безопасность: идентификация и аутентификация. 1997.

2. *Брюхомицкий Ю. А., Казарин М. Н.* Исследование биометрических систем динамической аутентификации пользователей ПК по рукописному и клавиатурному почеркам. Учебно-методическое пособие. Таганрог: Изд-во ТРТУ.

3. Password alternatives Network Security. Volume 1995, Issue 2, February 1995, Pp. 11–15.

4. *Фор А.* Восприятие и распознавание образов. М.: Машиностроение, 1989.

5. *Уоссермен Ф.* Нейрокомпьютерная техника, М.: Мир.

6. Итоги науки и техники: физические и математические модели нейронных сетей. Том 1. М.: ВИНИТИ.

7. Artificial Neural Networks: Concepts and Theory, IEEE Computer Society Press.

8. *Трушина Е. А.* Идентификация пользователя ЭВМ по клавиатурному почерку, как метод защиты от несанкционированного доступа. 1997. http://www.securityclub.ru/

9. Нейронные сети в финансовых прогнозах. http://icmm.ru

10. Лепёшкин О.М., Скубицкий А.В. Разработка подхода к распознаванию биометрического портрета пользователя по клавиатурному почерку на основе методов нелинейной динамики.

11. *Бугайченко Д. Ю., Соловьев И. П.* Методы накопления и анализа опыта для самонастраивающихся мультиагентных систем с временными ограничениями.

12. *Широчин В. П., Кулик А. В., Марченко В. В.* Динамическая аутентификация на основе анализа клавиатурного почерка.

13. http://www.math-interactive.com/

14. *Короткий С.* Нейронные сети: основные положения.

15. Журнал Byte. Август 1989 г. С. 227–233.

16. Журнал AI Expert. Январь 1991 г.

РАСПОЗНАВАНИЕ ЖЕСТОВ С ИСПОЛЬЗОВАНИЕМ ОБУЧЕНИЯ НЕЙРОННОЙ СЕТИ БЕЗ УЧИТЕЛЯ

А. Р. Ханов

Руководитель:
М. В. Баклановский

Задача распознавания жестов пользователя с помощью веб-камеры приобрела большую популярность. Существует множество работ на эту тему, для ее решения применяются мощные методы как алгоритмические, так и нейросетевые. Однако на данный момент не существует точного решения этой задачи в общем случае. Мы рассмотрим статический метод с использованием карты Кохонена (SOM) в качестве классификатора [1].

Эта задача обладает большой изменчивостью: рука может находиться в различных обстановках, на которых также могут быть движущиеся объекты. Поэтому на задачу следует наложить ряд ограничений:

1) Известно, что на подаваемом с камеры потоке кадров присутствует лишь одна рука как единственный движущийся объект на изображении.
2) Фон руки остается неподвижным и неизменным.

Для выделения объекта мы заранее зафиксируем изображение, которое будем считать фоном, затем вычтем его из каждого последующего кадра. Таким образом останется лишь изображение распознаваемого нами объекта — руки. Этот метод не новый и применялся ранее во многих работах [4]. К сожалению на реальном видеопотоке появляются различные помехи:

1) Случайные помехи обусловлены небольшими колебаниями в воспринимаемом камерой цвете, а также особенностями сжатия видеопотока, поступающего с нее.
2) Неслучайные помехи появляются на изображении из-за того, что в зависимости от освещенности, тени и других факторов могут появляться повторяющиеся из кадра в кадр несоответствия цветов пикселя фона и пикселя изображения.

Кроме того, на самом объекте цвет пикселя может совпасть с цветом на фоновом изображении, что приведет к появлению различных «артефактов» на объекте. Наша задача — суметь извлечь нужные нам признаки несмотря на эти помехи.

Следующий шаг — сформировать пространство признаков для SOM. На эту тему существует множество работ. Можно использовать цветовые характеристики кожи руки, в случае динамического распознавания можно использовать траекторию движения руки и ее частей, можно выделять контур руки, линии пальцев. Особое место среди этих признаков занимают инварианты — признаки, которые не чувствительны к различным аффинным преоб-

разованиям: сдвиг по осям, изменение масштаба, повороты. Среди чисто нейросетевых решений стоит выделить такое устройство как Неокогнитрон [2]. Его слои позволяют уменьшить зависимость распознавания от аффинных преобразований, однако из-за громоздкости низкой скорости работы он мало используется для этой задачи.

Для описанных способов можно выделить множество недостатков: высокая вычислительная сложность, неустойчивость к аффинным преобразованиям, неустойчивость к помехам. Кроме того вычислительная сложность многих методов сильно зависит от числа пикселей и скорость вычисления сильно падает с увеличением разрешения.

В случае распознавания руки мы, прежде всего, имеем дело с кистью и предплечьем, мы хотим уметь задавать форму кисти, однако при различных сдвигах руки предплечье вносит дополнительную помеху. Если мы получаем контур или выводим из изображения составные части в виде линий, то мы имеем возможность проанализировать структуру руки, но выделение этих признаков — достаточно долгая операция, зависящая напрямую от числа пикселей на изображении.

Вместо этого мы попытаемся сразу избавиться от зависимости $O(W * H)$ и перейти к зависимости $O(W + H)$, где W и H — это соответственно ширина и высота. Сформируем массив размером W и в каждую его ячейку просуммируем число ненулевых пикселей в вертикальном сечении. Также сформируем массив размером H и в каждую его ячейку запишем число пикселей в соответствующем горизонтальном сечении. Получаются две свертки. Это некоторые функции, в которых вместе $W + H$ отсчетов. Разумеется, существуют два различных объекта, обладающих в точности одинаковыми свертками и в этом случае они для нас будут неразличимы. Тогда заведем еще один массив с $W + H$ элементами, в каждую его ячейку будем записывать число ячеек в диагональном сечении. Таких сверток можно посчитать достаточно много, просто поворачивая прямую, вдоль которой будут производиться сечения. И таким образом в случае недостаточной точности можно просто добавлять признаки. В итоге все изображение можно представить как последовательность векторов $(v_1, ..., v_n)$, каждый из которых является сверткой вдоль какой-то прямой.

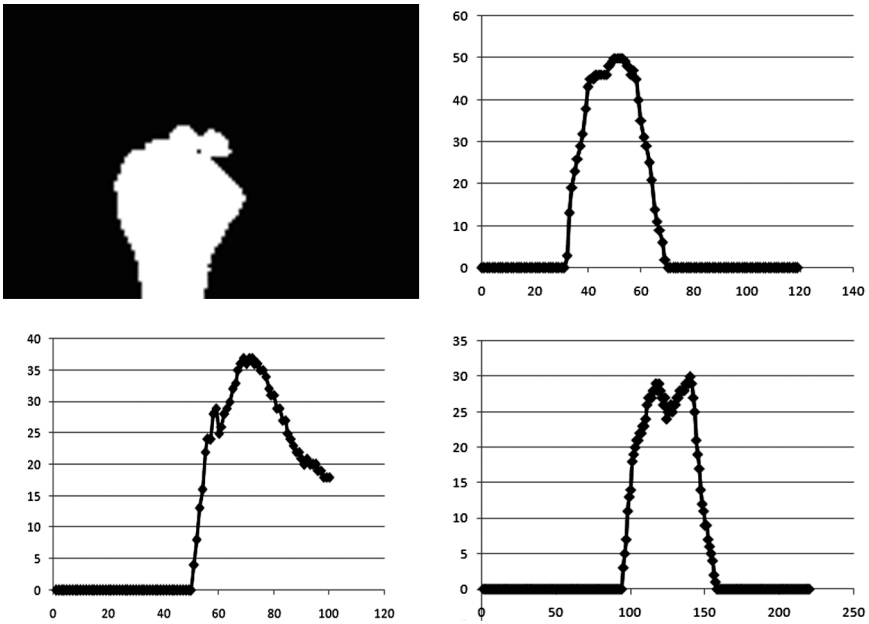
Однако сформировать пространство признаков недостаточно. Для обучения карты Кохонена нужно научиться сравнивать эти признаки. В результате нужно выдать число, которое будет оценивать схожесть признаков. Это число мало, если признаки схожи. Но и этого может быть недостаточно. Для нашей последовательности векторов можно построить метрику, которая будет учитывать аффинные преобразования. В данный момент рассматривается лишь операция сдвига по осям.

Для сравнения двух признаков вначале научимся сравнивать два вектора. В качестве метрики для них можно было бы выбрать обычную сумму квадратов отклонений соответствующих отсчетов. Но для учета их возможного

сдвига возьмем минимум по всем циклическим перестановкам. В этом случае мы получим определенную независимость от сдвига.

Теперь нужно научиться сравнивать две последовательности векторов. Для этого были использованы различные методы: умножение метрик, сложение, взятие минимума. В данный момент идет поиск лучшей метрики для более точного учета признаков.

Пример. На рисунке изображена кисть руки, сжатая в кулак. Для нее было посчитано три признака: свертка по ширине, высоте и диагонали (левый нижний — правый верхний углы).



Карта Кохонена (50×50, двумерный тор) была обучена с использованием 9 семплов и сумела равномерно разбиться на кластеры. Сдвиги семпла привели к тому, что карта относила его к тому кластеру, который и должен был с несущественной погрешностью. Метод также показал высокую устойчивость к различным помехам на изображении.

В данный момент идет работа по осуществлению распознавания с учетом не только сдвигов, но и изменений масштаба и поворотов.

В будущем можно формировать признаки не только линейным суммированием. Кроме того можно подбирать наиболее существенные наборы признаков для конкретных видов изображений.

Литература

1. *Т. Кохонен.* Самоорганизующиеся карты.
2. *Г. Э. Яхъяева.* Нечеткие множества и нейронные сети.
3. *Tobely, Yoshiki, Tsuda, Tsuruta, Amamiy.* Dynamic Hand Gesture Recognition Based On Randomized Self-Organizing Map Algorithm.
4. *Simei Gomes Wysoski.* A Rotation Invariant Static Hand Gesture Recognition System.

Автоматное управление, эволюционные алгоритмы, верификация на моделях

Шалыто
Анатолий Абрамович

д.т.н.
профессор,
заведующий кафедрой технологии программирования
СПбГУ ИТМО

РАЗРАБОТКА МЕТОДА ВОССТАНОВЛЕНИЯ ФРАГМЕНТОВ НУКЛЕОТИДНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ПО ПАРНЫМ ЧТЕНИЯМ

А. А. Сергушичев, В. В. Исенбаев, Ф. Н. Царев, А. А. Шалыто

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Е. Б. Прохорчук

Центр «Биоинженерия» РАН

Многие современные задачи биологии и медицины требуют знания генома живых организмов, который состоит из нескольких нуклеотидных последовательностей ДНК — по одной в каждой хромосоме. В связи с этим возникает необходимость в дешевом и быстром методе секвенирования, то есть определения последовательности нуклеотидов в образце ДНК.

Существующие технические средства не позволяют считать разом всю молекулу ДНК организма, поэтому перед считыванием молекулы ДНК дробятся на маленькие фрагменты. Изначально, например, для проекта «Геном человека» [1], использовались фрагменты длиной около 800–1000 нуклеотидов, но такой подход является дорогим и обладает низкой пропускной способностью — число прочитанных нуклеотидов в единицу времени. По этим причинам в настоящее время используется следующая достаточно дешевая и эффективная технология: сначала вычленяется случайно расположенный в геноме фрагмент длиной около 500 нуклеотидов, а затем происходит считывание двух последовательностей с его концов (длиной примерно по 100 нуклеотидов каждая). Эти последовательности называются *парными чтениями*. Процесс повторяется такое число раз, чтобы обеспечить достаточно большое покрытие генома чтениями. Такая технология реализуется, например, в секвенаторах компании Illumina [2]. Заметим, что при физическом чтении могут возникать ошибки, но для каждого прочитанного нуклеотида известно его качество — вероятность того, что он был прочитан правильно.

Кроме того, программные средства тоже не позволяют восстановить геном целиком. Поэтому результатом работы сборщиков является набор *контигов*, объединенных в *скэффолды*. Контигами называются максимальные подпоследовательности генома, которые удалось восстановить. Скэффолдами называются такие последовательности контигов с оценками на расстояния между ними, про которые предполагается, что они в той же последовательности и на таких же расстояниях находятся в геноме.

На данный момент существует множество сборщиков геном из парных чтений [3–6], большинство из которых полностью основаны на использовании графа Де Брюина. Граф Де Брюина размерности k над алфавитом

Σ — это граф, вершинами которого являются все строки над алфавитом Σ длины k (они называются *k-мерами*), а ребра соединяют пары таких вершин, что суффикс длины $k-1$ первой вершины является префиксом второй вершины. Можно заметить, что есть однозначное соответствие между ребрами и $(k+1)$-мерами — каждому ребру соответствует $(k+1)$-мер, полученный конкатенацией k-мера начальной вершине ребра и последнего символа k-мера конечной вершины ребра. В графе Де Брюина над алфавитом $\{A, T, G, C\}$ геном представляет из себя набор путей (возможно не простых), соответствующих хромосомам.

Как уже говорилось, эти сборщики не позволяют собрать весь геном полностью, поэтому имеет смысл искать методы, которые позволят улучшить результаты. Кроме того, желательно оптимизировать использование памяти и быстродействие, чтобы уменьшить общую стоимость секвенирования.

В работе [7] была предложена идея с помощью графа Де Брюина восстанавливать фрагменты, отвечающие парным чтениям, а затем использовать полученные последовательности по 500 нуклеотидов для сборки генома с помощью имеющихся сборщиков из «длинных» чтений, основанных, например, на алгоритме Overlap-Layout-Consensus [8], таких как Newbler [9] или Celera Assembler [10]. К сожалению, предложенный в этой работе алгоритм восстановления фрагментов оказался неприменим для больших геномов размером от миллиарда нуклеотидов.

В данной работе делается попытка развить метод восстановления фрагментов, предложенный в работе [7], чтобы он стал эффективным по используемой памяти и времени работы и его можно было применить к геномам размером несколько миллиардов нуклеотидов.

Важным предварительным шагом является исправление ошибок, например, с помощью частотного анализа. Этот шаг позволяет сильно снизить общий объем информации во входных данных, при этом увеличив полезный объем.

Таким образом, весь процесс восстановления генома по парным чтениям состоит из четырех этапов:

1. Исправление ошибок в исходных данных.
2. Восстановление фрагментов. Восстановленные фрагменты будем называть квазиконтигами.
3. Сборка контигов из квазиконтигов.
4. Сборка скэффолдов.

Данная работа сфокусирована на втором этапе.

В предлагаемом методе используется подграф графа Де Брюина, в котором множество ребер состоит только из тех $(k+1)$-меров, которые встречаются в чтениях достаточно большое число раз, чтобы их можно было с очень большой вероятностью считать входящими в геном, а множество вершин такое же, как и в исходном графе. Если участок нуклеотидной последователь-

ности покрылся достаточно хорошо, то есть все входящие в него $(k+1)$-меры по много раз входят в исходные данные, то в этом подграфе существует путь между первым и последним k-мерами участка

Предлагаемый метод основан на поиске такого пути для фрагмента, соответствующего парным чтениям. Из всех путей нас интересуют только те, которые укладываются в априорные границы длин фрагментов, поэтому слишком короткие и слишком длинные пути можно отбросить. Из оставшихся путей следует выбрать те, которые достаточно «похожи» на парные чтения. Для этого можно сравнить все нуклеотиды из парных чтений с соответствующими им нуклеотидами из пути, посчитать вероятность таких ошибок и сравнить ее с математическим ожиданием этой вероятности. Те пути, для которых различие между этими величинами велико, отбрасываются. Оставшиеся пути — хорошие кандидаты на роль пути, соответствующего фрагменту в действительности. Если нашелся единственный такой путь, то можно с очень большой уверенностью сказать, что он отвечает реальному пути в геноме, поэтому этот фрагмент считается восстановленным, а найденный путь выводится как квазиконтиг. Если таких путей несколько, то не ясно какой из них на самом деле соответствует фрагменту, поэтому этот фрагмент не восстанавливается. Если не нашлось ни одного такого пути, то данный фрагмент ДНК был плохо покрыт чтениями и его восстановление невозможно.

Для того, чтобы потребление памяти предлагаемого метода было не очень большим, необходимо иметь компактное представление используемого подграфа графа Де Брюина. Для его хранения достаточно хранить только множество его ребер, что можно эффективно делать, используя, например, хеш-таблицу с открытой адресацией. Преимуществами такого подхода хранения перед другими являются его простота, быстродействие и возможность балансировки между используемой памятью и скоростью. Более эффективными с точки зрения потребляемой памяти являются rank/select словари [11], которые позволяют сделать ее использование близким к энтропии, но из-за этого увеличивается время доступа.

Для поиска путей, соединяющих две заданные вершины и не превосходящих по длине некоторой максимальной длины $L_{\max}$, применяется методика meet-in-the-middle, в которой происходит одновременный поиск путей из первой вершины по прямым ребрам и путей из второй вершины по обратным ребрам. Если конец какого-то пути из первой вершины совпадает с концом пути из второй вершины, то можно, объединив эти пути, получить путь из первой вершины во вторую. Для реализации такого подхода удобно запустить одновременно два обхода в ширину: из первой вершины по прямым ребрам и из второй — по обратным. Тогда на каждом шаге L можно поддерживать инвариант: для первой вершины известны все исходящие из нее пути длины L_1, а для второй — все входящие пути длины L_2, причем $L_1+L_2=L$. Таким образом, на каждом шаге можно пересечь множества конечных вершин пу-

тей и найти все пути длины L из первой вершины во вторую. Для перехода к следующей итерации необходимо выбрать, в каком из обходов увеличивать на единицу длину путей. Самым простым является поочередное увеличение длин, но для более эффективного использования памяти и времени лучше производить увеличение в том обходе, в котором на данный момент число концевых вершин меньше, чем в другом. Кроме того, разумно параллельно отбрасывать те пути, которые на текущей итерации уже сильно отличаются от парных чтений.

Заметим, что это решение хорошо масштабируется на несколько ядер или даже на несколько компьютеров, так как все парные чтения можно разбить на группы, которые можно обрабатывать независимо, а построение хеш-таблицы занимает относительно немного времени.

Данный подход был разработан и применен в рамках проекта *dnGASP* [12], в котором предлагалось восстановить синтетический геном размером в 1,8 миллиардов нуклеотидов. Восстановление фрагментов работало на 24-ядерной машине с 64 ГБ оперативной памяти. Значение k было выбрано равным 30 — в этом случае один k-мер можно хранить в 64-битном целом числе. За сутки было обработано 350 миллионов парных чтений, для 67 % которых удалось восстановить исходные фрагменты. Для 6 % не удалось найти ни одного пути, для 27 % фрагментов однозначно восстановить не получилось из-за большого числа путей

В дальнейшем планируется увеличить k, а также исследовать возможность внести дополнительные улучшения в этот метод для увеличения его эффективности и быстродействия.

Литература

1. International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome // Nature, Volume 409. 15 February 2001. P. 860–921. http://www.nature.com/nature/journal/v409/n6822/pdf/409860a0.pdf

2. Illumina, Inc. http://www.illumina.com/

3. *Zerbino D. R., Birney E.* Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research 18. 2008. P. 821–829.

4. *Simpson J. T., Wong K., Jackman S. D., Schein J. E., Jones S. J., Birol I.* ABySS: A parallel assembler for short read sequence data. Genome Research 19. 2009. P. 1117–1123.

5. *Li R., Zhu H., Ruan J., Qian W., Fang X., Shi Z., Li Y., Li S., Shan G., Kristiansen K.,* et al. De novo assembly of human genomes with massively parallel short read sequencing. Genome Research 20. 2010. P. 265–272.

6. *Pevzner P. A., Tang H., Waterman M. S.* An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences 98. 2001. P. 9748–9753.

7. *Исенбаев В. В., Шалыто А. А.* Разработка системы секвенирования ДНК с использованием paired-end данных. http://is.ifmo.ru/genom/_isenbaev_thesis.pdf

8. *Pevzner P.* Computational molecular biology: an algorithmic approach. MIT Press. 2000. P. 61–62.

9. Products & Solutions — Analysis Tools — GS De Novo Assembler : 454 Life Sciences, a Roche Company. http://454.com/products-solutions/analysis-tools/gs-de-novo-assembler.asp

10. SourceForge.net: wgs-assembler. http://sourceforge.net/apps/mediawiki/wgs-assembler/index.php?title=Main_Page

11. Okanohara D., Sadakane K. Practical Entropy-Compressed Rank/Select Dictionary. Computing Research Repository, arXiv:cs/0610001v1. 2006. http://arxiv.org/pdf/cs/0610001v1

12. De novo Genome Assembly Project (dnGASP). http://cnag.bsc.es

РАЗРАБОТКА МЕТОДА УДАЛЕНИЯ ОШИБОК ИЗ НАБОРА ЧТЕНИЙ НУКЛЕОТИДНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ

А. В. Александров, С. В. Казаков, С. В. Мельников, Ф. Н. Царев, А. А. Шалыто

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Е. Б. Прохорчук

Центр «Биоинженерия» РАН

Многие современные задачи биологии и медицины требуют знания генома живых организмов, который состоит из нескольких нуклеотидных последовательностей ДНК. В связи с этим возникает необходимость в дешевом и быстром методе секвенирования, то есть определения последовательности нуклеотидов в образце ДНК.

Существующие технические средства (*секвенаторы*) не позволяют считать разом всю молекулу ДНК организма. Вместо этого они позволяют читать фрагменты генома небольшой длины. Длина фрагмента может варьироваться и является важным параметром секвенирования, так как от нее напрямую зависит стоимость секвенирования и время, затрачиваемое на чтение одного фрагмента: чем больше длина считываемого фрагмента, тем выше стоимость чтения и тем дольше это чтение происходит. В связи с этим сейчас получил распространениес ледующий дешевый и эффективный подход: сначала вычленяется случайно расположенный в геноме фрагмент длиной около 500 нуклеотидов, а затем считываются его префикс и суффикс (длиной по 114 каждый). Эти префикс и суффикс называются *парными чтениями*. Этот процесс повторяется такое число раз, чтобы обеспечить достаточно большое покрытие генома чтениями.

Описанный выше технологический процесс реализуется, например, секвенаторами компании *Illumina* [1]. При этом важной особенностью работы этих секвенаторов является допуск ошибок. Это означает, что некоторые нуклеотиды секвенируются неверно (например, вместо нуклеотида A читается нуклеотид G). Ошибки замены — не единственный тип ошибок, допускаемый секвенаторами. Так, возможны также ошибки вставки и удаления. Однако эти ошибки встречаются в довольно спецефических случаях и по сравнению с ошибками замены очень редки.

Помимо самой последовательности нуклеотидов результатом работы секвенатора является также последовательность из величин качества для каждого нуклеотида. В ней содержится информация о качестве чтения каждого нуклеотида, по которой может быть вычислена вероятность того, что данный нуклеотид был прочитан неверно.

Для эффективной работы последующих стадий алгоритма очень важно исправить как можно больше ошибок в чтениях.

После исправления ошибок запускается алгоритм восстановления фрагментов нуклеотидной последовательности, результатом работы которого являются целые фрагменты, а не только их префиксы и суффиксы. Эти фрагменты называются *квазиконтиги и* используются для построения *контигов* — максимальных непрерывных последовательностей нуклеотидов, которые удалось восстановить. Затем контиги используются для построения *скэффолдов* — последовательностей контигов, разделенных промежутками с длинами, для которых известны оценки. Таким образом, результатом работы сборщиков является набор последовательностей нуклеотидов, разделенных промежутками более-менее известной длины.

Существует большое число сборщиков [2–6], осуществляющих все или только некоторые из приведенных выше этапов. В данной работе описан метод, осуществляющий исправление ошибок и использующийся в качестве первого этапа сборки генома из набора чтений.

Для эффективного исправления ошибок необходимо, чтобы каждая позиция генома была прочитана несколько раз, что, ввиду небольшой вероятности ошибки, дает право считать, что наибольшее число раз нуклеотид на каждой позиции был прочитан верно. На практике используются наборы чтений, покрывающие геном несколько десятков раз. Важно отметить, что не только отдельные позиции всего генома были прочитаны несколько десятков раз, но и небольшие его подстроки (не длиннее самих чтений) встречаются в чтениях несколько раз, причем чем длиннее подстрока, тем меньше шансов, что несколько различных чтений ее содержат. Последнее соображение вытекает не только из соображений вероятности попадания чтения на конкретную подстроку, но и из факта наличия в чтениях ошибок.

Подстроки, упомянутые выше, обычно называются k-мерами. К выбору величины k следует подходить достаточно серьезно, так как этот параметр сильно влияет на работу алгоритма. При выборе значения этого параметра следует учитывать следующие простые, но важные соображения.

- Величина k должна быть значительно меньше длины чтений. Если длина k-мера будет сравнима с длиной чтения, то большинство k-меров будет встречаться в чтениях один раз, что не даст никакой информации для исправления ошибок.
- Величина k должна быть достаточно большой, чтобы вероятность того, что случайный k-мер заданной длины встречается в чтениях, была ничтожно маленькой. В противном случае некоторые k-меры, содержащие ошибку, не будут исправлениы только потому, что в чтениях присутствует «похожий» на них k-мер, прочитанный из другого места генома.

Первая часть работы алгоритма состоит в собирании информации о k-мерах. Для каждого k-мера, присутствующего в чтениях хотя бы раз, подсчитывается, сколько раз он встречается в чтениях. На основании этой

статистики все *k*-меры можно разделить на 2 группы — *«надежные» k*-меры и *«подозрительные»*. «Надежные» *k*-меры — это те, которые встречаются в чтениях достаточно большое число раз (точное значение данного порога — еще один, но не такой существенный, как значение *k,* параметр алгоритма). С «надежными» *k*-мерами делать ничего не нужно, предполагается, что в них ошибок нет. «Подозрительные» же *k*-меры полагаются содержащими одну или, что менее вероятно, несколько ошибок, которые надлежит исправить.

После выделения «подозрительных» *k*-меров для каждого из них необходимо решить, в какой именно позиции была совершена ошибка. Для этого предлагается перебрать все позиции *k*-мера (их ровно *k* штук) и все возможные нуклеотиды, попробовать заменить имеющийся нуклеотид на перебираемый и проанализировать получившийся *k*-мер. Если новый *k*-мер попадает в группу «надежных», значит, возможно, рассматриваемый *k*-мер является результатом ошибочного прочтения. Если в течение перебора был найден только один «надежный» *k*-мер, получающийся из «подозрительного» путем замены одного нуклеотида на другой, полагается, что данный «подозрительный» *k*-мер исправлен, а соответствующее исправление запоминается. Если таких *k*-меров несколько, неясно, какое из исправлений запоминать, поэтому в таких случаях «подозрительные» *k*-меры не исправляются. И, наконец, если не было найдено ни одного способа исправить «подозрительный» *k*-мер, полагается, что в нем было совершено больше одной ошибки, после чего запускается аналогичная процедура, но с исправлением не одного, а сразу двух нуклеотидов. Аналогичным образом можно пытаться изменить не только пары, но и тройки, а также кортежи из большего числа нуклеотидов, однако данное обобщение ощутимо сказывается на быстродействии алгоритма.

Важно отметить, что алгоритм поиска ошибок в *k*-мерах легко распараллеливается, так как для обработки одного *k*-мера ему требуется только доступ на чтение к общей структуре данных, хранящей статистику по содержанию *k*-меров в чтениях, а также кратковременный доступ на запись для сохранения результата.

Описанный подход был разработан и применен в рамках проекта *dnGASP [*7]. В этом проекте участникам предлагалось восстановить синтетический геном, содержащий около 1,8 миллиардов нуклеотидов. При реализации описанного алгоритма для хранения *k*-меров использовался хэш-мэп с открытой адресацией. Это позволило осуществлять добавление нового *k*-мера и проверку «надежности» нового за константное время (последнее особенно важно для распараллеливания алгоритма). Ввиду того, что в рамках этого конкурса чтения имели длину 114, было выбрано значение *k*, равное 30. Эмпирически было установлено, что «надежные» *k*-меры — те, которые встречаются в чтених не менее 4 раз. Алгоритм исправления ошибок работал на 24-ядерном компьютере с 64 ГБ оперативной памяти. До запуска алгоритма в исходных данных было 6,5 миллиардов различных *k*-меров, из которых 3 миллиарда «надежных». Алгоритм работал около суток, после чего в дан-

ных стало 3,9 миллиардов (что на 40 % меньше, чем в начале) различных *k*-меров, из которых 3,3 миллиарда «надежных».

В настоящий момент исследуются возможные изменения, которые можно осуществить для улучшения эффективности и быстродействия предложенного метода.

Литература

1. Illumina, Inc. http://www.illumina.com/

2. *Zerbino D. R., Birney E.* Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research 18. 2008. P. 821–829.

3. *Simpson J. T., Wong K., Jackman S. D., Schein J. E., Jones S. J., Birol I.* ABySS: A parallel assembler for short read sequence data. Genome Research 19. 2009. P. 1117–1123. http://genome.cshlp.org/content/19/6/1117.full.pdf+html

4. *Li R., Zhu H., Ruan J., Qian W., Fang X., Shi Z., Li Y., Li S., Shan G., Kristiansen K.*, et al. De novo assembly of human genomes with massively parallel short read sequencing. Genome Research 20. 2010. P. 265–272.

5. *Pevzner P. A., Tang H., Waterman M. S.* An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences 98. 2001. P. 9748–9753.

6. *Butler J., MacCallum I., Kleber M., Shlyakhter I. A., Belmonte M. K., Lander E. S., Nusbaum C., Jaffe D. B*. AllPaths: De novo assembly of whole-genome shotgun microreads. Genome Research 18. 2008. P. 810–820.

7. De novo Genome Assembly Project (dnGASP). http://cnag.bsc.es/

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ НА ОСНОВЕ ОБУЧАЮЩИХ ПРИМЕРОВ ДЛЯ ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ ДЛЯ УПРАВЛЕНИЯ МОДЕЛЬЮ БЕСПИЛОТНОГО САМОЛЕТА

А. В. Александров, С. В. Казаков, А. А. Сергушичев, Ф. Н. Царев, А. А. Шалыто

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В последнее время для программирования систем со сложным поведением все шире применяется автоматное программирование, в рамках которого поведение программ описывается с помощью конечных детерминированных автоматов (в дальнейшем — автоматов) [1].

В автоматном программировании программы предлагается строить в виде набора автоматизированных объектов управления. Каждый такой объект состоит из объекта управления и системы управления (системы управляющих автоматов). Система автоматов получает на вход события из внешней среды и от объекта управления. На основании этих данных система управления вырабатывает выходные воздействия для объекта управления.

Для многих задач управляющие автоматы удается строить эвристически, но существуют задачи, для которых такое построение невозможно или затруднительно. К этому классу относятся, например, задача «Умный муравей» [2–4], задача «Умный муравей-3» [5] и задача об управлении моделью беспилотного летательного аппарата [6].

Существует несколько подходов к решению последней задачи. Один из них состоит в выделении «идеальной» траектории из нескольких полетов, выполненных человеком, и последующем следовании ей. Такой подход описан в работе [7].

Другой подход — использование автоматов для управления беспилотным летательным аппаратом и построение таких автоматов с помощью генетических алгоритмов, описанных в работах [8–12].

В работе [13] для генерации конечного автомата верхнего уровня, управляющего моделью беспилотного самолета, применяется алгоритм генетического программирования, основанный на использовании метода сокращенных таблиц для представления конечных автоматов. При этом вычисление функции приспособленности базируется на моделировании поведения самолета во внешней среде, которое занимает около 5 минут для одного автомата на одном ядре современного процессора.

Цель настоящей работы — разработка лишенного указанного недостатка метода. Для этого предлагается строить автоматы управления объектами от-

дельно для каждого из их режимов работы с помощью генетического программирования, *используя обучающие примеры*, создаваемые для каждого режима. Задавая большое число обучающих примеров, можно, как и в работе [7], избавиться от неточностей, допускаемых человеком при управлении. Такой подход является развитием идей, предложенных в работе [14], в которой рассматривались только объекты, управляемые дискретными воздействиями.

В настоящей работе указанный подход обобщается на объекты, которые управляются с помощью не только дискретных, но и непрерывных воздействий. При этом в качестве примера объекта со сложным поведением рассмотрена модель беспилотного самолета, а в качестве режимов работы — разгон самолета и «мертвая петля».

Для объединения автоматов, управляющих режимами, с помощью метода сокращенных таблиц [13] или эвристического метода строится головной автомат, каждое из состояний которого соответствует одному из режимов. В результате формируется иерархическая система взаимодействующих автоматов. Вопрос о построении головного автомата в данной работе не рассматривается.

Для моделирования беспилотного самолета применяется свободный авиасимулятор *FlightGear* (http://www.flightgear.org), который позволяет осуществлять программное управление самолетом, а также сохранение параметров полета (скорость, направление полета и т. д.) и состояния самолета (положение руля, элеронов, состояние стартера и т. п.).

В результате проверки эффективности разработанного метода были получены автоматы, управляющие самолетом, который выполняет трюк «мертвая петля». Для этой задачи вычисление функции приспособленности для одного автомата занимало 0.2 секунды на одном ядре процессора *Intel Core 2 Duo T7250* с тактовой частотой 2 ГГц.

Приведем ссылки на некоторые видеозаписи полетов под управлением человека:

- успешное выполнение «мертвой петли» (этот полет вошел в обучающий набор) — http://www.youtube.com/watch?v=G5Kcx0ohpNo;
- неудачное выполнение — http://www.youtube.com/watch?v=OGVTcha97A.

Ниже приведены ссылки на видеозаписи трех вариантов реализации «мертвой петли» под управлением «лучшего» (по мнению авторов) из полученных автоматов:

- Выполнение «мертвой петли», близкое к «идеальному» (http://www.youtube.com/watch?v=TzrLoJjjVTA);
- Выполнение «мертвой петли» с заваливанием на левый борт с последующим выравниванием (http://www.youtube.com/watch?v=C6WV7x2bqE8);
- Последовательное выполнение двух «мертвых петель» (http://www.youtube.com/watch?v=yFiG4yz67Ks).

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2010. http://is.ifmo.ru/books/_book.pdf

2. *Angeline P., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. Cambridge: MIT Press. 1993. P. 154–163. http://www.demo.cs.brandeis.edu/papers/ep93.pdf

3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life /Proceedings of Second Conference on Artificial Life. MA: Addison-Wesley. 1992. P. 549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html

4. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» / Сборник трудов IV-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит, 2007. С. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf

5. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/ant

6. *Параценко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок». Проектная документация. СПбГУ ИТМО, 2006. http://is.ifmo.ru/unimod-projects/plates/

7. *Coates A., Abbeel P., Ng A. Y.* Learning for Control from Multiple Demonstrations / Proceedings of the 25th International Conference on Machine Learning. Helsinki, 2008. P. 144–151.

8. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.

9. *Рассел С., Норвиг П.* Искусственный интеллект: современный подход. М.: Вильямс, 2006.

10. *Koza J. R.* Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.

11. *Курейчик В. М.* Генетические алгоритмы. Состояние. Проблемы. Перспективы // Известия РАН. Теория и системы управления. 1999. № 1. С. 144–160.

12. *Курейчик В. М., Родзин С. И.* Эволюционные алгоритмы: генетическое программирование // Известия РАН. Теория и системы управления. 2002. № 1. С. 127–137.

13. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования // Известия РАН. Теория и системы управления. 2010. № 2. С. 100–117.

14. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5. С. 31–36.

МЕТОД ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ ВЕРХНЕГО УРОВНЯ ДЛЯ УПРАВЛЕНИЯ МОДЕЛЬЮ БЕСПЛОТНОГО САМОЛЕТА НА ОСНОВЕ ОБУЧАЮЩИХ ПРИМЕРОВ

С. В. Казаков, Ф. Н. Царев, А. А. Шалыто

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В последнее время для программирования систем со сложным поведением все шире применяется автоматное программирование, в рамках которого поведение программ описывается с помощью конечных детерминированных автоматов (в дальнейшем — автоматов) [1].

В автоматном программировании программы предлагается строить в виде набора автоматизированных объектов управления. Каждый такой объект состоит из объекта управления и системы управления (системы управляющих автоматов). Система автоматов получает на вход события из внешней среды и от объекта управления. На основании этих данных система управления вырабатывает выходные воздействия для объекта управления.

Для многих задач управляющие автоматы удается строить эвристически, но существуют задачи, для которых такое построение невозможно или затруднительно. К этому классу относятся, например, задача «Умный муравей» [2–4], задача «Умный муравей-3» [5] и задача об управлении моделью беспилотного летательного аппарата [6].

Существует несколько подходов к решению последней задачи. Один из них состоит в выделении «идеальной» траектории из нескольких полетов, выполненных человеком, и последующем следовании ей. Такой подход описан в работе [7].

Другой подход — использование автоматов для управления беспилотным летательным аппаратом и построение таких автоматов с помощью генетических алгоритмов, описанных в работах [8–12].

В работе [13] для генерации конечного автомата верхнего уровня, управляющего моделью беспилотного самолета, применяется алгоритм генетического программирования, основанный на использовании метода сокращенных таблиц для представления конечных автоматов. При этом вычисление функции приспособленности базируется на моделировании поведения самолета во внешней среде, которое занимает около 5 минут для одного автомата на одном ядре современного процессора.

С целью избавления от указанного недостатка в работе [14] было предложено использование обучающих примеров, которые записывает человек и которые можно использовать для замены моделирования. Результатами той

работы были конечные автоматы, которые могли управлять самолетом только в конкретном режиме.

Для обеспечения всего непрерывного процесса полета самолета предполагалось построение автомата верхнего уровня, каждое состояние которого соответствует одному из режимов полета.

В настоящей работе исследуется этот вопрос. Для построения автомата верхнего уровня используются обучающие примеры, которые содержат весь процесс полета самолета.

Благодаря использованию обучающих примеров как в процессе построения автомата верхнего уровня, так и в процессе построения автоматов нижнего уровня, появляется возможность объединить эти два процесса в один. Таким образом, достаточно задать набор обучающих примеров для всего полета.

Для моделирования беспилотного самолета применяется свободный авиасимулятор *FlightGear* (http://www.flightgear.org), который позволяет осуществлять программное управление самолетом, а также сохранение параметров полета (скорость, направление полета и т. д.) и состояния самолета (положение руля, элеронов, состояние стартера и т. п.).

Для экспериментального доказательства работоспособности разработанного метода была выбрана задача полета самолета, начиная с разгона и кончая приземлением, с возможностью выполнения трюка «мертвая петля» в полете. Возможность выполнения трюка означает, что трюк может быть выполнен при определенных (заранее заданных человеком) условиях. Например, если пришло подтверждение с Земли о разрешении выполнения трюка. Или, например, если погодные условия благоприятные.

В настоящее время обучающие примеры для поставленной задачи уже записаны и идет этап реализации и применения разработанного алгоритма, но автомат верхнего уровня ещё не получен.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2010. http://is.ifmo.ru/books/_book.pdf

2. *Angeline P., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. Cambridge: MIT Press. 1993. P. 154–163. http://www.demo.cs.brandeis.edu/papers/ep93.pdf

3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life / Proceedings of Second Conference on Artificial Life. MA: Addison-Wesley. 1992. P. 549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html

4. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» / Сборник трудов IV Международной научно-практической конференции «Интегрированные модели и мягкие

вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007. С. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf

5. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/ant

6. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок». Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates/

7. *Coates A., Abbeel P., Ng A. Y.* Learning for Control from Multiple Demonstrations / Proceedings of the 25th International Conference on Machine Learning. Helsinki: 2008. P. 144–151.

8. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.

9. *Рассел С., Норвиг П.* Искусственный интеллект: современный подход. М.: Вильямс, 2006.

10. *Koza J. R.* Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.

11. *Курейчик В. М.* Генетические алгоритмы. Состояние. Проблемы. Перспективы // Известия РАН. Теория и системы управления. 1999. № 1. С. 144–160.

12. *Курейчик В. М., Родзин С. И.* Эволюционные алгоритмы: генетическое программирование // Известия РАН. Теория и системы управления. 2002. № 1. С. 127–137.

13. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования // Известия РАН. Теория и системы управления. 2010. № 2. С. 100–117.

14. *Александров А. В., Казаков С. В., Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* Применение генетического программирования на основе обучающих примеров для генерации конечных автоматов, управляющих объектами со сложным поведением // Научно-технический вестник СПбГУ ИТМО. 2011. № 2. С. 3–11.

ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ С ИСПОЛЬЗОВАНИЕМ ЭВОЛЮЦИОННЫХ СТРАТЕГИЙ

М. В. Буздалов

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Введение

В мире проводится большое число олимпиад по программированию. Они способствуют выявлению талантливых программистов среди школьников и студентов. Среди них можно отметить международную студенческую олимпиаду по программированию International Collegiate Programming Contest [1], проводимую Association for Computing Machinery, с развитой сетью отборочных соревнований, международную олимпиаду школьников по информатике [2], соревнования, проводимые компанией TopCoder [3], интернет-олимпиады по информатике и программированию [4] и многие другие.

На олимпиадах по программированию предлагается решить одну или несколько задач. Решением задачи является программа, написанная на одном из разрешенных языков программирования.

Программа тестируется на наборе тестов, неизвестных участникам. На работу программы накладываются определенные ограничения, такие как максимальное время выполнения и максимальный объем используемой памяти.

Решение считается прошедшим определенный тест, если оно при работе с ним не нарушило ограничений, завершилось без ошибок, и его ответ признан правильным. О конкретных видах задач и ограничениях можно прочитать, например, на сайте олимпиады [5].

Цель работы

Создание тестов для олимпиадных задач является сложным творческим процессом. В большинстве случаев, тесты создаются вручную или с помощью программ, генерирующих тесты по некоторому шаблону. Для некоторых задач такой способ генерации тестов приводит к тому, что набор тестов оказывается слабым, и в результате засчитывается множество неэффективных решений.

Цель работы — генерация тестов, на которых неэффективные решения работают как можно дольше. Качество таких тестов может выражаться количественно, что позволяет использовать для поиска качественных тестов методы оптимизации. В качестве метода оптимизации в работе используется (1+1)-эволюционная стратегия [6].

Метод генерации тестов рассматривается на примере задачи “Work for Robots” [7], размещенной на сервере Timus Online Judge [8].

Описание предлагаемого подхода

Для генерации теста выбирается решение, против которого нужно сгенерировать этот тест. При успешной генерации теста, им может быть «покрыт» целый класс решений, таким образом, при разумном выборе решений, можно сгенерировать достаточно полный набор тестов.

Исходный код решения модифицируется, чтобы, помимо ответа на решаемую задачу, решение вычисляло функцию приспособленности теста. Функция приспособленности может быть как числом, так и более сложным объектом, например, вектором чисел, который может сравниваться с другим таким вектором лексикографически. Такой подход обладает большой гибкостью, так как позволяет описывать достаточно разнообразные функции приспособленности, которые, в свою очередь, могут ускорить процесс нахождения оптимума.

Для генерации теста используется $(1+1)$-эволюционная стратегия, особью которой является тест, а функция приспособленности этого теста вычисляется при запуске на нем модифицированного решения. Для задачи “Work for Robots”, используемой для демонстрации подхода, тест задается симметричной матрицей булевых значений размером 50×50 — матрицей смежности графа, который содержится в тесте.

Используется два оператора мутации. Первый из них изменяет содержимое случайно выбранной ячейки матрицы (и симметричной ей ячейки) на противоположное, второй изменяет 10, 100 или 1000 таких ячеек. Операторы мутации применяются по очереди.

Результаты

До применения описанного подхода, для задачи “Work for Robots” было засчитано 86 решений. Тесты генерировались против восьми из этих решений. По итогам генерации было выбрано пять тестов, которые оказались трудными для семи из восьми выбранных решений. Эти тесты были добавлены в набор тестов на сервер Timus Online Judge. 45 из имевшихся решений не прошли новые тесты. Этот результат показывает высокую эффективность описанного метода.

Литература

1. ACM International Collegiate Programming Contest. http://en.wikipedia.org/wiki/ACM_ICPC.

2. International Olympiad in Informatics. http://www.ioinformatics.org

3. TopCoder. http://www.topcoder.com/tc

4. Интернет-олимпиады по информатике. http://neerc.ifmo.ru/school/io/

5. Правила проведения полуфинала NEERC. http://neerc.ifmo.ru/information/contest-rules.html

6. *Bäck T., Hoffmeister F., Schwefel H.-P.* A Survey of Evolution Strategies. In Proceedings of the Fourth International Conference on Genetic Algorithms. 1991. Pp. 2–9.

7. Задача “Work for Robots”. http://acm.timus.ru/problem.aspx?num=1695

8. Timus Online Judge. Архив задач с проверяющей системой. http://acm.timus.ru

ГЕНЕРАЦИЯ КОНЕЧНЫХ АВТОМАТОВ С ПОМОЩЬЮ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧ НАВИГАЦИИ*

М. В. Буздалов

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Введение

Задачи навигации возникают в различных отраслях современной науки, включая робототехнику. На практике часто возникают задачи о перемещении робота из одной точки в другую в различном пространстве конфигураций.

В данном исследовании рассматриваются роботы, находящиеся в двумерной области с препятствиями. Робот должен добраться из стартовой точки в целевую, не пересекая препятствия. При этом он знает свои текущие координаты и координаты цели, а также способен хранить крайне небольшой объем данных (несколько чисел). Робот получает информацию об окружающей среде от контактных сенсоров.

Существуют алгоритмы, решающие данную задачу — алгоритмы семейства *Bug* [1–4]. В данном исследовании исследуется возможность *автоматического* построения управляющего конечного автомата для робота, который решает описанную задачу навигации. Построенный автомат может применяться в качестве части управляющей системы, построенной согласно парадигме автоматного программирования [5].

Постановка задачи

Упростим ранее сформулированную задачу поиска цели в области с препятствиями. Определим область как бесконечное клетчатое поле. Некоторые клетки этой области заняты препятствиями, причем таких клеток конечное число. Цель находится в одной из клеток, не занятых препятствием. Агент также занимает одну клетку. Агент может перемещаться только в клетки, смежные по стороне с текущей и не занятые препятствиями.

Агент знает свои координаты, а также координаты цели. Будем также считать, что у агента есть ориентация в пространстве и, как следствие, выделенное направление «вперед».

Агент управляется конечным автоматом. Конечный автомат принимает ряд входных воздействий и формирует на их основе ряд выходных воздействий. Также у агента есть O(1) дополнительной памяти — помимо собствен-

* Работа выполнена при финансовой поддержке РФФИ, проект № 10 01 00654а.

ных координат, координат цели и информации об ориентации в пространстве, он может хранить координаты еще одной точки и число, определяющее ориентацию автомата. Доступ к координатам агента, цели и сохраненной точки осуществляется с помощью входных и выходных воздействий автомата.

Генетический алгоритм

В данном исследовании для автоматического построения конечного автомата применяется генетический алгоритм [6]. Функция приспособленности каждого конкретного автомата определяется исходя из того, как этот автомат проходит некоторый набор полей — тестов. В свою очередь, этот набор полей также эволюционирует параллельно с эволюцией популяции автоматов, таким образом, применяется коэволюция [7]. Цель применения коэволюции — обеспечить отсутствие сходимости генетического алгоритма к неверным решениям вследствие неполноты набора тестов, а также создание благоприятных условий для работы генетического алгоритма.

Особью генетического алгоритма является список деревьев решений [8], задающих конечный автомат. Для работы с деревьями решений используются генетические операторы, стандартные для генетического программирования.

Полученные результаты

С помощью описанного генетического алгоритма и метода генерации тестов, при котором цель всегда была достижима, был сгенерирован конечный автомат всего из трех состояний, решающий задачу при условии, что цель всегда достижима. Его граф переходов изображен на рис. 1.

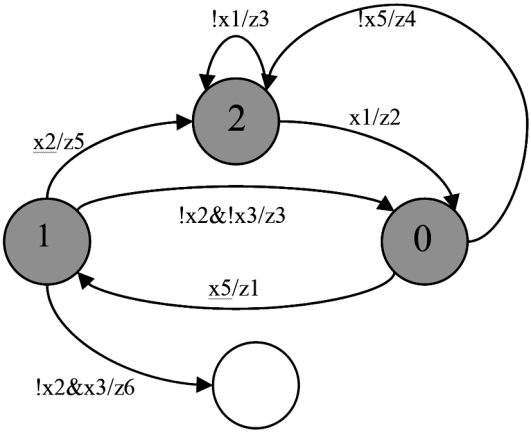


Рис. 1. Автомат для случая достижимой цели

Данный автомат весьма элегантно производит проверку всех необходимых условий, движение в сторону цели, когда это возможно, и обход препятствия. На рис. 2 продемонстрирована траектория движения агента, управляемого данным автоматом. На основе анализа траекторий агента, а также графа переходов управляющего автомата, можно доказать корректность реализованного алгоритма.

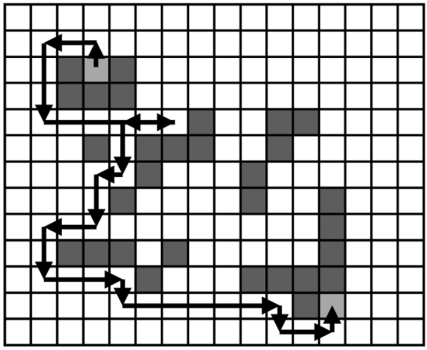


Рис. 2. Траектория движения агента, управляемого автоматом

Полученный алгоритм напоминает алгоритм *Distant Bug* при том условии, что в сведении задачи на клетчатое поле используется манхэттэнская метрика, в отличие от более традиционной евклидовой.

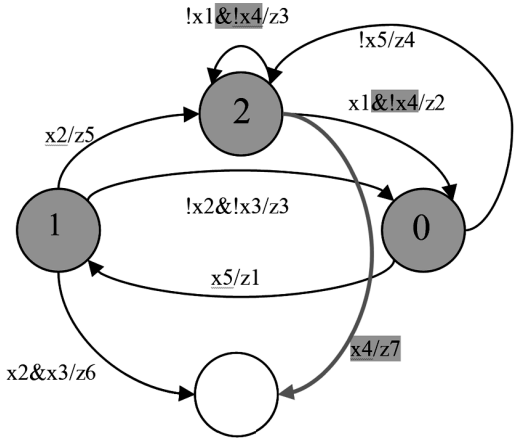


Рис. 3. Автомат для произвольного случая

Полученный автомат достаточно просто дополнить одним переходом и немного модифицировать условия в состоянии 2, чтобы получить полное решение задачи. Этот автомат изображен на рис. 3.

Заключение

Генетические алгоритмы с использованием коэволюции возможно применять для генерации конечных автоматов, используемых при решении тех задач навигации, где применима парадигма автоматного программирования. При этом полученные решения обладают аналитически доказуемой корректностью, а в процессе их поиска компьютер находит новые нетривиальные методы решения встречающихся задач.

Литература

1. *Lumelsky V. J., Stepanov A. A.* Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2: 403–430, 1987.

2. *Lumelsky V. J., Skewis T.* Incorporating range sensing in the robot navigation function. IEEE Transactions on Systems, Man, and Cybernetics, 20(5):1058–1068, 1990.

3. *Kamon I, Rimon E, Rivlin E.* A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots. CIS — Center of Intelligent Systems 9517, Computer Science Dept., Technion, Israel, 1995.

4. *Liu Y. H., Arimoto S.* Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. International Journal of Robotic Research, 11(4):376–382, 1992.

5. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2010.

6. *Holland John H.* Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.

7. *Hillis W. D.* Co-evolving parasites improve simulated evolution as an optimization procedure. Phys. D. Vol. 42. Issue 1–3 (Jun. 1990). Pp. 228–234.

8. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления со сложным поведением. СПбГУ ИТМО, 2007. Бакалаврская работа. http://is.ifmo.ru/papers/danilov_bachelor

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ПОСТРОЕНИЯ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ОСНОВЕ ВЕРИФИКАЦИИ МОДЕЛЕЙ И ОБУЧАЮЩИХ ПРИМЕРОВ

К. В. Егоров, Ф. Н. Царев

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Введение

Автоматное программирование — это парадигма программирования, в рамках которой программы предлагается проектировать в виде совокупности взаимодействующих автоматизированных объектов управления [1]. В автоматных программах выделяют три типа объектов: поставщики событий, система управления и объекты управления. Система управления представляет собой конечный автомат или систему взаимодействующих конечных автоматов. Поставщики событий генерируют события, а система управления по каждому событию может совершать переход, считывая значения входных переменных у объектов управления для проверки условия перехода.

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение затруднительно [2–4]. Одним из авторов настоящей работы был предложен метод построения автоматов с помощью генетического программирования на основе тестов [5]. Однако, как известно, тесты не могут полностью описывать поведение программы, а их выполнимость не может служить критерием ее корректности.

Целью настоящей работы является расширение возможностей метода построения автоматных программ на основе генетического программирования за счет использования верификации на этапе вычисления функции приспособленности, скрещивания и мутации.

Верификация автоматных программ

Для описания требований к автоматным программам будем применять язык логики линейного времени (*Linear Temporal Logic, LTL*). По любой LTL-формуле можно построить автомат Бюхи. Алгоритм верификации основан на проверке пустоты языка пересечения допускаемого конечным автоматом модели и отрицанием LTL-формулы [7, 8].

Верификатор получает на вход модель автоматной программы и LTL-формулу [9]. После проверки модели верификатор либо сообщает, что формула выполняется, либо приводит контрпример — путь в модели, опровергающий утверждение [10].

Построение управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования

При использовании метода построения управляющих конечных автоматов на основе обучающих примеров (тестов) каждый тест представляет собой последовательность входных воздействий (входная последовательность, соответствующая i-ому тесту, будет обозначаться как Input[i]) и соответствующую ей последовательность выходных воздействий (в дальнейшем будет обозначаться, как Answer[i]). Под входными воздействиями понимаются события от поставщиков событий и условия переходов, а под выходными — вызываемые действия объектов управления.

Функция приспособленности основана на редакционном расстоянии [6]. Для ее вычисления выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[i]. Обозначим последовательность выходных воздействий, которую сгенерировал автомат на входе Input[i] как Output[i]. После этого вычисляется величина FF_1:

$$FF_1 = \frac{\sum_{i=1}^{n}\left(1 - \frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)}\right)}{n}.$$

Здесь как ED(A, B) обозначено редакционное расстояние между строками A и B, как Answer[i] обозначена эталонная выходная последовательность, которую должен генерировать автомат на входе Input[i]. Отметим, что значения этой функции лежат в пределах от 0 до 1, при этом, чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Итоговое значение функции приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Она вычисляется по формуле

$$FF_2 = FF_1 + \frac{1}{M} \cdot (M - cnt).$$

Здесь cnt — число переходов в рассматриваемом конечном автомате, а M — некоторое число, большее максимально возможного числа переходов в автомате с заданным числом состояний.

Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF_1, отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньше переходов. Учет числа переходов в функции приспособленности необходим, так как минимизация числа переходов приводит к тому, что в результирующем автомате отсутствуют неиспользуемые в тестах переходы.

Совместное применение генетического программирования и верификации

Предлагается при вычислении функции приспособленности учитывать как поведение автомата при обработке тестов, так и число верных для автомата LTL-формул. При этом, чем больше число верных формул и успешно пройденных тестов, тем больше значение функции приспособленности.

Для вычисления функции приспособленности конечный автомат, задаваемый рассматриваемой особью, запускается на всех тестах и проверяется на соответствие всем темпоральным формулам, составляющим спецификацию. После этого вычисляется величина

$$\mathrm{FF} = \mathrm{FF}_1 \cdot (1 + \frac{n_1}{n_2}) + \frac{1}{M} \cdot (\mathrm{M} - \mathrm{cnt}).$$

Здесь как n_2 обозначено общее число темпоральных формул в спецификации, а как n_1 — число формул, которые являются верными для рассматриваемого конечного автомата.

Структура хромосомы в алгоритме генетического программирования

Конечный автомат в алгоритме генетического программирования представляется в виде объекта, который содержит описания переходов для каждого из состояний и номер начального состояния. Для каждого из состояний хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход выполняется и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода.

Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок, который аналогичен предложенному в работе [11].

Идея алгоритма расстановки пометок состоит в том, что генетическим алгоритмом строится только «скелет» конечного автомата, а пометки на переходах — вырабатываемые на них выходные воздействия — расставляются на основе тестов. При этом расстановка пометок происходит таким образом, чтобы получившийся в результате автомат как можно лучше «соответствовал» тестам.

Опишем более формально алгоритм расстановки пометок на переходах, применяемый в настоящей работе. Подадим на вход конечному автомату последовательность событий, соответствующую одному из тестов, и будем наблюдать за тем, какие переходы выполняет автомат. Зная эти переходы и информацию о том, сколько выходных воздействий должно быть сгенерировано на каждом из переходов, можно определить, какие выходные воздействия

должны вырабатываться на переходах, использовавшихся при обработке входной последовательности (рис. 1).

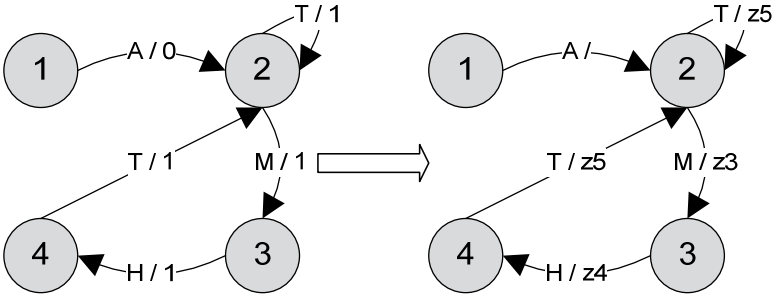


Рис. 1. Применение алгоритма расстановки пометок

Для случая нескольких тестов этот принцип можно обобщить следующим образом. Для каждого перехода T и каждой последовательности выходных воздействий zs вычисляется величина $C[T][\mathrm{zs}]$ — число раз, когда при обработке входной последовательности, соответствующей одному из тестов, на переходе T должны быть выработаны выходные воздействия, образующую последовательность zs. Далее, каждый переход помечается той последовательностью zs_0, для которой величина $C[T][\mathrm{zs}]$ максимальна.

Операции скрещивания и мутации

Операция мутации может выполняться двумя способами — традиционным и учитывающим результат верификации. Традиционный способ описан в работе [5] — он используется в методе построения управляющих автоматов на основе обучающих примеров. Операция скрещивания может быть осуществлена тремя способами — традиционным, с учетом тестов и с учетом результата верификации. Первые два метода также описаны в работе [5].

Опишем методы мутации и скрещивания, учитывающие результат верификации. Напомним, что алгоритм верификации основан на двойном обходе в глубину автомата модели и автомата Бюхи, построенного по отрицанию LTL-формулы. При использовании такого алгоритма, та часть модели, которая была посещена в процессе первого обхода в глубину, удовлетворяет LTL-формуле и может быть использована в процессе скрещивания точно так же, как в методе скрещивания с учетом тестов (помеченные переходы копируются в новые особи напрямую). Иными словами, подграф переходов, которые обошел верификатор в процессе верификации, может перейти без изменений в новую особь.

В то же время, мы можем не только сохранять часть модели, на которой выполняется темпоральное свойство, но и удалять те переходы, которые входят в контрпример, возвращаемый верификатором. Такой контрпример

представляет собой путь в модели, поэтому при мутации мы можем либо удалить переход из этого пути, либо изменить его конечное состояние, число генерируемых выходных воздействий или событие, инициирующее переход.

Экспериментальное исследование

Экспериментальное исследование предлагаемого метода машинного обучения проводилось на задаче построения автомата управления дверьми лифта. Эта система содержит пять возможных входных событий (e11 — нажата кнопка «Открыть двери»; e12 — нажата кнопка «Закрыть двери»; e2 — открытие или закрытие дверей успешно завершено; e3 — препятствие мешает закрыть дверь; e4 — дверь сломалась) и три выходные воздействия (z1 — начать открытие дверей; z2 — начать закрытие дверей; z3 — позвонить в аварийную службу).

При построении управляющего автомата использовались девять тестов (табл. 1) и 11 темпоральных свойств (табл. 2).

Таблица 1

Тесты для системы управления дверьми лифт

Входная последовательность	Выходная последовательность
e11, e2, e12, e2	z1, z2
e11, e2, e12, e2, e11, e2, e12, e2	z1, z2, z1, z2
e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2
e11, e2, e12, e2, e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
e11, e2, e12, e3, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
e11, e4	z1, z3
e11, e2, e12, e4	z1, z2, z3
e11, e2, e12, e2, e11, e4	z1, z2, z1, z3
e11, e2, e12, e3, e4	z1, z2, z1, z3

Таблица 2

Темпоральные свойства, составляющие спецификацию системы управления дверьми лифта

Формула	Комментарий
G(wasEvent(ep.e11) => wasAction(co.z1))	Если было событие e11, то было вызвано действие z1
G(wasEvent(ep.e12) <=> wasAction(co.z2))	Событие e12 обрабатывается тогда и только тогда, когда вызывается z2
G(wasEvent(ep.e4) <=> wasAction(co.z3))	Событие e4 обрабатывается тогда и только тогда, когда вызывается z3

Формула	Комментарий
G(wasEvent(ep.e3) => wasAction(co.z1))	Если было событие e3, то было вызвано действие z1
G(wasEvent(ep.e2) => X(wasEvent(ep.e11) or wasEvent(ep.e12)))	Если было событие e2, то следующим обработанным событием будет e11 или e12
G(wasEvent(ep.e11) => X(wasEvent(ep.e4) or wasEvent(ep.e2)))	Если было событие e11, то следующим обработанным событием будет e4 или e2
G(wasAction(co.z1) => X(wasEvent(ep.e2) or wasEvent(ep.e4)))	Если было вызвано действие z1, то следующим обработанным событием будет e2 или e4
G(wasEvent(ep.e12) => X(wasEvent(ep.e2) or wasEvent(ep.e3) or wasEvent(ep.e4)))	Если было событие e12, то следующим обработанным событием будет e2, или e3, или e4
G(wasAction(co.z1) => X(U(!wasAction(co.z1), wasAction(co.z2) or wasEvent(ep.e4))))	Если было вызвано действие z1, то оно не будет больше вызвано, пока не будет вызвано z2 или обработано событие e4
G(wasAction(co.z2) => X(U(!wasAction(co.z2), wasAction(co.z1) or wasEvent(ep.e4))))	Если было вызвано действие z2, то оно не будет больше вызвано, пока не будет вызвано z1 или обработано событие e4
!F(wasEvent(ep.e4) and X(F(wasEvent(ep.e11) or wasEvent(ep.e12) or wasEvent(ep.e2) || wasEvent(ep.e3))))	Не верно, что в будущем будет после события e4 когда либо будут обработаны e11, e12, e2 или e3 (лифт не может самостоятельно починиться)

Целью экспериментального исследования было сравнение метода построения управляющих конечных автоматов на основе тестов с предлагаемым в настоящей работе методом, использующим верификацию моделей на различных стадиях генетического алгоритма (вычисление функции приспособленности, скрещивание и мутация). Для каждого метода было проведено 1000 экспериментов, для каждого из которых записывалось число вычислений функции приспособленности.

Эксперименты показали, что при построении автоматов только на основе тестов, очень редко (всего в девяти случаях из 1000) результатом являлся автомат, который полностью удовлетворяет спецификации. Пример автомата, построенного только на основе тестов, приведен на рис. 2.

Это автомат обладает тем недостатком, что может отдать команду на закрытие дверей после того, как они сломаются или же начать открывать (закрывать) двери, когда они уже открыты (закрыты).

При использовании предлагаемого метода (с применением верификации моделей) построение автомата (рис. 3) занимало больше времени, но построенный автомат удовлетворял всем требованиям спецификации.

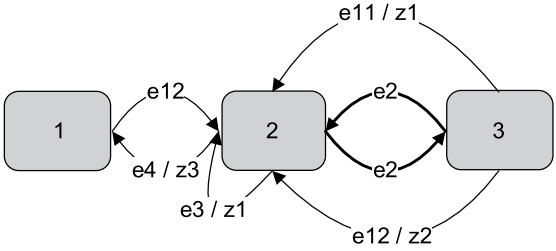


Рис. 2. Автомат управления дверьми лифта, построенный только на основе обучающих примеров

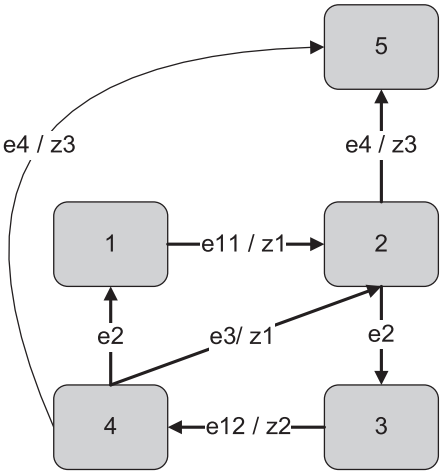


Рис. 3. Автомат управления дверьми лифта, построенный с использованием верификации

При построении конечного автомата управления дверьми лифта только на основе тестов, среднее значение вычислений функции приспособленности оказалось равным 7.479×10^4 (минимальное число вычислений — 2.184×10^4, максимальное — 2.999×10^5, среднеквадратичное отклонение — 2.54×10^4).

При использовании верификации моделей совместно с тестами, среднее значение числа вычислений функции приспособленности оказалось равным 8.372×10^5 (минимальное число вычислений — 6.331×10^4, максимальное — 5.912×10^6, среднеквадратичное отклонение — 7.57×10^5).

Таким образом, использование верификации хоть и замедляет процесс построения управляющего конечного автомата примерно в десять раз, но если принять во внимание то, что при построении только на основе тестов процент правильно построенных автоматов меньше 1 %, то совместное применение тестов и верификации оправдывает себя.

Заключение

В работе предложен метод машинного обучения для построения управляющих конечных автоматов на основе обучающих примеров. Предложенный метод основан на совместном применении генетического программирования и верификации моделей программ. Применение верификации в процессе построения автомата позволяет говорить об автоматизированном построении автоматов с гарантированным поведением.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009.

2. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. 1993. http://www.demo.cs.brandeis.edu/papers/ep93.pdf

3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. 1992. http://www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html

4. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.

5. *Царев Ф. Н.* Метод построения автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования / Материалы Международной научной конференции «Компьютерные науки и информационные технологии». Саратов: СГУ, 2009. С. 216–219.

6. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР 163.4. С. 845–848.

7. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002.

8. *Gerth R., Peled D., Vardi M. Y., Wolper P.* Simple On-the-fly Automatic Verification of Linear Temporal Logic / Proc. of the 15th Workshop on Protocol Specification, Testing, and Verification. Warsaw, 1995. Pp. 3–18.

9. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Второй этап. СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf

10. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. СПб: Политехника. 2008. № 5. С. 15–21.

11. *Lucas S., Reynolds T.* Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. Volume 11. Issue 3. June 2007. Pp. 308–325.

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ПОСТРОЕНИЯ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ОСНОВЕ КОНТРАКТОВ И ТЕСТОВЫХ ПРИМЕРОВ

К. В. Егоров, А. А. Шалыто

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Введение

Автоматное программирование — это парадигма программирования, в рамках которой программы предлагается проектировать в виде совокупности взаимодействующих автоматизированных объектов управления [1]. В автоматных программах выделяют три типа объектов: поставщики событий, система управления и объекты управления. Система управления представляет собой конечный автомат или систему взаимодействующих конечных автоматов. Поставщики событий генерируют события, а система управления по каждому событию может совершать переход, считывая значения входных переменных у объектов управления для проверки условия перехода.

Существуют различные способы построения автоматов управления со сложным поведением. Чаще всего такие системы строятся эвристически, но они могут содержать ошибки и требуют дополнительных проверок. В работе [2] был предложен способ построения автоматов с помощью генетического программирования на основе тестовых примеров. Однако такой вариант построения конечных автоматов требовал дополнительной валидации и верификации, а в случае обнаружения ошибки, необходимо было изменять тестовые примеры и заново строить систему. В любом случае, при таком подходе нельзя гарантировать поведение построенной системы на входных данных отличных от тестовых. В работе [3] было предложено использовать верификацию при вычислении функции приспособленности, однако вклад результата верификации был дискретным — 0 или 1. В работе [4] было предложено строить систему совместно на основе обучающих примеров и темпоральных формул. Формулы записываются на языке логики линейного времени (*Linear Temporal Logic, LTL*) и позволяют утверждать, что построенная система соответствует заявленной спецификации. Результат верификации учитывается при мутации, скрещивании и при вычислении функции приспособленности, причем вклад каждой темпоральной формулы — значение на отрезке [0, 1]. При этом 0 — формула нарушается сразу же в стартовом состоянии, а значение 1 — формула выполнятся.

Однако построение конечных автоматов на основе LTL-формул приводило к замедлению процесса построения. Кроме того, запись утверждений

на языке LTL оказалась сложной для неподготовленных пользователей. Допустив ошибку в формуле и не заметив ее, процесс построения автомата может никогда не завершиться, и сложно выявлять такие ошибки.

Автоматное программирование по контрактам

В настоящей работе предлагается вместо или совместно с LTL-формулами использовать контракты [5, 6]. Обычно выделяют три вида контрактов: предусловия, постусловия и инварианты. В классическом понимании «программирования по контрактам» в объектно-ориентированном программировании: *предусловие* — ожидание метода объекта на входные параметры и состояние класса при его вызове, *постусловие* — обязательства метода при его завершении, *инвариант* — условие выполняющиеся на протяжении всего времени жизни экземпляра класса. При автоматном подходе контракты могут накладываться как на состояния, так и на переходы и группы состояний.

При автоматическом построении автомата управления заранее не известно о состояниях конечного автомата, поэтому не представляется возможным использование контрактов для состояний или групп состояний. Определим контракты через LTL-формулы. Язык LTL состоит из пропозициональных переменных и стандартных булевых и специальных темпоральных операторов [7]. Для настоящей работы важны два из них:

- **X** (ne**X**t) — «*Xp*» — в следующий момент выполнено p;
- **G** (**G**lobally in the future) — «*Gp*» — всегда в будущем выполняется p.

Определим предусловие как $G(Xp_1 \rightarrow p_2)$ — если на следующем шаге выполнено p_1, то выполнено p_2; постусловие как $G(p_1 \rightarrow Xp_2)$ — если выполнено p_1, то на следующем шаге выполнено p_2; инвариант как $G(p_1 \rightarrow p_2)$ — если выполнено p_1, то выполнено p_2. Например, предусловие на переход можно определить как $G((!e \&\& Xe) \rightarrow p)$ или как $G(Xe \rightarrow p)$, постусловие как $G((e \&\& X!e) \rightarrow Xp)$ или как $G(e \rightarrow Xp)$, инвариант как $G(e \rightarrow p)$, где e — «переход по событию *e*», p — предикат.

По любой LTL-формуле можно построить автомат Бюхи. Алгоритм верификации основан на проверке пустоты языка пересечения допускаемого конечным автоматом модели и отрицанием LTL-формулы [7]. Можно показать, что для верификатора постусловие и предусловие оказываются одинаковыми, так как их отрицание представляются «похожими» автоматами Бюхи. В принципе, контрактом можно назвать любую LTL-формулу, отрицание которой приведет к заранее заданной структуре автомата Бюхи. Под «заранее заданной структурой» будем понимать недетерминированный конечный автомат, который эквивалентен автомату контракта с точностью до пометок на переходах. На рисунке представлены автоматы для предусловия (а), постусловия (б) и инварианта (в) на переходы.

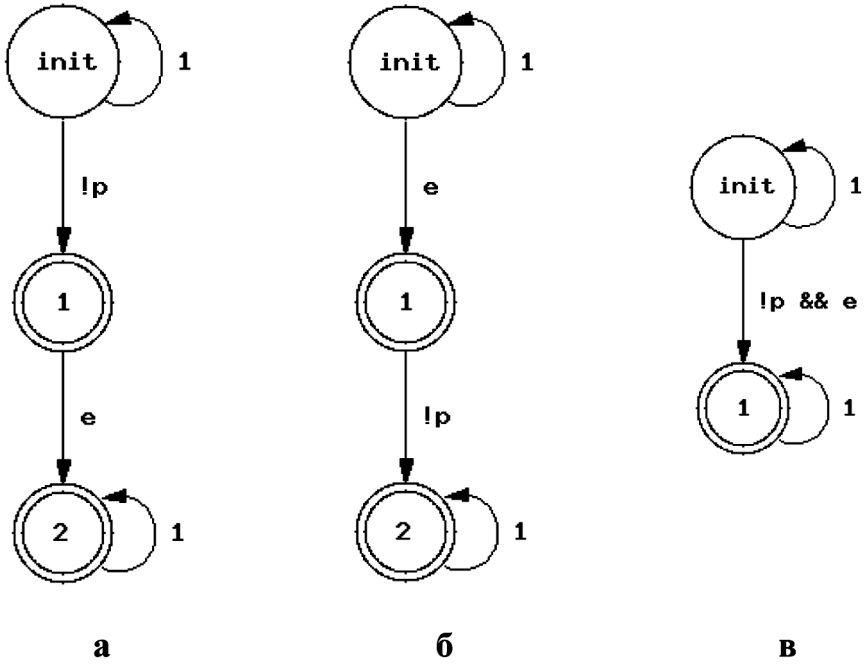


Рис. Автоматы Бюхи, построенные для отрицания LTL-формул G(Xe → p) (*а*), G(e → Xp) (*б*), G(e → p) (*в*)

Применение контрактов для построения автоматов управления

При верификации произвольных темпоральных свойств, заранее не известна их семантика. Это приводит к тому, что, обнаружив контрпример в автомате, невозможно определить, какой переход нарушает формулу. Когда конечный автомат строится на основе контрактов, точно известно, какие переходы в контрпримере нарушают его. Например, для инварианта последний переход нарушает его, а для предусловия и постусловия — последний и предпоследний. В результате такой априорной информации упрощается процесс модификации автомата с целью соблюдения контракта.

В настоящей работе предлагается генетический алгоритм построения конечных автоматов на основе тестовых примеров и контрактов. Особь в каждом поколении представляет собой конечный автомат, с событием на переходах и с указанием числа выходных воздействий [2]. Тестовые примеры позволяют расставлять выходные воздействия. Контракты позволяют проверять, что конечный автомат соответствует заявленной спецификации, и исключать из популяции особи, заранее ей не соответствующие, и модифи-

цировать автомат так, чтобы контракт соблюдался. Для этого выполнимость контракта оценивается как значение на отрезке [0, 1], и полученный результат вносит вклад в функцию приспособленности. Переходы, нарушающие контракт, с большей вероятностью подвергаются мутации и/или не участвуют в скрещивании.

Для того, что бы вклад контракта в функцию приспособленности не был дискретным (0 или 1), предлагается в процессе верификации помечать те переходы, которые верны и соответствуют спецификации. Алгоритм верификации LTL-формул основан на двойном обходе в глубину [7], поэтому в тот момент, когда при первом обходе алгоритм покидает состояние, все исходящие переходы можно считать соответствующими спецификации. Таким образом, в качестве вклада контракта в функцию приспособленности в простейшем случае можно взять отношение числа проверенных переходов к общему числу достижимых. Тем самым, чем больше число проверенных переходов, тем больше функция приспособленности.

Аналогично вычислению функции приспособленности, помеченные переходы можно учитывать при скрещивании двух особей: переходы, проверенные верификатором и на которых соблюдаются контракты, можно сохранить в новой особи без изменений. Таким образом, часть переходов в автомате уже будет соответствовать спецификации. Такое скрещивание позволяет сохранить ту часть автомата, на которой соблюдаются контракты, и приводит к росту функции приспособленности новой особи.

Мутация переходов может быть следующих типов: изменение входного воздействия, изменение числа выходных воздействий, изменение конечного состояния или удаление перехода. Как было отмечено выше, контракт нарушают два последних перехода в контрпримере, или же последний переход, в случае инварианта. Таким образом, увеличение вероятности мутации таких переходов, приводит к росту функции приспособленности.

Экспериментальные исследования

Были проведены экспериментальные исследования на примере автомата управления дверьми лифта из работы [4]. Было проведено 1000 построений для каждого из перечисленных выше методов. В процессе экспериментов популяция состояла из 2000 особей и использовалась стратегия элитизма, когда в новое поколение переходило 10 % лучших особей. В качестве оцениваемого параметра использовалось число вычислений функции приспособленности.

При использовании только тестовых примеров построенный автомат менее чем в 1 % случаев соответствовал спецификации, при использовании совместно верификации и тестовых примеров среднее число вычислений функции приспособленности оказалось равным — 827 857. При совместном использовании тестов и контрактов — 710 882.

Заключение

В работе исследована возможность применения контрактов для автоматического построения автоматов управления систем со сложным поведением, предложен метод мутации и скрещивания на основе выполнимости или не выполнимости контракта, проведено экспериментальное исследование метода при построении автомата управления дверьми лифта.

В результате проведенных экспериментов было показано, что применение контрактов приводит к ускорению построения автомата управления, и запись утверждений становится проще. Однако заметим, что контрактами не всегда можно полностью описать поведение системы, так что целесообразно использовать их совместно с LTL-формулами.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2010.

2. *Царев Ф. Н.* Метод построения автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования / Материалы Международной научной конференции «Компьютерные науки и информационные технологии». Саратов: СГУ, 2009. С. 216–219.

3. *Johnson C.* Genetic Programming with Fitness based on Model Checking. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2007. V. 4445. Pp. 114–124.

4. *Егоров К. В., Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации // Научно-технический вестник СПбГУ ИТМО. 2010. № 69. С. 81–85.

5. *Мейер Б.* Объектно-ориентированное конструирование программных систем. М.: Интернет-университет информационных технологий, 2005.

6. *Борисенко А., Федотов П., Степанов О., Шалыто А.* Разработка надежного программного обеспечения со сложным поведением / Сборник трудов конференции 5th Central and Eastern European Software Engineering Conference in Russia. М.: 2009. С. 125–128.

7. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002.

ПРИМЕНЕНИЕ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ О ВЫПОЛНИМОСТИ БУЛЕВОЙ ФОРМУЛЫ ДЛЯ ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ

В. И. Ульянцев, Ф. Н. Царев

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Введение

В последнее время все в более широком кругу задач начинает применяться автоматное программирование, в рамках которого поведение программ описывается с помощью детерминированных конечных автоматов [1].

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение автоматов затруднительно. К задачам этого класса относятся, в частности, задачи об «Умном муравье» [2–4], об управлении моделью беспилотного летательного аппарата [5].

Для построения автоматов в задачах такого типа успешно применяются генетические алгоритмы [6], в том числе на основе обучающих примеров [7]. Недостатком генетических алгоритмов является то, что время их работы весьма велико и его достаточно трудно оценить аналитически.

Целью настоящей работы является разработка метода построения управляющего конечного автомата, лишенного указанных недостатков.

Постановка задачи

Управляющим конечным автоматом будем называть детерминированный конечный автомат, каждый переход которого помечен *событием*, последовательностью *выходных воздействий* и *охранным условием*, представляющей собой логическую формулу от *входных переменных*.

Автомат получает события от так называемых *поставщиков событий* (в их роли могут выступать внешняя среда, интерфейс пользователя и т. д.) и генерирует выходные воздействия для *объекта управления*. При поступлении события автомат выполняет переход в соответствии с охранными условиями и значениями входных переменных. При выполнении перехода генерируются выходные воздействия, которыми он помечен, и автомат переходит в соответствующее состояние. Отметим, что состояния такого автомата не делятся на допускающие и не допускающие.

Формальное определение управляющего автомата дано в [1]. Пример управляющего автомата приведен на рисунке.

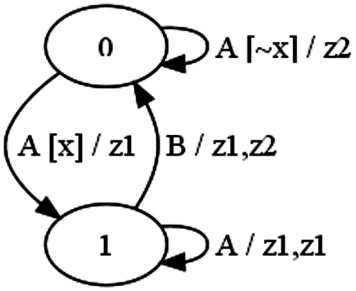


Рис. Пример управляющего автомата

Для данного автомата множество входных событий равно $\{A, B\}$, охранные условия зависят от единственной логической входной переменной x, множество выходных воздействий равно $\{z1, z2\}$. Далее, состояние автомата с номером 0 будем считать начальным.

В настоящей работе в качестве исходных данных для построения управляющего конечного автомата используется множество *сценариев работы*. Сценарием работы будем называть последовательность $T_1\$1\$\ldots\$1\$T_n$ троек $T_i = <e_i, f_i, A_i>$, где e_i — входное событие, f_i — булева формула от входных переменных, задающая охранное условие, A_i — последовательность выходных воздействий. В дальнейшем тройки T_i будем называть *элементами сценария*.

Будем говорить, что автомат, находясь в состоянии *state*, *удовлетворяет элементу сценария* T_i, если из *state* исходит переход, помеченный событием e_i, последовательностью выходных воздействий A_i и охранным условием, тождественно равным f_i как булева формула. Автомат *удовлетворяет сценарию работы* $T_1\$1\$\ldots\$1\$T_n$, если он удовлетворяет каждому элементу данного сценария, находясь при этом в состояниях пути, образованного соответствующими переходами.

В настоящей работе решается задача построения управляющего конечного автомата с заданным числом состояний C по заданному множеству сценариев работы Sc, которым автомат должен удовлетворять.

Описание предлагаемого метода

Построение управляющего автомата осуществляется в пять этапов:

1. Построение дерева сценариев.
2. Построение графа совместимости вершин дерева сценариев.
3. Построение булевой КНФ-формулы, задающей требования к раскраске построенного графа и выражающей непротиворечивость системы переходов результирующего автомата.
4. Запуск сторонней программы, решающей задачу о выполнимости булевой КНФ-формулы.

5. Построение автомата по найденному выполняющему набору значений переменных.

Экспериментальное исследование

Экспериментальное исследование проводилось на задаче построения автомата управления часами с будильником [1]. Было задано 38 сценариев, аналогичных тестам, приведенным в работе [7]. На основе этих сценариев был построен автомат, изоморфный автомату, построенному вручную в работе [1]. Его построение заняло менее секунды на персональном компьютере с процессором Intel Core 2 Quad Q9400, что позволяет говорить о достаточно высокой производительности разработанного метода.

Заключение

В настоящей работе предложен метод построения управляющих конечных автоматов по сценариям работы. Этот метод основан на сведении указанной задачи к задаче о выполнимости булевой формулы. Работоспособность метода проверена на задаче построения автомата управления часами с будильником. На этой задаче соответствующий управляющий автомат был построен корректно, а время работы алгоритма составляло меньше секунды на персональном компьютере с процессором Intel Core 2 Quad Q9400.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009.

2. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. 1993.

3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. 1992. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html

4. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.

5. *Царев Ф. Н., Шалыто А. А.* Гибридное управление беспилотными летательными объектами на основе автоматного программирования / 1-я Российская мультиконференция по проблемам управления. Сборник докладов четвертой научной конференции «Управление и информационные технологии». СПб ГЭТУ «ЛЭТИ». 2006. С. 138–144.

6. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.

7. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5. С. 31–36.

АВТОМАТИЧЕСКИЕ ДОКАЗАТЕЛЬСТВА АНАЛОГОВ ГИПОТЕЗЫ ЧЕРНИ—ПЭНА

С. Э. Вельдер

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В докладе рассматриваются некоторые задачи теории синхронизируемых автоматов. Проблемы, связанные с синхронизируемостью, встречаются в различных областях информационных технологий, таких как теория кодирования, робототехника, и др. Свойство синхронизируемости детерминированного конечного автомата в узком смысле означает существование *синхронизирующего слова* — последовательности символов (команд), переводящей все состояния автомата в одно и то же состояние. То есть синхронизирующее слово переводит множество всех состояний автомата в синглетон. Говорят, что слово *имеет дефект* k в заданном автомате из n состояний, если оно переводит множество всех его состояний (имеющее размер n) в множество размера $n-k$. Синхронизируемость автомата определяется существованием для него слова дефекта $n-1$.

Одной из основных и наиболее сложных задач в этой теории является оценка минимальной длины (в худшем случае) синхронизирующего слова или слова заданного дефекта при условии, что такое слово существует [1]. Длину минимального слова дефекта k в худшем случае обозначим через $L(k)$. Гипотеза Ж.-Э. Пэна (1978) гласит, что $L(k) = k^2$. Её частный случай при $k=n-1$ известен как гипотеза Я. Черни [2]. На функцию L известны следующие оценки (рис. 1):

$$k^2 \le L(k) \le k(k+1)(k+2)/6 - 1$$

(первое неравенство приводится в [1], второе — в [3]).

Функция называется *перечислимой снизу* (*сверху*), если её подграфик (надграфик) перечислим. Функция является вычислимой тогда и только тогда, когда она перечислима и снизу, и сверху.

Функция L перечислима снизу путём перебора всех автоматов. Данный факт означает следующее. Рассмотрим двуместный предикат $P(k, l) = «L(k) \le l»$: он обозначает высказывание «в любом автомате, имеющем слово дефекта не менее k, найдётся слово дефекта не менее k и длины не более l». Если $P(k, l)$ не выполняется для некоторых k и l, то это можно подтвердить предъявлением соответствующего автомата — он будет являться алгоритмически проверяемым сертификатом, опровергающим $P(k, l)$. Возможность перебирать такие сертификаты — это и есть перечислимость снизу функции L.

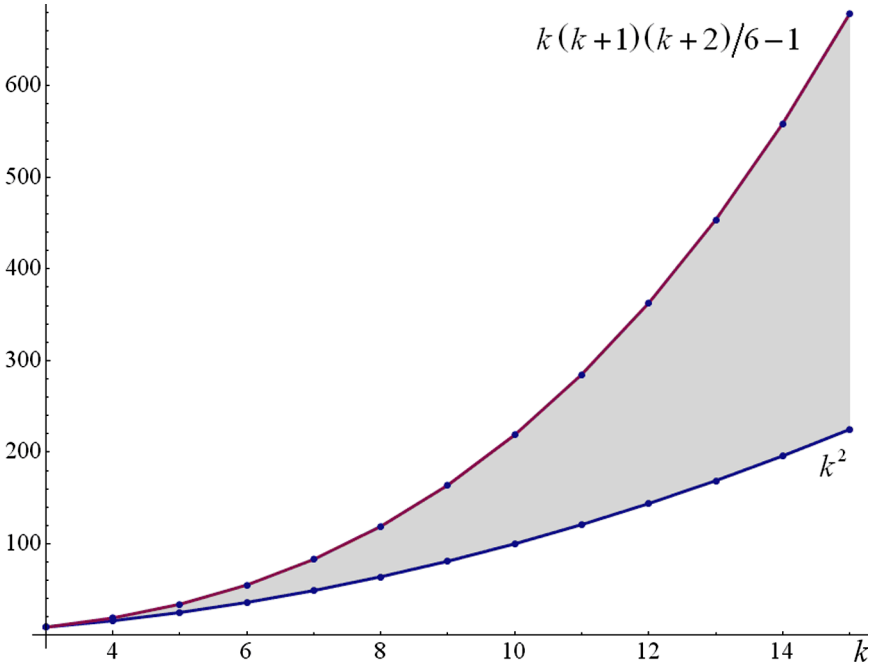


Рис. 1. Оценки функции Пэна $L(k)$

В случае же, когда $P(k, l)$ верно для некоторых k и l, этот факт является неочевидной теоремой и доказывается для каждых k и l по-своему. Действительно, это высказывание имеет квантификацию «для всех автоматов», а их бесконечно много.

Известны, например, следующие факты:

1) $P(0, 0) = «L(0) \le 0»$, тривиально;
2) $P(1, 1) = «L(1) \le 1»$, тривиально;
3) $P(2, 4) = «L(2) \le 4»$, нетрудно;
4) $P(3, 9) = «L(3) \le 9»$, Ж.-Э. Пэн [3];
5) $P(4, 16) = «L(4) > 16»$, Я. Кари [4] — последний пример (опровергающий гипотезу Пэна) был получен полным перебором автоматов.

Результаты, полученные в работе, состоят в построении алгоритмов, позволяющих автоматически генерировать доказательства утверждений такого вида (когда они верны), а также определять случаи, когда они неверны, не перебирая все автоматы. Наличие такого алгоритма, в частности, доказывает, что функция L перечислима и сверху (т. е. вычислима).

Ключевая идея алгоритма состоит в следующем. Введём специальную комбинаторную структуру данных, которая будет иметь конечный размер

и кодировать разбиения (бесконечного) множества всех автоматов на конечное число (бесконечных) классов эквивалентности. Процесс выполняется итеративно: сначала выполняется разбиение на небольшое число классов, потом каждый из полученных классов разбивается ещё на несколько и т. д. (рис. 2). Разбиения строятся на основе классификации наборов одноместных операций на конечном множестве. Каждая такая операция порождает ориентированный граф с постоянной исходящей степенью 1 (это совокупность циклов, оснащённых деревьями).

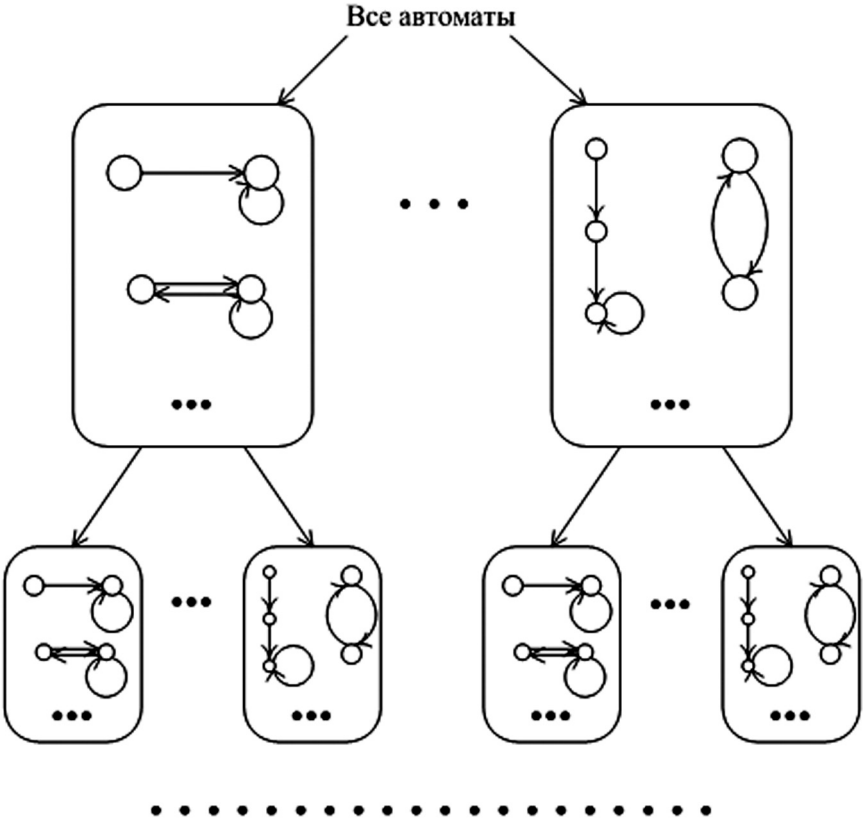


Рис. 2. Разбиение конечных автоматов на классы

После этого выполняем перебор всех классов, и для каждого из них генерируем доказательство искомого утверждения. Доказательство основывается на анализе так называемого *экспоненциального автомата* (*Exp*-автомата) — автомата, состояния которого являются множествами состояний исходного. Каждый автомат (класс автоматов) индуцирует свой экспоненциальный ав-

томат (класс автоматов). Требуется доказать, что если в экспоненциальном классе автоматов существует путь из стартовой вершины в вершину дефекта не менее k, то существует и аналогичный путь длины не более l. Таким образом, в качестве доказательства для каждого класса автоматов предъявляется либо соответствующий путь в экспоненциальном автомате, либо сам экспоненциальный автомат, в котором нет достижимой вершины дефекта не менее k. Этот автомат используется в качестве сертификата отсутствия соответствующего пути.

Описанная регулярная структура, кодирующая рассматриваемое разбиение (вместе с доказательствами для каждого класса) играет роль автоматически проверяемого сертификата, подтверждающего истинность $P(k, l)$ для заданных k и l. Корректность алгоритма обусловлена наличием таких сертификатов.

Литература

1. *Pin J.-E.* On two combinatorial problems arising from automata theory, Annals of Discrete Mathematics 17 (1983), 535–548.

2. *Černý J.* Poznámka k homogénnym eksperimentom s konečnými automatami, Matematicko-fyzikalny Časopis Slovensk. Akad. Vied, 14. No. 3, 208–216.

3. *Pin J.-E.* Le problème de la synchronisation et la conjecture de Černý, Quaderni de la Ricerca Scientifica vol. 109, CNR, Roma, 1981.

4. *Kari J.* A counter example to a conjecture concerning synchronizing words in finite automata, EATCS Bull., 73, 146.

ОБРАБОТКА СТРОК НА ОСНОВЕ СУФФИКСНЫХ АВТОМАТОВ

Д. А. Паращенко, А. С. Станкевич

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В настоящее время для решения большого числа строковых задач применяются суффиксные деревья [1]. При этом все известные алгоритмы построения суффиксного дерева за линейное время [1] достаточно сложны для понимания и реализации.

В настоящей работе разработан достаточно простой алгоритм построения суффиксного дерева за линейное время, содержащий в качестве одного из своих этапов построение суффиксного автомата. Таким образом, помимо алгоритмов Вайнера [2], Мак-Крейта [3] и Укконена [4] предложен еще один алгоритм построения суффиксного дерева за линейное время. Кроме этого в работе проведено сравнение времени их работы, а также сложности их реализации.

Поясним, как появилась идея использовать суффиксный автомат для решения рассматриваемой задачи. Ввиду того, что суффиксные деревья и суффиксные автоматы являются родственными структурами данных (суффиксный автомат является минимизированным суффиксным бором [5], а суффиксное дерево — сжатым суффиксным бором [5]), то автор посчитал целесообразным для решения строковых задач вместо суффиксных деревьев использовать суффиксные автоматы. Одним из преимуществ этого подхода является тот факт, что суффиксный автомат является более простой структурой данных [6], а алгоритм его построения значительно проще в реализации, чем алгоритм построения суффиксного дерева.

Разработанный автором алгоритм построения суффиксного дерева за линейное время немного уступает другим алгоритмам построения суффиксного дерева (Укконена и Мак-Крейта) как по количеству используемой дополнительной памяти, так и по времени работы. Однако он реализуется намного проще и быстрее упомянутых выше алгоритмов. Это позволяет применять его в случаях, когда требуется быстро написать правильно работающий код, затратив на это минимум усилий, например, на соревнованиях ACM ICPC.

В работе также предложен метод, позволяющий избавиться от использования в алгоритмах суффиксных деревьев. Суть этого метода состоит в изменении алгоритма таким образом, чтобы вместо операций над суффиксным деревом в нем использовались операции над суффиксным автоматом. В связи с тем, что суффиксный автомат является более простой в использовании структурой данных по сравнению с суффиксным деревом, применение данного метода позволяет в большинстве случаев ускорить работу алгоритма.

Опишем суть упомянутого метода. Для этого введем некоторые определения и приведем ряд их свойств.

Определение. Правым контекстом [5] строки y в строке w называется множество всех таких строк x, что строка xy является суффиксом строки w.

Определение. *Множеством представителей* правого контекста C в строке w будем называть множество строк, правый контекст которых в строке w совпадает с C.

Лемма 1. Множество представителей правого контекста состоит из суффиксов наибольшего представителя, длина которых не меньше длины наименьшего представителя.

Лемма 2. Для любого правого контекста C и числа *len* существует не более одного слова y длины *len* с правым контекстом C.

Заметим, что можно говорить о представителях состояния суффиксного автомата, подразумевая под этим представителей соответствующего этому состоянию правого контекста.

Лемма 3. Множество всех строк, приводящих автомат из начального состояния в некоторое состояние s, совпадает с множеством представителей правого контекста этого состояния.

Состояние суффиксного автомата, в которое ведет суффиксная ссылка из состояния s, обозначим за $suffix(s)$, а длину наибольшего представителя состояния s — за $repr_{\max}(s)$.

Теорема 1. Пусть непустое слово x является некоторым представителем состояния s_1 суффиксного автомата. Пусть s_2 — состояние, в которое придет суффиксный автомат, приняв на вход слово $x[2..|x|]$. Тогда:

- если $|x| = repr_{\max}(suffix(s_1)) + 1$, то s_2 совпадает с $suffix(s_1)$;
- если $|x| > repr_{\max}(suffix(s_1)) + 1$, то s_2 совпадает с s_1.

Для эмуляции суффиксного дерева необходимо установить связь между его состояниями и состояниями суффиксного автомата.

Теорема 2. Каждой вершине суффиксного дерева можно поставить в соответствие пару <*состояние суффиксного автомата, длина представителя этого состояния*>.

Следует понимать, что некоторым состояниям суффиксного автомата не будет соответствовать ни одна вершина суффиксного дерева. Указанные состояния обладают двумя свойствами: не являются конечными и при этом имеют лишь один исходящий переход. Все остальные состояния суффиксного автомата будем называть *вилками*.

Построив структуру, позволяющую эффективно переходить по переходам суффиксного автомата до ближайшей вилки, а также вычислив длины

наибольших представителей состояний можно эффективно реализовывать на автомате все операции суффиксного дерева:

- получать корневую вершину дерева;
- определять, является ли некоторая вершина суффиксного дерева конечной;
- получать суффиксную ссылку из данной вершины (см. теорему 1);
- находить исходящую из данной вершины дугу с заданным первым символом дуговой метки.

На практике, значительная часть задач требует далеко не всех операций, предоставляемых суффиксным деревом. Это позволяет при переносе алгоритма на суффиксный автомат обойтись без вычисления длин наибольших представителей состояний, а в некоторых случаях — и без вычисления вилок.

Для демонстрации эффективности предложенного метода в работе рассмотрена задача о нахождении наибольшего общего префикса двух суффиксов [7]. Основной причиной выбора этой задачи является то, что к ней можно свести большое число других задач на строках. Установлено, что применение метода позволяет значительно сократить время фазы инициализации алгоритма для решения рассмотренной задачи.

В дальнейшем планируется выяснить, что выгоднее: сразу строить сжатый суффиксный автомат, или строить, а затем сжимать суффиксный автомат. Также планируется произвести сравнение количества памяти, требуемой для хранения суффиксного дерева, и памяти, требуемой для хранения суффиксного автомата с поддержкой всех либо лишь части операций суффиксного дерева.

Литература

1. *Гасфилд Д.* Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. СПб.: Невский диалект; БХВ-Петербург, 2003.

2. *Weiner P.* Linear pattern matching algorithms / Proc. of the 14th IEEE Symp. on Swithing and Automata Theory. 1973. Pp. 1–11.

3. *McCreight E. M.* A space-economical suffix tree construction alorithm // J. ACM. 1976. Vol 23. Зр. 262–272.

4. *Ukkonen E.* Online construction of suffix-trees // Algoritmica. 1995. Vol. 14. Pp. 249–260.

5. *Lothaire M.* Applied Combinatorics on Words // Encyclopedia of Mathematics and its Applications, 2005. Vol. 90. Cambridge University Press, Cambridge.

6. *Хопкрофт Дж., Мортвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

7. *Bender M., Farach-Colton M.* The LCA Problem Revisited / LATIN 2000. Pp. 88–94.

СУФФИКСНЫЕ АВТОМАТЫ С СОХРАНЕНИЕМ ПРОМЕЖУТОЧНЫХ ВЕРСИЙ И ИХ ПРИЛОЖЕНИЯ

Д. А. Паращенко, А. С. Станкевич

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Структуры данных, которые хранят свои промежуточные версии, имеют более широкий круг приложений, чем их варианты без такой возможности [1–4]. Классическим примером является задача нахождения области, содержащей заданную точку [1]. Она может быть элегантно решена с помощью применения дерева поиска с сохранением промежуточных версий.

Не существует общего механизма, позволяющего после совершения какие-либо модификаций над обычной структурой данных вернуться к ее предыдущей версии. Структуры данных, хранящие свою хронологию, называются персистентными.

Существует несколько видов персистентных структур данных [1]. Частичная персистентность позволяет модифицировать последнюю версию структуры данных и делать запросы к любой из промежуточных версий. Полная персистентность позволяет совершать запросы и модификации с любой версией структуры данных. При таком типе персистентности версии образуют не линейную, а древовидную структуру. В обоих случаях доступ может осуществляться по номеру требуемой версии.

В настоящее время для решения множества строковых задач используются суффиксные деревья, суффиксные массивы и суффиксные автоматы [5–9]. Также известны методы, позволяющие делать суффиксные деревья персистентными. В свете работы [10] возникает вопрос о возможности эффективной реализации персистентных суффиксных автоматов.

Рассмотрены существующие методы модификаций структур данных с целью добавления в них возможности сохранения промежуточных версий применительно к суффиксному автомату. Предложен метод, позволяющий с небольшими дополнительными затратами сделать суффиксный автомат персистентным.

Свойство персистентности позволяет суффиксному автомату поддерживать помимо операции добавления символа операцию удаления последнего символа. При этом последняя операция не вызывает никаких дополнительных временных затрат.

Кроме того, персистентность позволяет использовать суффиксные автоматы в многопоточных приложениях, в которых предъявляются особые требования к быстродействию. Применение персистентной структуры данных позволяет выполнять операции по модификации суффиксного автомата

без блокировки операций чтения. В случае суффиксных автоматов это является достоинством, так как модификация суффиксного автомата может занимать до $O(n)$ времени [6].

В работе выполнен анализ существующих методов построения персистентных структур данных, а также разработан эффективный метод построения персистентного суффиксного автомата. При использовании предлагаемого метода доступ к прежним версиям суффиксного автомата в большинстве случаев не требует значительных дополнительных временных затрат, однако, для некоторых видов строк верхняя оценка времени выполнения запросов может увеличиться в $O(\log n)$ раз.

Предложен метод, позволяющий поддерживать множество допускающих состояний суффиксного автомата в процессе его построения. При этом требуется $O(\log n)$ дополнительного времени и $O(1)$ дополнительной памяти на каждый символ базовой строки суффиксного автомата. В связи с тем, что описанный метод основывается на использовании древовидной структуры данных, можно без дополнительных затрат хранить промежуточные версии множества допускающих состояний суффиксного автомата, тем самым, сделав упомянутое множество персистентным.

Литература

1. *Karger D.* Persistent Data Structures. Advanced Algorithms: Lecture 2. September 9, 2005.

2. *Bentley J. L., Saxe J. B.* Decomposable searching problems I: Static-to-dynamic transformations // J. Algorithms. 1980. Vol 1. Pp. 301–358.

3. *Chazelle B.* Filtering search: A new approach to query-answering // SIAM J. Comput. 1986. Vol. 15. Pp. 703–724.

4. *Chazelle B.* How to search in history // Inform. and Control. 1985. Vol 77. Pp. 77–99.

5. *Гасфилд Д.* Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. СПб.: Невский диалект; БХВ-Петербург, 2003.

6. *Lothaire M.* Applied Combinatorics on Words // Encyclopedia of Mathematics and its Applications, 2005. Vol. 90. Cambridge University Press, Cambridge.

7. *Хопкрофт Дж., Мортвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

8. *Кормен Т., Лейзерсон Ч., Ривест Л.* Алгоритмы: построение и анализ. М.: МЦНМО, 2000.

9. *Sartaj Sahni Dr.* Data Structures, Algorithms, & Applications in Java. Suffix Trees. CISE Department Chair at University of Florida http://www.cise.ufl.edu/\~sahni/dsaaj/enrich/c16/suffix.htm

10. *Паращенко Д.* Обработка строк на основе суффиксных автоматов. Бакалаврская работа. СПбГУ ИТМО, 2007.

ПРИМЕНЕНИЕ СИСТЕМ ТИПОВ ДЛЯ ВАЛИДАЦИИ И ВЕРИФИКАЦИИ АВТОМАТНЫХ ПРОГРАММ

Я. М. Малаховски

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Валидация и верификация являются ключевыми требованиями при разработке ответственных систем. Автоматное программирование [1] позволяет значительно упростить процесс верификации, поскольку переход от автоматной программы к модели Крипке может быть произведен автоматически и изоморфно [2]. В работе [3] был предложен подход к реализации событийных конечных автоматов [1] на функциональных языках программирования и показаны его преимущества перед традиционными реализациями. При этом для валидации функций переходов использовались свойства алгебраических типов данных. В работе [4] было предложено обобщение данного подхода на структурные автоматы с переменными. Однако валидация производилась не в процессе компиляции, а в процессе конструирования автомата во время работы программы.

Валидация отдельного автомата обычно не требует много времени, поскольку, на практике, размер формул на дугах графа переходов весьма мал. При этом время валидации системы автоматов есть сумма времен валидации каждого автомата в отдельности. Таким образом, добавление нового автомата не сильно увеличивает задержку при запуске программы, а потому результаты работы [4] являются пригодными для практического использования. Однако, для верификации данный подход не применим, поскольку время верификации системы автоматов есть функция от произведения мощностей множеств состояний каждого автомата. Таким образом, верификация программы должна быть частью процесса ее сборки и не может производится в процессе исполнения программы. Недостатком существующих средств верификации [2] является то, что спецификация программы, выраженная на языке, понятном верификатору, не является частью самой программы.

В настоящей работе предлагается метод верификации автоматных программ с использованием темпоральной логики CTL, производимый во время компиляции компилятором языка на котором выражены как автоматная программа, так и ее спецификация.

Разработанный подход основан на использовании зависимых систем типов в комбинации со встроенными предметно-ориентированными языками программирования (eDSL). При этом, при помощи встроенных предметно-ориентированных языков выражаются автоматные программы и спецификации на языке CTL, а проверка того, что автоматная программа удовлетворяет

спецификации производится при помощи доказательства утверждений в зависимой системе типов функционального языка программирования.

Для апробации и практического применения предложенного подхода был выбран язык программирования Agda (версии 2). eDSL для описания автоматов был, насколько это возможно, перенесён из работы [4].

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер. 2010. 176 с.

2. *Вельдер С. Э., Лукин М. А. Шалыто А. А., Яминов Б. Р.* Верификация автоматных программ. СПб.: ИТМО, 2011.

3. *Малаховски Я. М., Шалыто А. А.* Реализация конечных автоматов на функциональных языках программирования //Информационно-управляющие системы. 2009. № 6. С. 30–33.

4. *Малаховски Я. М., Корнеев Г. А.* Валидация автоматов с переменными на функциональных языках программирования //Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2010. № 6. С. 73–77.

ГЕНЕРАЦИЯ КЛЕТОЧНЫХ АВТОМАТОВ НА ОСНОВЕ ОБУЧАЮЩИХ ПРИМЕРОВ ПРИ ПОМОЩИ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

А. В. Тихомиров

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Краткое вступление, постановка проблемы

В настоящее время широко распространено использование клеточных автоматов для симуляции многих физических процессов, например: диффузия энергии, разнообразные химические реакции, рост кристаллов и т. д. [1–4]. Однако использование клеточных автоматов затрудняется тогда, когда физическая система известна, а описывающий ее клеточный автомат нет, или построение его обычными эвристическими методами затруднительно, так как автомат может иметь большое число состояний, переходов, условий и действий на переход [5].

Цель настоящей работы — устранить этот недостаток, используя генетическое программирование.

Цель работы

Целью настоящей работы является разработка алгоритма автоматической генерации клеточных автоматов с заранее неизвестными числом переходов и условий на них при помощи генетического программирования.

Постановка задачи

Задана двумерная плоскость определенной размерности с координатной сеткой, которая делит плоскость на квадраты [6, 7]. При этом задаются данные для различных временных шагов, т. е. состояние системы в начале, несколько промежуточных состояний и конечное состояние системы.

Каждая ячейка плоскости управляется одинаковым автоматом.

Цель поставленной задачи — вырастить клеточный автомат, который наиболее точно описывает заданную физическую систему.

Базовые положения исследования

Классический генетический алгоритм представляет собой эвристический метод поиска и оптимизации решений для задач моделирования, используя

операции генерации случайного решения, скрещивания (комбинирование нескольких решений) и мутации (случайное изменение части известного решения). Общий принцип работы классического генетического алгоритма напоминает биологическую эволюцию.

Промежуточные результаты

В работе предложен улучшенный вариант классического генетического алгоритма для достижения поставленной цели.

В работе освещены следующие аспекты алгоритма:

- структура хромосомы клеточного автомата;
- генерация начального поколения;
- генетические операторы;
  - операторы скрещивания;
  - операторы мутации;
  - дополнительные операторы;
- проблема вырождения популяции особей и методы ее решения;
- вычисление функции приспособленности и влияние на нее размерности клеточного автомата;
- алгоритм отбора нового поколения и исследование его влияния на время работы алгоритма.

Проведено сравнение скорости работы алгоритма со временем полного перебора всех возможных вариантов автоматов.

Полученные результаты

Для тестового примера была поставлена задача вырастить клеточный автомат, зная только состояния клеток на каждом конкретном шаге расчетов. Эталонный клеточный автомат, сделанный для проверки, содержит 5 состояний, 8 переходов и 8 условий на переходах. Система в среднем выращивает автомат, который является эквивалентным искомому за 700–1000 поколений, при этом среднее количество перебранных клеточных автоматов составляет 30 000–45 000.

Для ускорения процесса генерации было сделано допущение, что автоматы, у которых из какого-либо состояния истинно условие у нескольких переходов, являются корректными. На них было наложено условие, что при нескольких вариантах перехода, переход осуществляется в состояние с меньшим индексом.

После упрощения полученного клеточного автомата и сравнения его с эталоном было установлено, что эти автоматы являются идентичными.

Получившийся клеточный автомат содержит избыточные переходы, по которым никогда не происходит перехода. Однако при исследовании

было установлено, что если задавать условие минимизации в генетических операторах и выборе нового поколения, то время работы системы сильно увеличивается.

Быстродействие предложенного способа сильно зависит от начальных настроек, таких как пропорции между генетическими операциями, величина начальной популяции и многими другими.

Заключение

В работе предложен метод генерации клеточных автоматов произвольного количества состояний. Он позволяет автоматически получать клеточные автоматы, которые довольно точно описывают данные о физической системе на входе. Также этот метод позволяет получать клеточные автоматы, которые получить эвристическими методами затруднительно

Литература

1. *Тоффоли Т., Марголус Н.* Машины клеточных автоматов. Мир, 1991.

2. *Frish U.* Lattice gas hydrodynamics in two and three dimensions // Complex Systems. 1987. V. 1. P. 649–707.

3. *Wolfram S.* Cellular automation Fluids // J. Stat. Phys. 1986. V. 45. P. 471–526.

4. *Фон Нейман Дж.* Теория самовоспроизводящихся автоматов: Пер. с англ. М.: Мир, 1971.

5. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «умном муравье». http://is.ifmo.ru/genalg/_ant_ga.pdf

6. *Наумов Л. А.* Метод введения обобщенных координат и инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов. Дис… канд. техн. наук: 05.13.12. СПб., 2007. 283 с.

7. *Скаков П. С.* Классификация поведения одномерных клеточных автоматов. http://is.ifmo.ru/papers/_skakov_master.pdf

РАЗРАБОТКА И ВЕРИФИКАЦИЯ МНОГОПОТОЧНЫХ АВТОМАТНЫХ ПРОГРАММ

М. А. Лукин

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Предлагаемый подход объединяет разработку и верификацию многопоточных автоматных [1, 2] программ в одном инструментальном средстве.

В автоматном программировании Автоматная программа состоит из *источников событий* для конечных автоматов, *системы конечных автомато*в и *объектов управления*. Источником событий может быть также и внешняя для программы среда.

Каждый поток управляется автоматом либо системой вложенных автоматов. Для обеспечения синхронизации потоков конечные автоматы отправляют друг другу события (то есть, одни конечные автомат становятся источниками событий для других конечных автоматов).

Верификация программ общего вида (в части построения модели) практически не может быть автоматизирована. Для автоматных программ эта задача решаема. Поэтому в настоящей работе также ставится задача разработки метода автоматической верификации автоматных программ. Наиболее известным верификатором является верификатор *SPIN* [3], который является открытым и бесплатным. Кроме того, *SPIN* был разработан для верификации многопоточных алгоритмов. При верификации [4, 5] на его основе требования к программе записываются на языке линейной темпоральной логики (*LTL*) [6]. Инверсии каждой *LTL*-формулы может быть сопоставлен автомат *Бюхи* [7]. Собственно верификация состоит в том, что верификатор с целью построения контрпримера (если он имеется) должен «пересечь» модель *Крипке [*4] и автомат *Бюхи*. Модель *Крипке* автоматически строится верификатором по модели программы, записанной на языке *Promela.*

Для визуальных автоматных программ, во-первых, построение модели на указанном языке по графам переходов может быть автоматизировано, а, во-вторых, переход от контрпримера на модели к контрпримеру в терминах автоматов также может быть автоматизирован.

Отличительной особенностью предлагаемого метода является возможность верифицировать параллельные программы.

Для поддержки метода было создано инструментальное средство Stater, которое позволяет разрабатывать автоматные программы, включая генерацию кода на разных языках программирования и верификацию. Для инструментального средства был разработан удобный интерфейс, подсвечивающий контрпример. Кроме того, Stater позволяет создавать автоматизированные классы [8], которые удобно встраиваются в уже существующие проекты.

Литература

1. *Шалыто А. А.* Технология автоматного программирования. http://is.ifmo.ru/works/tech_aut_prog/

2. *Туккель Н. И., Шалыто А. А.* Программирование с явным выделением состояний // Мир ПК. 2001. № 9. С. 132–138.

3. *SPIN home page*. http://SPINroot.com

4. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002.

5. *Лифшиц Ю.* Верификация программ и темпоральные логики. Лекция № 3 курса «Современные задачи теоретической информатики». http://download.yandex.ru/class/lifshits/lecture-note03.pdf

6. Linear temporal logic. http://en.wikipedia.org/wiki/Linear_temporal_logic

7. Büchi automaton. http://en.wikipedia.org/wiki/Büchi_automaton

8. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009.

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС ДЛЯ ИССЛЕДОВАНИЯ АВТОМАТНОГО УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ

С. А. Алексеев, В. О. Клебан

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В данной работе рассматривается комплекс, предназначенный для проектирования и исследования автоматных программ управления мобильными роботами.

Роботы — это физические агенты, которые выполняют поставленные перед ними задачи, проводя манипуляции в физическом мире. Для исследования задач управления роботами существуют специальные программные средства.

Однако при использовании существующих программных комплексов для исследования задач управления роботами присутствуют серьезные ограничения. В данной статье предлагается новый вариант такого комплекса.

В настоящей работе предложен программно-аппаратный комплекс для сквозного проектирования и отладки автоматных систем управления мобильными роботами. В состав комплекса входят: реальный робот; графический язык проектирования автоматных программ; модель робота в среде твердотельного моделирования; среда эмуляции и способ конвертирования модели робота в объект этой среды; контроллер робота с функцией отладки по шагам и функцией интерпретирования автоматной программы; протокол связи.

Для построения такого средства разработки, использована технология автоматного программирования [1].

Функционирование комплекса осуществляется за счет взаимодействия системы управления, контроллера робота и реального (виртуального) робота.

Объектом управления в данном комплексе является робот. Он построен на основе вычислительной платформы *Arduino*.

Для проектирования и отладки автоматных программ применена система управления на базе инструментального средства *Unimod*. Система управления позволяет проектировать конечные автоматы при помощи встроенных инструментов, а также запускать и отлаживать их. При разработке данного комплекса было принято решение использовать среду эмуляции *Webots [*2], предназначенную для моделирования, программирования и эмулирования мобильных роботов.

Виртуальная модель робота была спроектирована в системе твердотельного моделирования *Solidworks* [3]. Это позволяет экспортировать ее в сре-

ду *Webots*, а также получать чертежи для изготовления реальных роботов. На рисунке изображен виртуальный робот в среде эмуляции *Webots*, а также реальный прототип робота.

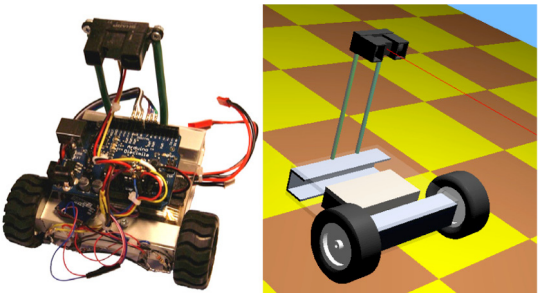


Рис. Реальная и виртуальная модели роботов

В качестве примера применения комплекса, рассматривается задача обеспечения робота максимально точными данными о наличии препятствий на маршруте [4].

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2011.

2. Cyberbotics Ltd. Webots reference manual. 2009. http://www.cyberbotics.com/cdrom/common/doc/webots/reference/reference.html

3. *Тику Ш.* Эффективная работа: SolidWorks 2004. СПб.: Питер, 2005.

4. *Алексеев С. А.* Программно-аппаратный комплекс для исследования автоматного управления мобильными роботами. СПбГУ ИТМО 2010. http://is.ifmo.ru/papers/_alekseev_bachelor.pdf

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ЛОКАЛЬНОЙ ОПТИМИЗАЦИИ ПРОГРАММНОГО КОДА

Е. В. Смирнов

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Скорость работы всегда оставалась одним из основных критериев оценки качества программного обеспечения. По этой причине стадия оптимизации в той или иной мере всегда присутствует в процессе разработки. И если затратная тонкая ручная оптимизация применяется только к тем программам, в которых критична каждая лишняя миллисекунда работы, то использование автоматических средств ускорения кода дешево и подходит всем, позволяя при этом добиться заметного повышения скорости.

Одним из этапов работы, который в той или иной мере включают в себя все современные оптимизирующие средства, является так называемая «локальная оптимизация». Эта оптимизация представляет собой поиск определенных небольших последовательностей низкоуровневых инструкций (на уровне ассемблера) и замена их на более эффективный аналог [1]. Локальный оптимизатор использует таблицу правил преобразования фрагментов кода, в которой содержатся неоптимальные наборы инструкций и то, чем их следует заменить. Обычно такая таблица составляется либо вручную (и может содержать лишь ограниченный набор достаточно простых правил), либо с помощью полного перебора наборов инструкций (что требует значительных затрат времени, как описано в [2, 3]).

Цель работы

Целью данной работы является проверка эффективности применения генетических алгоритмов как замены полного перебора при поиске лучшего (в данном случае в смысле «работающего быстрее») набора команд.

Ход работы

Была разработана реализация эволюционного процесса, оперирующего наборами инструкций и при этом поддерживающего их корректность (возможность запуска) и эквивалентность исходному набору. Были созданы реализации, позволяющие производить оптимизацию байткода виртуальной машины *Java* и инструкций архитектуры *x86*. На данный момент поддержаны только некоторые подмножества наборов команд целевых платформ, в основном различные арифметические действия и базовые операции с регистрами, памятью, стеком.

Полученное программное решение может быть в дальнейшем расширено для поддержания других целевых платформ. Оно представляет собой универсальное клиент-серверное приложение с поддержкой распределенных вычислений и сохранением всех накопленных успешных оптимизаций в базу данных. Такое решение может в дальнейшем быть использовано либо для накопления наборов правил для использования в обычных локальных оптимизаторах, либо для индивидуальной оптимизации конкретных приложений.

В качестве эксперимента были проведены поиски более быстрых вариантов для нескольких наборов инструкций, взятых из различных прикладных программ.

Пример

Из неоптимизированного исполняемого файла была взята последовательность из двух инструкций (№1):

```
xor eax, -1
and eax, ebx
```

С помощью широко используемых компиляторов языка *C++ MSVC* и *ICC,* при включенной максимальной оптимизации по скорости, был получен улучшенный вариант (№2, оба компилятора дали одинаковый результат):

```
not eax
and eax, ebx
```

Результатом работы генетического алгоритма стал вариант (№3):

```
xor eax, ebx
and eax, ebx
```

Варианты 2 и 3 превосходят исходный по скорости выполнения приблизительно в **полтора** раза, при этом вариант №3 обгоняет вариант №2 еще на **5–10 %**.

Результаты

Результат работы генетического процесса показал его высокую эффективность в деле оптимизации. Были найдены как стандартные решения (например, замена целочисленного умножения на два на сдвиг), так и нетривиальные оптимизации, для нахождения которых с помощью прочих методов потребовался бы глубокий анализ свойств вычисляемых выражений и детальное знание целевой архитектуры.

Как было показано на примере выше, генетические алгоритмы способны также улучшать результаты работы уже существующих оптимизирующих средств.

Генетические алгоритмы позволяют найти достаточно хорошее решение за намного меньшее число итераций, чем полный перебор. Например, для одной из рассмотренных последовательностей длиной в 11 инструкций, более выгодный вариант в большинстве случаев выращивается за 100–150 поколений, по 100 особей в каждом, тогда как всего возможно около 10^{20} наборов инструкций. Результат при этом оказывается лучше выдаваемого другими средствами на 3–15 %.

Литература

1. *McKeeman W. M.* Peephole Optimization // Communications of the ACM. № 8. Нью-Йорк, 1965. С. 443–444.

2. *Bansal S., Aiken A.* Automatic Generation of Peephole Superoptimizers // Proceedings of the 2006 ASPLOS Conference.

3. *Massalin H.* Superoptimizer — A Look at the Smallest Program // ACM SIGPLAN Notices. № 22. Нью-Йорк, 1987.

ВЫВОД NULLNESS-КОНТРАКТОВ ИЗ ИСХОДНОГО КОДА С ПОМОЩЬЮ ГРАФА ПОТОКА УПРАВЛЕНИЯ

А. В. Купцов

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Развитие разработки программного обеспечения всегда было направлено на уменьшение ее стоимости. Эффективным способом удешевления разработки является создание средств для разработчика, помогающих устранять программные ошибки на раннем этапе разработки, а, как известно, стоимость устранения ошибки тем меньше, чем раньше она будет обнаружена [1].

Одним из распространенных видов ошибок, которые тяжело обнаружить на стадии разработки, являются ошибки, связанные с обращением по нулевой ссылке [2]. Существуют специальные анализаторы, позволяющие выявлять такие ошибки [3]. В данной работе рассматривается анализатор для языка C#, основанный графе потока управления, позволяющий выявлять некоторые такие ошибки. Его алгоритм требует для лучшей своей работы контракты вида «может ли метод вернуть или принять в качестве аргумента значение null». Такие контракты называются nullness-контрактами. В данной работе предлагается метод вывода nullness-контрактов, предназначенных для рассмотренного анализатора.

Граф потока управления

Если представить операции, совершаемые компьютером при выполнении программы, как узлы, а возможные переходы между операциями — как ребра, их соединяющие, то получится граф потока управления [4]. Из-за различных языковых конструкций, используемых в разных языках, а также целей применения графа, его элементы могут иметь различные свойства. Дуги графа, рассматриваемого в данной работе, имеют описанные ниже свойства.

Каждая дуга графа помечена типом перехода (рис. 1). В языке C# переходы между операциями можно поделить на четыре типа:

- NEXT — обычный переход (на следующее утверждение, на следующий вызов метода или оператора, в текущем утверждении или выражении и т. д.);
- GOTO — переход по операторам goto, break, continue;
- RETURN — штатный выход из метода с передачей управления в вызывающую подпрограмму;
- THROW — «аварийный» выход из метода (с броском исключения).

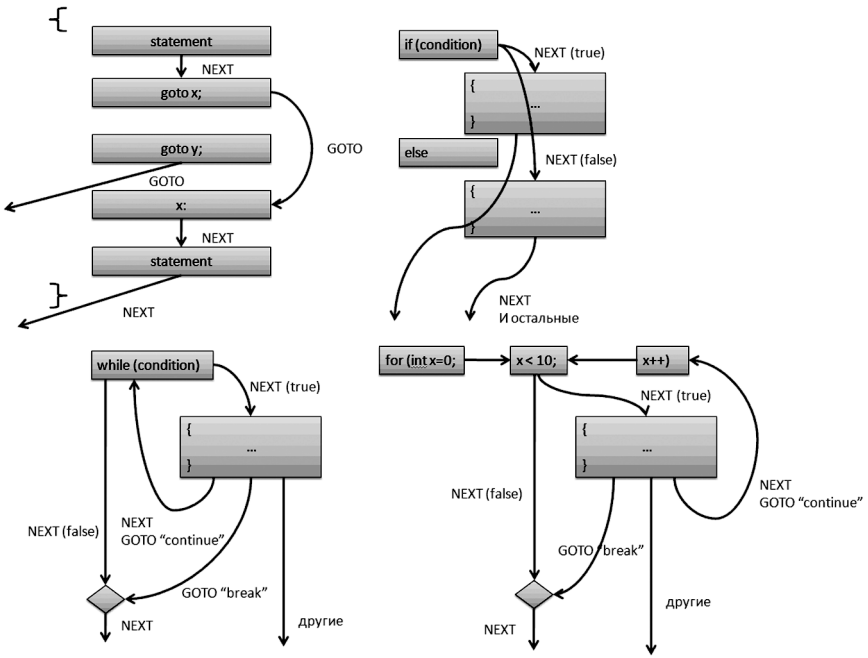


Рис. 1. Построение графа для последовательности операторов, операторов if, while, for

Некоторые NEXT-переходы помечаются еще одним свойством: при выходе из булевого выражения добавляется значение true или false. Оно назначается в зависимости от направления, в котором произойдет соответствующий переход, если это выражение будет вычислено в то или иное значение.

С помощью графа потока управления можно делать простые статические анализы кода. Например, сообщения компиляторов и интегрированных средств разработки о недостижимом коде строятся на основании недостижимых узлов графа.

Nullness-анализ

Для динамического анализа исходного кода с помощью графа потока управления с каждой его дугой ассоциируется некоторое состояние анализатора. При обходе графа в каждом узле графа объединяются состояния всех входящих дуг с непустым состоянием, и формируются состояния исходящих дуг. Этот процесс останавливается, когда состояния перестанут меняться.

Nullness-анализ — это эвристический динамический анализ кода, призванный выявить возможные места в программе, где происходит обращение по нулевой ссылке (в терминах C# — NullReferenceException). Также это анализ позволяет выявлять другие возможные ошибки в коде. Так как этот анализ эвристический, он по своей природе не может быть точным на 100 %. Поэтому различают два подхода: оптимистический — выявлять только те места в программе, где ошибка вероятность ошибки максимальна, и свести к минимуму ложные срабатывания, а также пессимистический — выявлять все места, где ошибка возможна. В данной работе рассматривается только оптимистический вид анализа.

В данном анализе подконтрольная переменная может находиться в следующих состояниях: UNKNOWN (состояние, введенное специально для оптимистического анализатора), TRUE, FALSE (оба — соответствующие состояния булевых переменных), NULL, NOT-NULL (оба — состояния ссылочных переменных, имеющих соответственно значение null или отличное от него). Вектор состояний — это набор состояний, в котором каждой подконтрольной переменной сопоставлено ровно одно состояние. Множество векторов состояний — неупорядоченный набор различных векторов состояний. Такое множество определяет возможные комбинации состояний переменных на ребре графа перехода. В зависимости от языка программирования каждому типу узла сопоставляется правило создания множества векторов состояний для выходного ребра. Такого рода правила заключаются в том, что множества векторов состояний, полученных на входе в узел, объединяются, а затем изменяются по определенному критерию. Например, при присваивании в переменную значения, полученного в результате выполнения оператора создания экземпляра класса, состояние этой переменной во всех векторах изменяется на NOT-NULL. А при сравнении значений двух ссылочных переменных в NEXT(true)-ребро попадут вектора состояний, имеющие одинаковые состояния сравниваемых переменных, а в NEXT(false)-ребро — остальные. Для предопределенных операторов не сложно составить набор правил изменения состояний на ребрах, но уже при использовании методов нам не обойтись без контрактов вида «этот метод может вернуть null», «этот метод никогда не возвращает null», «в данный параметр нельзя подставить null». Контракты такого рода называются nullness-контрактами (или nullness-аннотациями).

Обойдя граф потока управления с помощью этого алгоритма можно найти места в коде, где высока вероятность ошибки использования ссылки со значением null. Например, если при обращении к объекту по ссылке окажется, что один из векторов состояний переменных на ребре, входящим в узел, соответствующий этому обращению, содержит состояние NULL для данной ссылки, то существует такой путь в программе, при выполнении по которому значение в переменной будет null. А значит, разработчика можно известить о его возможной ошибке предупреждением вида «Возможно выбрасывание исключения NullReferenceException».

Вывод Nullness-контрактов

В данной работе вывод nullness-контрактов для краткости будет называться аннотированием. Анализатор, занимающийся аннотированием, называется аннотатором. Метод (или другая сущность программы, например, свойство, индексатор, аксессор, поле класса и т. д.) называется аннотированным, если он помечен nullness-контрактами.

Nullness-анализ может быть использован для аннотирования возможности возвращения значения null методом или иной сущностью программы. Если считать, что метод, в котором выводятся контракты, использует уже аннотированные сущности, то в результате анализа его графа потока управления мы можем сделать вывод о том, может ли он вернут null. Для этого рассматриваются все его RETURN-ребра. Если существует вектор на RETURN-ребре с состоянием возвращаемого значения NULL, то метод помечается контрактом [CanBeNull] (здесь и далее в квадратные скобки обозначают контракты, обозначение введено, чтобы не было путаницы с обозначением состояний на ребрах графа потока управления), если все RETURN-ребра содержат только состояния NOT-NULL возвращаемого значения, то метод помечается контрактом [NotNull]. Аналогично выводятся контракты для out-параметров и постусловий ref-параметров.

Чтобы аннотировать предусловие для параметра (например, для обычного параметра или ref-параметра) используется подход, при котором метод анализируется дважды: первый анализ проводился с начальными состояниями параметра UNKNOWN, второй — с состоянием NULL. Затем графы, полученные в результате этих двух анализов, сравниваются. Если во втором графе существуют выходные ребра THROW, отсутствующие в первом графе, то значение null, подставленное в качестве аргумента этого параметра приведет к исключению (в сравнении с ненулевым значением). А значит, необходима проверка значения перед передачей его в метод. Такие параметры помечаются контрактом [NotNull].

Если не учитывать полиморфизм, то аннотировать сущности в библиотеке можно итеративно следующим образом: на каждой итерации аннотируем все сущности программы, остановка алгоритма происходит, когда на очередной итерации не удается вывести новые контракты. Данный алгоритм сходится, потому что на все предусловия и постусловия могут только усиливаться при очередном проходе аннотатора, а количество возможных предусловий и постусловий ограничено.

При полиморфизме возникает вопрос, что делать с базовыми сущностями, если конкретные аннотируются по-разному. Чтобы контракты сущностей полиморфной иерархии были согласованы, необходимо соблюдать следующее:

- Предусловия конкретных параметров всегда не сильнее предусловий базовых.
- Постусловия конкретных сущностей всегда не слабее постусловий базовых.

Сила условий определяется в следующем порядке: отсутствие контракта (Unknown), [CanBeNull], [NotNull]. Ввиду того, что постусловия сущностей не могут быть сильнее постусловий конкретных, а так же того, что аннотатор не может ослаблять условия, важен порядок аннотирования сущностей. Если использовать порядок «сначала конкретные сущности, затем базовые», появляются похожие проблемы в выводе несогласованных по иерархии предусловий. Поэтому алгоритм был усовершенствован следующим образом. Каждая итерация аннотатора делится на две части: в первой аннотатор обходит сущности в порядке «сначала конкретные, затем базовые», аннотируя только постусловия, а во второй — аннотируя только предусловия в порядке «сначала базовые, затем конкретные сущности».

Использование согласования сущностей в полиморфных иерархиях позволяет проаннотировать на 37 % больше сущностей, чем аннотатор, который не учитывает несогласованные сущности, а просто не выводит их в результат. Сравнение проводилось на стандартных библиотеках .NET Framework.

Литература

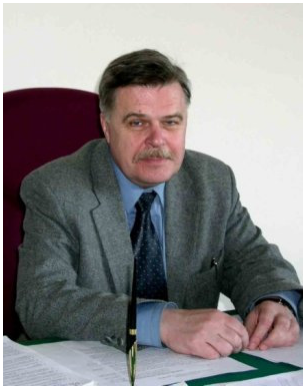
1. *Билл Грэхем (Bill Graham), Пол Леру (Paul N. Leroux), Тодд Лендри (Todd Landry).* Использование статического и динамического анализа для повышения качества продукции и эффективности разработки. QNX Software Systems, Klocwork. http://www.swd.ru/index.php3?pid=828

2. *Knute Axelson, Mary Bellino, Dave Harper, Dave Iffland.* Coding: The Handbook for Information Technology. Blue Line Press Inc., 2005.

3. *Arnout F. M. Engelen.* Nullness Analysis of Java Source Code. Al-Imam Muhammad ibn Saud Islamic University, Master's thesis. 2006.

4. *Thomas William Reps, Mooly Sagiv, Jorg Bauer.* Program analysis and compilation, theory and practice. Springer–Verlag, 2007.

Синтез элементов компьютерной архитектуры



Леонов
Геннадий Алексеевич

председатель оргкомитета конференции

д.ф.-м.н., профессор, чл.-корр. РАН
декан математико-механического факультета СПбГУ
заведующий кафедрой прикладной кибернетики СПбГУ

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМА АНАЛИЗА ТИПОВ ГРАНИЦ УСТОЙЧИВОСТИ

М. А. Киселева

Санкт-Петербургский государственный университет

Важной проблемой прикладной механики является исследование поведения динамических систем вблизи границ области устойчивости.

В 1949 году Баутиным [1] был предложен метод, позволяющий различить безопасные и опасные границы области устойчивости. В первом случае достаточно малое смещение (изменение параметров системы) приводит к незначительному изменению поведения системы, тогда как во втором случае малейшее изменение опасной границы влечет неконтролируемое нарастание отклонений от начального состояния системы. В последствии, изменения поведения систем вблизи безопасной и опасной границ устойчивости Андроновым [2] были классифицированы как «мягкое» и «жесткое» возбуждение колебаний соответственно.

Для анализа поведения системы вблизи границы области устойчивости в классических работах Пуанкаре [3] и Ляпунова [4] был разработан метод вычисления ляпуновских величин (или констант Пуанкаре-Ляпунова), которые определяют поведение системы в окрестности границы. Вопрос о принадлежности границы к опасной сводится к определению знака первой ляпуновской величины, отличной от нуля, в точке, лежащей на границе. Существуют различные методы определения ляпуновских величин.

В настоящем сообщении представлен алгоритм анализа типов границ устойчивости двумерных квадратичных систем, основанный на использовании предложенного Леоновым [5–8] прямого метода вычисления ляпуновских величин во временной области и евклидовой системе координат. Данный метод позволяет строить приближения решений системы и времени “оборота” траектории в виде конечного ряда по степеням начального данного. Использование сведения квадратичных систем к системе Льенара с помощью интегрального преобразования специального вида позволило разработать высокоэффективные библиотеки в пакете MatLab, реализующие вышеописанный метод для последовательного вычисления символьных выражений ляпуновских величин.

Литература

1. *Баутин Н. Н.* Поведение динамических систем вблизи границ области устойчивости. М.: Гостехиздат, 1949.

2. *Andronov A. A., Vitt A. A., Khaikin S. E.* Theory of oscillations. Oxford, New York: Pegamon Press, 1966.

3. *Poincare H.* Memoire sur les courbes definies par les equations differentielles // J. de Mathematiques Pures et Appliquees, (4). Vol. 1. Pp. 167–244. 1885.

4. *Lyapunov A. M.* Stability of Motion. New York and London: Academic Press, 1966 (1892).

5. *Kuznetsov N. V., Leonov G. A.* Computation of the first Lyapunov quantity for the second-order dynamical system. Third IFAC Workshop Periodic Control System, 2007.

6. *Kuznetsov N. V., Leonov G. A.* Computation of Lyapunov quantities. 6th EUROMECH Nonlinear Dynamics Conference (http://lib.physcon.ru/?item=1802), 2008.

7. *Kuznetsov N. V., Leonov G. A.* Lyapunov quantities, limit cycles and strange behaviour of trajectories in two-dimensional quadratic systems // Journal of Vibroengineering. Vol. 10. Iss. 4. Pp. 460–467. 2008.

8. *Леонов Г. А., Кузнецов Н. В., Кудряшова Е. В.* Прямой метод вычисления ляпуновских величин двумерных динамических систем // Тр. ИММ УрО РАН, 16. № 1. С. 119–126. 2010.

ВЫЧИСЛЕНИЕ ХАРАКТЕРИСТИКИ ФАЗОВОГО ДЕТЕКТОРА-ПЕРЕМНОЖИТЕЛЯ ДЛЯ ДВУХ ИМПУЛЬСНЫХ СИГНАЛОВ

М. В. Юлдашев

Санкт-Петербургский государственный университет

В настоящее время существует большое количество методов решения задачи синхронизации сигналов различного типа в радиотехнике [5, 7] и компьютерной архитектуре [1–4, 6]. Однако, проблема построения точных математических моделей, адекватно описывающих системы фазовой автоподстройки частоты (ФАП) далека от полного решения и требует разработки специальных подходов, связанных с интегральными, дифференциальными и интегро-дифференциальными уравнениями.Разработано несколько типов систем ФАП(классические системы LPLL, цифровые DPLL, полностью цифровые ADPLL, программные SPLL), работающие с различными типами сигналов(синусоидальные, импульсные и другие). Отметим, что различные типы ФАП имеют особенности реализации основных элементов системы, что отражается на параметрах работы системы.

Типичная система ФАП содержит эталонный генератор (ЭГ), подстраиваемый генератор (ПГ), фазовый детектор (ФД) и низкочастотный фильтр. Для построения строгой математической модели, требуется определение характеристика фазового детектора. В настоящем сообщении рассмотрен стандартный ФД, реализованный в виде перемножителя сигналов [5, 7]. Используя изложенные в [3] подходы, разработан метод вычисления характеристики ФД для высокочастотных сигналов импульсного типа с переменной продолжительностью импульса.

Результаты, полученные описанными методами, позволяют проводить дальнейший анализ ФАП и находить основные параметры подобных систем, такие как диапазон захвата, которые невозможно определить методом локального анализа.

Литература

1. *Abramovitch D.* Phase-locked loops: A control centric tutorial / D. Abramovitch // Proceedings of the American Control Conference. Vol. 1. 2002. Pp. 1–15.

2. DSP Processor Fundamentals: Arhetecture and Features / P. Lapsley, J. Bier, A. Shoham, E. A. Lee. New York: IEE Press, 1997.

3. *Leonov G.* Automation control — Theory and Practice / G. Leonov, N. Kuznetsov, S. Seledzhi. In-Tech, 2009. Pp. 89–114.

4. *Leonov G.* Stability and bifurcations of phase-locked loops for digital signal processors / G. Leonov, S. Seledghi // International journal of bifurcation and chaos. 2005. Vol. 15. No. 4. Pp. 1347–1360.

5. *Lindsey W.* Synchronization systems in communication and control / W. Lindsey. New Jersey: Prentice-Hall, 1972.

6. *Smith S. W.* The Scientist and Engineer's Guide to Digital Signal Processing / S. W. Smith. San Diego, California: California Technical Publishing, 1999.

7. *Viterbi A.* Principles of coherent communications / A. Viterbi. New York:McGraw-Hill, 1966.

ВЫЧИСЛЕНИЕ ХАРАКТЕРИСТИКИ ФАЗОВОГО ДЕТЕКТОРА-ПЕРЕМНОЖИТЕЛЯ ДЛЯ СИНУСОИДАЛЬНОГО И ИМПУЛЬСНОГО СИГНАЛОВ

Р. В. Юлдашев

Санкт-Петербургский Государст,венный Университет

Системы фазовой автоподстройки частоты (ФАП, phase locked loops — PLL) широко распространены в радиотехнике [2, 6, 7] и компьютерной архитектуре [2, 4].

В настоящее время активно используются различные программные и электронные реализации ФАП. Достоинством программных протоколов автоподстройки частоты является относительная простота их создания, которая, однако, ведет к ограничениям на максимально возможные частоты работы, связанным с внутренней реализаций самого протокола.

Недостатком же электронной реализации — является необходимость проведения сложного нелинейного анализа моделей ФАП. Данная работа посвящена одному из аспектов проведения такого анализа.

Для построения адекватной математической нелинейной модели ФАП необходимо определять [1] характеристику фазового детектора (ФД) — нелинейного элемента, на входе которого сигналы эталонного и подстраиваемого генераторов, а выход содержит низкочастотный корректирующий сигнал.

В настоящем сообщении рассмотрен стандартный ФД, реализованный в виде перемножителя сигналов [5, 7]. Используя изложенные в [3, 4] подходы, основанные на качественной теории динамических систем, проведен нелинейный анализ систем синхронизации. Разработан метод вычисления характеристик фазового детектора для синусоидального и импульсного сигналов. Полученные результаты позволяют использовать методы нелинейного анализа [3, 4, 7] для более широкого класса сигналов.

Литература

1. *Abramovitch D.* Phase-locked loops: A control centric tutorial / D. Abramovitch // Proceedings of the American Control Conference. Vol. 1. 2002. Pp. 1–15.

2. DSP Processor Fundamentals: Arhetecture and Features / P. Lapsley, J. Bier, A. Shoham, E. A. Lee. New York: IEE Press, 1997.

3. *Leonov G.* Automation control — Theory and Practice / G. Leonov, N. Kuznetsov, S. Seledzhi. In-Tech, 2009. Pp. 89–114.

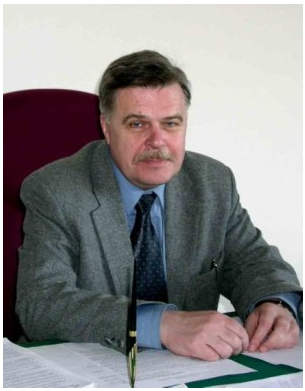
4. *Leonov G.* Stability and bifurcations of phase-locked loops for digital signal processors / G. Leonov, S. Seledghi // International journal of bifurcation and chaos. 2005. Vol. 15. No. 4. Pp. 1347–1360.

5. *Lindsey W.* Synchronization systems in communication and control / W. Lindsey. New Jersey: Prentice-Hall, 1972.

6. *Smith S. W.* The Scientist and Engineer's Guide to Digital Signal Processing / S. W. Smith. San Diego, California: California Technical Publishing, 1999.

7. *Viterbi A.* Principles of coherent communications / A. Viterbi. New York:McGraw-Hill, 1966.

Организация производственной практики студентов факультета в ИТ-компаниях



Леонов
Геннадий Алексеевич

председатель оргкомитета конференции

д.ф.-м.н., профессор, чл.-корр. РАН
декан математико-механического факультета СПбГУ
заведующий кафедрой прикладной кибернетики СПбГУ

СОТРУДНИЧЕСТВО МАТЕМАТИКО-МЕХАНИЧЕСКОГО ФАКУЛЬТЕТА И КОМПАНИИ EXIGEN SERVICES В ОБЛАСТИ ПОДГОТОВКИ ИТ-СПЕЦИАЛИСТОВ

А. А. Волков, В. И. Кияев, Н. В. Кузнецов, Г. А. Леонов, В. В. Оносовский, С. М. Селеджи

Санкт-Петербургский Государственный Университет

Взаимодействие университета с компанией Exigen Services началось в 2006 г. и явилось результатом совместной инициативы президента Exigen Services Алека Милославского и декана математико-механического факультета Г. А. Леонова. Главной стратегической целью этого взаимодействия является подготовка и адаптация студентов ИТ-специальностей к последующей профессиональной деятельности в крупных международных ИТ-компаниях, работающих в области коммерческого бизнес-программирования — наиболее обширной и востребованной в настоящее время части программной инженерии.

Взаимодействие с компанией Exigen Services строится на некоммерческой основе — вкладом со стороны компании является экспертиза в области самых современных технологий и методологий разработки ПО, используемых в индустрии, и знание современной корпоративной культуры, передаваемое студентам в рамках участия специалистов компании в учебном процессе; вкладом факультета является предоставление компании доступа «изнутри» к процессу обучения студентов, позволяющий осуществлять целенаправленный отбор отдельных студентов для последующей индивидуальной работы с ними. Как показывает опыт, такой подход позволяет, оставаясь в рамках университетского образовательного процесса, приблизиться к решению главной проблемы — соответствия профиля подготовки ИТ-специалистов реальным (и быстро меняющимся) потребностям индустрии. Кроме того, поскольку курсы лекций и практики Exigen Services начинаются, как правило, на третьем курсе (до этого студенты еще не имеют достаточной базовой подготовки для восприятия материала), реальные перспективы работы или стажировки в ИТ-компании (не обязательно Exigen Services) у них появляются к четвертому курсу, когда свободного времени (и, следовательно, возможностей безболезненно совмещать работу с учебой) становится намного больше. Ценность прошедших такую подготовку студентов для потенциальных работодателей оказывается намного выше, чем у «обычных» студентов, пытающихся в первый раз устроиться на работу в ИТ-компании.

При этом как для университета, так и для компании Exigen не столь существенно, в какой именно компании будет работать после окончания университета тот или иной студент, подготовленный в рамках программы — гораздо

важнее сам процесс повышения общей технологической и производственной культуры в среде молодых ИТ-профессионалов. Таким образом, непосредственный рекрутинг студентов в компанию Exigen Services не является приоритетом — принципиальным моментом является тесное взаимодействие компании с университетом в рамках учебного процесса, при котором возможен индивидуальный отбор сравнительно небольшого числа «духовно близких» студентов, персонально заинтересованных в дальнейшей карьере именно в компании Exigen.

Участие специалистов Exigen Services в учебном процессе осуществляется в следующих формах:

- чтение специалистами Exigen Services учебных курсов для студентов по промышленному программированию и тестированию ПО;
- организация практик для студентов в компании Exigen Services;
- предоставление заинтересованным студентам возможности прохождения углублённой подготовки в рамках внутрифирменного образовательного центра Exigen — ИТ-колледжа;
- проведение специалистами Exigen интенсивных тренингов и мастер-классов для заинтересованных студентов и преподавателей факультета, посвящённых углублённому изучению современных промышленных технологий и средств программирования;
- консультационная помощь преподавателям Университета в виде «семинаров для преподавателей», позволяющих им лучше ориентироваться в современных передовых технологиях и программных продуктах.

Основной **лекционный курс**, который читается специалистами Exigen Services в рамках программы сотрудничества — это годовой курс «Введение в бизнес-программирование на Java/J2EE» для студентов 3 курса кафедры теоретической кибернетики. Курс носит междисциплинарный, синтетический характер. Его главная задача — не только научить студентов использованию разнообразных технологий, входящих в J2EE, но и дать им живое ощущение тех реалий, с которыми они столкнутся, придя на работу в промышленные IT-компании, занимающиеся бизнес-программированием. Поэтому, наряду с изучением конкретных технологий, курс включает в себя элементы менеджмента проектов, организации командной работы, деловои коммуникации и ряда других дисциплин, относящихся к так называемым «soft skills», которые традиционно слабо представлены в университетском обучении, но в то же время жизненно необходимы молодым специалистам при работе над промышленными проектами разработки ПО для бизнеса.

Параллельно с чтением лекций, в рамках курса проводятся практические занятия и лабораторные работы, цель которых — дать студентам адекватное представление о практических аспектах применения изучаемых технологий на «приближенных к реальности» проектах.

Студенческие практики — это групповые практические занятия студентов в условиях, приближенных к условиям реальных проектов. Практики проводятся по двум направлениям — программирование на Java/J2EE и программирование на C#/.NET.

Практики могут быть как частью учебных курсов, читаемых специалистами Exigen Services для студентов факультета, так и проводиться независимо. Как правило, группа в процессе практики выполняет один или несколько учебных проектов, представляющих собой облегчённые версии фрагментов реальных бизнес-приложений. Студенты для участия в практиках отбираются ВУЗами — партнёрами нашей компании, и мы не производим никакого дополнительного тестирования или отбора. В отдельных случаях на практики могут направляться люди, поступавшие в Школу, но по результатам отбора не попавшие в неё.

Практики могут проводиться в интенсивном режиме — две или три недели подряд, ежедневно, по 4–5 часов в день, либо в еженедельном режиме — один день в неделю, 4–5 часов в день, на протяжении одного или двух семестров. Занятия со студентами обычно проводятся по утрам, чтобы снизить нагрузку на компьютерные классы.

Основные требования к студентам, приходящим на практику — знание основ выбранного языка программирования (Java или C#), основ ООП, алгоритмов и структур данных. Программа каждой конкретной практики выстраивается в зависимости от общего уровня подготовки группы — варьируется степень сложности учебного проекта, а, кроме того, иногда в начале практики специалистам компании приходится проводить «ликбез» для повышения общего уровня программирования студентов.

По завершении практики отдельные студенты-участники могут получить приглашения на прохождение стажировки или приём на работу в компанию. Как правило, доля студентов, получающих такие приглашения, невелика, но, в то же время, практики позволяют отбирать людей, персонально мотивированных и заинтересованных в дальнейшей работе именно в компании Exigen Services, с которыми ведётся дальнейшая индивидуальная работа.

Деятельность внутрифирменного центра подготовки молодых специалистов Exigen Services — **ИТ-колледжа** — включает в себя обучение по трем дисциплинам:

- Программирование на Java/J2EE.
- Программирование на C#/.NET.
- Тестирование программного обеспечения.

Занятия в ИТ-колледже совмещают в себе теорию и практику. Основное внимание уделяется углублению и развитию уже имеющихся базовых знаний, а не освоению новых технологий; процесс обучения, таким образом, направлен не «вширь», а «вглубь»; методика обучения нацелена, прежде всего, на формирование практических навыков и приобретение опыта работы

в условиях, максимально приближенных к реальности. Такой подход ближе к подходу бизнес-школ, чем к традиционному университетскому обучению. Курсы в ИТ-колледже, в зависимости от конкретной дисциплины, занимают 4–6 недель, занятия проводятся по вечерам (обычно с 16 до 20 часов), 2–4 дня в неделю.

Поступление на курсы в ИТ-колледж проходит на конкурсной основе, по результатам отбора и тестирования кандидатов. Отбор достаточно жёсткий, как правило, конкурс составляет не менее 5 человек на место. Основные требования к кандидатам на направления, связанные с программированием — это знание выбранного языка программирования (Java или C#), знание основ ООП, алгоритмов и структур данных, очень желательно наличие опыта самостоятельного прогораммирования и хорошее знание английского языка. При отборе кандидатов на обучение тестированию, проверяется общий уровень технической подготовки на уровне пользователя ПК, уровень знания английского языка, но основное внимание уделяется персональным качествам кандидата — чёткости мышления, аналитическим способностям, аккуратности и методичности. Существенным условием при прохождении отбора в ИТ-колледж является потенциальная возможность кандидата уделять достаточно времени работе в компании после окончания курсов (в случае получения приглашения на работу в Exigen Services) — 32–40 часов в неделю, в зависимости от дисциплины.

Обучение в ИТ-колледже не накладывает на учащихся никаких специальных обязательств по отношению к компании Exigen Services в отношении дальнейшей работы. По окончании курсов многие выпускники получают приглашения от Exigen Services на стажировку или работу в компании.

Проводимые специалистами Exigen Services тренинги и мастер-классы для студентов и сотрудников университета представляют собой, как правило, адаптированные для студенческой аудитории варианты используемых в компании внутрифирменных тренингов. Тематика тренингов охватывает два основных направления: с одной стороны, это тренинги и мастер-классы, посвящённые современным технологиям, методологиям и инструментальным средствам разработки и тестирования ПО, которые активно используются в индустрии в настоящее время. Другое направление проводимых тренингов — это развитие надпрофессиональных навыков (soft skills), таких, как эффективная деловая коммуникация, навыки работы в команде, развитие креативности и нестандартного мышления и т. д. Как показывает опыт нашей работы, именно такие тренинги оказываются наиболее востребованы студентами.

ОРГАНИЗАЦИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ СТУДЕНТОВ МАТЕМАТИКО-МЕХАНИЧЕСКОГО ФАКУЛЬТЕТА В КОМПАНИИ МОТОРОЛА

А. А. Волков, В. И. Кияев, Н. В. Кузнецов, Г. А. Леонов, В. В. Оносовский, С. М. Селеджи

Санкт-Петербургский Государственный Университет

Российское отделение Моторолы (ЗАО «Моторола ЗАО») существует в Санкт-Петербурге с 1997 года — и за это время сотрудниками компании стало более ста сорока выпускников математико-механического факультета СПбГУ. Компания заинтересована в эффективном сотрудничестве с университетами, её Академическая программа нацелена в первую очередь на подготовку перспективных и квалифицированных кадров для компании. По инициативе высшего менеджмента компании при деятельном участии главного конструктора программного обеспечения С. Н. Баранова компания уже несколько лет активно и успешно сотрудничает с университетами Санкт-Петербурга. На рисунке 1 показан базовый принцип этого взаимодействия.

Рис. 1.

Практическое обучение студентов проводится непосредственно в офисах компании — Моторола привлекает студентов и аспирантов непосредственно в реальные проекты, выполняемые в подразделениях компании. В процессе работы студенты проходят проектное обучение по принципу «от простого

к сложному», постепенно набирая необходимые знания и опыт проектной работы, усваивая принципы корпоративной культуры. В конце каждого семестра руководители студенческих групп устраивают публичные отчеты в виде презентаций результатов выполненных работ по проектам.

Для практических занятий в расписании для студентов 3-го курса кафедры прикладной кибернетики специально выделен полный день, в течение которого они обязаны находиться в подразделениях компании, выполняя порученную работу по проектам. В 2007–2010 учебных годах запущено 17 проектов с участием студентов СПбГУ. На рисунке 2 показана схема организации проектного взаимодействия компании с заинтересованным факультетом любого университета.

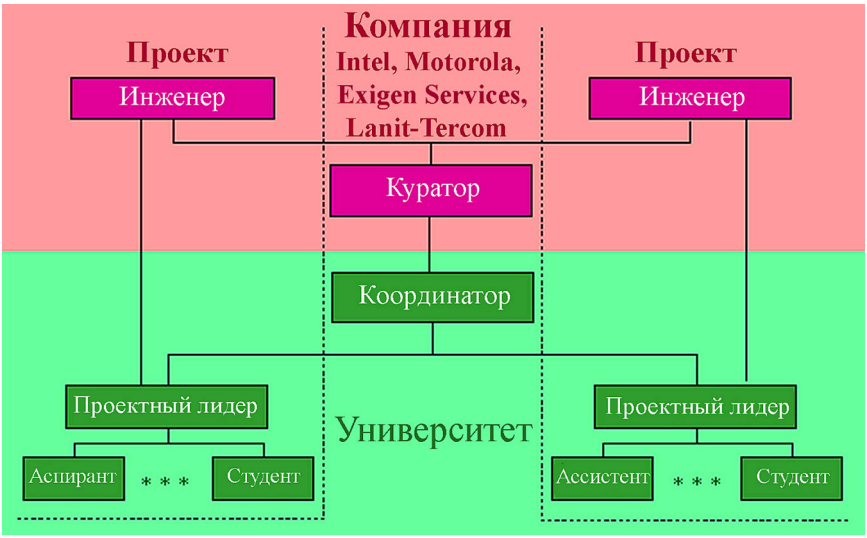


Рис. 2.

Требования, предъявляемые к участникам, довольно высоки: использование английского языка на базовом уровне, знание реляционных баз данных (SQL), алгоритмических языков программирования высокого уровня C, C++, C#, Java, скриптовых языков PHP, Python, Perl, Shell, некоторые знания по Hardware, интерес к работе, усидчивость, трудолюбие, ответственность, коммуникабельность, желание работать в команде.

Отметим при этом, что льготы, получаемые студентами в процессе прохождения стажировки, весьма существенны: студентам выплачивается дифференцированная (в зависимости от достижений) добавка к стипендии, они имеют возможность заниматься на курсах английского языка (английский язык является официальным языком компании, документация и переписка

ведутся на английском языке). Самые «продвинутые» студенты работают в летнее время на полную рабочую неделю, получая соответствующую заработную плату. После окончания 5-го курса или аспирантуры выпускник, хорошо зарекомендовавший себя, имеет преимущественное право на зачисление в компанию в качестве постоянного сотрудника. Такая модель взаимодействия факультета с компаниями лежит в русле общей системы образовательных программ, принятых в компании для взаимодействия с вузами, в ее создании и разработке существенное участие принимали преподаватели математико-механический факультет факультета.

19 ноября 2010 года в Кракове состоялась Международная конференция получателей грантов Моторола-фонда. Участники конференции в докладах и во время проведения круглых столов заинтересованно обсуждали многочисленные вопросы использования проектов, поддерживаемых Моторолой для решения социальных и образовательных задач. Доклад В. В. Оносовского об адаптации студентов и аспирантов для работы в IT-компаниях вызвал живой интерес и большое количество вопросов о формах взаимодействия, содержании проектов, отчетности, поощрении студентов и т. д. Всё это показало перспективность используемой модели сотрудничества.

СОТРУДНИЧЕСТВО МАТЕМАТИКО-МЕХАНИЧЕСКОГО ФАКУЛЬТЕТА И КОМПАНИИ ИНТЕЛ В ОБЛАСТИ ПОДГОТОВКИ ИТ-СПЕЦИАЛИСТОВ

А. А. Волков, В. И. Кияев, Н. В. Кузнецов, Г. А. Леонов, В. В. Оносовский, С. М. Селеджи

Санкт-Петербургский Государственный Университет

Взаимодействие с корпорацией Intel — нацелена на долгосрочное сотрудничество и получение реальных взаимных выгод. Академическая программа Intel по взаимодействию с вузами в России и в других странах СНГ нацелена на создание в России учебно-образовательных лабораторий на базе ведущих университетов. Об этих лабораториях писал вице-президент корпорации Intel Эндрю Гроув в предисловии к своей замечательной книге «Выживают только параноики». Цель такой лаборатории — всемерное повышение уровня подготовки программистов и ИТ-специалистов, а также максимальное использование технологий и программных продуктов Intel в учебном и исследовательских процессах.

В 2003 году такая межфакультетская Учебно-исследовательская лаборатория системного программирования и информационных технологий (SPRINT Lab) была открыта на базе математико-механического факультета СПбГУ при непосредственном участии CEO корпорации Intel Пола Отеллини. Лаборатория создана при организационной и финансовой поддержке корпорации Интел в России на правах учебно-исследовательской лаборатории. Создание такой лаборатории соответствует взаимным долгосрочным интересам СПбГУ и корпорации Intel в России. Сотрудничество Университета с Intel способствует использованию в научной работе и учебном процессе Университета последних достижений в области современных компьютерных систем и информационных технологий. Для Intel поддержка деятельности Лаборатории позволит содействовать выполнению перспективных научных исследований и проектов в указанной области — в том числе и в интересах Интел: формировать целевые образовательные программы подготовки специалистов, иметь возможность проводить профессионально-ориентационную работу с выпускниками СПбГУ.

Миссия Лаборатории содержит следующие базовые составляющие:

- обновление и совершенствование образовательной деятельности в СПбГУ за счёт формирования целевых образовательных программ;
- решение актуальных научно-исследовательских задач в области современных компьютерных систем и информационных технологий;
- комплексная подготовка в СПбГУ высококвалифицированных специалистов в этой области.

Лаборатория организует обучение в форме стажировки, набирая для этого на конкурсной основе от 20 до 25 студентов старших курсов, аспирантов и молодых учёных (не старше 27 лет). Стажеры факультативно слушают общие и специальные курсы, выполняют практические задания, под руководством преподавателей факультетов и менторов от Intel участвуют в образовательных и учебно-исследовательских проектах по направлениям, находящихся в русле исследований Intel, участвуют в работе постоянно действующих семинаров. Одной из форм образовательной деятельности Лаборатории является организация приглашенных лекций по различным разделам компьютерных знаний. Лекции на актуальные темы читают сотрудники Intel, профессоры и специалисты российских и зарубежных университетов, сотрудники ведущих компаний по созданию программных продуктов.

Для закрепления знаний Intel организует на конкурсной основе на базе Нижегородского и Санкт-Петербургского университетов зимние и летние школы, лекции в которых читают известные российские и зарубежные специалисты в области системного программирования и информационных технологий. Занятия зимней школы продолжаются две недели, летних — около двух месяцев.

Управление информацией

Нестеров
Вячеслав Михайлович

д.ф.-м.н.
генеральный директор
Центра разработок компании EMC в Санкт-Петербурге

Суворов
Владимир Александрович

архитектор отраслевых решений
Центра разработок компании EMC в Санкт-Петербурге

УЧЕТ СЛОЖНОСТИ ВАРИАНТА ЗАДАНИЯ ПРИ ОЦЕНКЕ РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ

С. И. Гиндин

Научный руководитель:
Гуров В. С., кандидат технических наук,
доцент кафедры компьютерных технологий

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Краткое вступление, постановка проблемы

На сегодняшний день одной из наиболее технологичных и объективных форм контроля знаний является тестирование, повсеместно используемое во многих странах мира, в том числе, и в России, в области подготовки и сертификации специалистов, мониторинга и оценки качества образования. Развитие педагогики позволило существенно модернизировать тестовые технологии контроля знаний и поднять их на качественно иной уровень. Использование информационных технологий позволило автоматизировать обработку результатов и сделало возможным массовое тестирование, а также привело к созданию компьютерных систем тестирования знаний.

В настоящее время существуют два теоретических подхода к оценке тестов: классическая теория и более предпочтительная современная теория Item Response Theory (IRT), название которой в русскоязычной литературе переводится по-разному: «теория моделирования и параметризации педагогических тестов» (ТМПТ)[1], «математико-статистическая теория оценки латентных параметров заданий теста и уровня подготовленности испытуемых»[2] и прочее. Преимуществами ТМПТ перед классической теорией являются: независимость характеристик заданий от группы испытуемых, при помощи которой они были получены; независимость оценки выраженности тех или иных диагностических параметров у испытуемого от используемой методики; возможность, благодаря итерационной процедуре нахождения оценок параметров, раздельного учета степени значимости ответа на каждый вопрос методики для получения конечного балла[3].

---

[1] *Нейман Ю. М., Хлебников В. А.* Введение в теорию моделирования и параметризации педагогических тестов. М.: Прометей.

[2] *Аванесов В. С.* Применение тестовых форм в Rasch Measurement // Педагогические измерения. 2005. № 4.

[3] *Schumacker, Randall E.* ITEM RESPONSE THEORY. http://www.appliedmeasurementassociates.com/White%20Papers/ITEM%20RESPONSE%20THEORY.pdf

В ТМПТ устанавливается связь между двумя множествами значений латентных параметров, определяющих уровень подготовленности испытуемых θ_i, $i = 1 \ldots N$, где i — номер испытуемого, и трудность j-го задания b_j, $j = 1 \ldots K$. В наиболее общем варианте ТМПТ для дихотомного тестирования, модели Three Parameter Logistic (3PL), вероятность i-того испытуемого верно ответить на j-тый вопрос определяется формулой:

$$P_i(X_j = 1 \mid \theta_i) = c_j + \frac{1 - c_j}{1 + e^{-Da_j(\theta_i - b_j)}},$$

в которой, помимо параметров θ_i, b_j также вводятся дополнительные параметры, характеризующие задание: a_j — показатель уровня различающей способности задания дифференцировать испытуемых по уровню подготовленности, c_j — показатель потенциальной вероятности угадать ответ в случае, если все испытуемые не знают правильного ответа, и постоянный масштабный множитель D, позволяющий перейти от логистической функции к интегральной функции нормированного нормального распределения[4]

$$P_i(X_j = 1 \mid \theta_i) \sim \int_{-\infty}^{\theta_i - \beta_j} e^{t^2/2}\, dt.$$

Основные принципы IRT обуславливают необходимость привлечения довольно сложного математико-статистического аппарата, разработки специальных программных продуктов и использование компьютерной техники, и как следствие накладывает требования на количество испытуемых в тесте не менее 1000 человек для получения приемлемо точных оценок параметров[5].

Цель работы

Необходимо разработать методику учета сложности варианта задания при оценке результатов тестирования для объективного определения уровня подготовки тестируемых, подходящую для небольшого количества испытуемых, менее 1000 человек.

Базовые положения исследования

Предлагаемое решение — предложить эффективный метод повышения точности оценки параметров модели 3PL для небольшого количества испытуемых, позволяющий затем использовать 3PL для для анализа результатов тестирования.

---

[4] *Михеев О. В.* Математические модели педагогических измерений // Педагогические измерения. 2004. № 2.

[5] *Steven M.* Downing, Thomas M. Haladyna. Handbook of test development // Routledge, 2006. http://books.google.ru/books?id=CwxFhehJllwC

Метод повышения точности — состоит в дополнении исходных данных вспомогательными наборами, полученными из исходных с помощью копирования, последующего удаления некоторых значений набора (верный / неверный ответ) в случайных позициях и их замены по специальному правилу и на основании сохранившихся данных в других имеющихся наборах.

Оценка эффективности метода — производится на основе сравнения величин основных статистических погрешностей измерения, рассчитанных для значений параметров, смоделированных на основании «дополненного» множества ответов и заранее известных параметров исходного множества ответов. Измеряемые величины: среднеквадратическая ошибка

$$СКВО = \sqrt{\frac{\sum_{k=1}^{r}(\alpha_{i,k} - \alpha_{i,k})^2}{r}},$$

среднее отклонение

$$СОТКЛ = \frac{\sum_{k=1}^{r}(\alpha_{i,k} - \alpha_{i,k})}{r}.$$

Промежуточные результаты

Проведен информационно-аналитический обзор существующих подходов к дополнению неполных данных результатов тестирования, выделены их ограничения.

Проведено сравнение нескольких методов дополнения (метод случайного заполнения, метод максимального правдоподобия, регрессионный метод), а также количественных показателей процента увеличения исходного множества ответов и удаления элементов в дополняющих наборах. Оценки получены с помощью программных пакетов ltm, irtProb, irtoys при использовании среды программирования R[6].

Получено улучшение точности оцениваемых параметров результатов тестирования, включающих как уровень обученности (как измеряемый параметр), так и трудность тестовых заданий (как параметр задания, устанавливаемый при его аттестации), позволяющая преодолеть указанные ограничения на количество испытуемых. Точность улучшается с увеличением кратности копирования исходных наборов и усреднением количества замененных значений (верный / неверный ответ).

Создана программная реализации предложенных методов.

[6] The R Project for Statistical Computing, http://www.r-project.org/

Основной результат

Предложена методика, позволяющая применять модель 3PL для тестов с небольшим количеством испытуемых и позволяющая учитывать сложность предъявленного варианта, что приводит к повышению объективности и точности результатов каждого испытуемого в группе как независимо, так и относительно всей группы.

РАЗРАБОТКА УТИЛИТЫ ДЛЯ АВТОМАТИЧЕСКОГО ИЗМЕРЕНИЯ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ АЛГОРИТМОВ ОБРАБОТКИ ДАННЫХ В СИСТЕМАХ CCNUMA

Д. А. Драган, В. В. Башун, А. Г. Поваляев

Целью данной работы является проведение исследования влияния особенностей архитектуры NUMA на производительность алгоритмов блочной обработки данных. Дается общее описание архитектуры NUMA, а так же её поддержки ядром операционной системы Linux. Предлагается создание программно-аппаратного комплекса для получения эмпирических оценок времени работы алгоритмов при различных условиях доступа к памяти. В работе описана архитектура разработанного стенда.

I. Архитектура ccNUMA

С развитием серверных технологий и повышением производительности процессора и памяти все большую актуальность приобретает архитектура на базе технологии ccNUMA (cache coherency Non-Uniform Memory Access). NUMA — это схема реализации компьютерной памяти с ассиметричным доступом для процессоров [1]. Для такой архитектуры время доступа к памяти, из которой осуществляется операция чтения или записи данных, зависит от расположения памяти относительно процессора.

Рис. 1. Общая модель архитектуры NUMA

На рис. 1 представлено разбиение системы на блоки, в каждый из которых входит вычислительный узел и связанный с ним модуль памяти. Каждый узел состоит из одного или нескольких процессоров. Из данной схемы видно, что обращаясь к своей памяти, узел уменьшает нагрузку на общую шину. Вместе с тем, при обращении к памяти другого узла на общей межпроцессор-

ной шине возникает дополнительная нагрузка. Скорость такого обращения будет значительно ниже, чем при обращении к локальной (связанной с данным вычислительным модулем) памяти.

Производители оборудования предлагают свои реализации серверных аппаратных платформ с поддержкой архитектуры ccNUMA. Среди наиболее распространенных можно выделить Intel Nehalem (начиная с Xeon 5500) [2], AMD Opteron [3, 4] и IBM NUMA-Q (платформы IBM POWER 6/7). На рис. 2 и рис. 3 приведено схематичное представление ccNUMA в реализации от Intel и вариант от AMD с диаграммой последовательности команд для операции считывания из памяти.

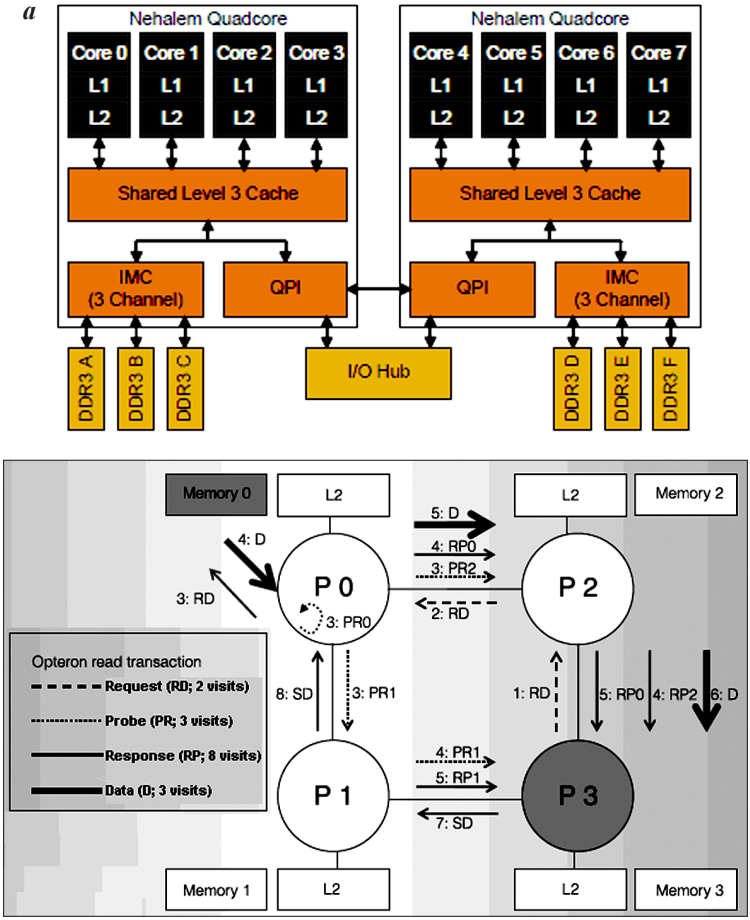


Рис. 2. Модель архитектуры NUMA: *а* — модель архитектуры NUMA от Intel [2]; *б* — модель архитектуры NUMA от AMD с диаграммой последовательности команд

На рис. 3 показана последовательность операций, которые будут выполнены в случае если данные, которые запрашивает процессор, находятся в памяти другого узла. Все потоки данных разделены по типу (запрос — Request, подтверждение — Probe, отклик — Response, данные — Data), обозначена последовательность шагов (16 шагов в сумме). Каждый шаг привносит нагрузку на общую системную шину и тем самым нагружает разделяемый системный ресурс.

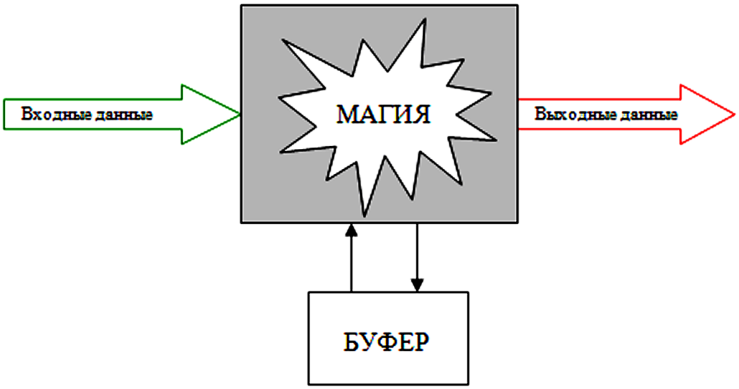


Рис. 3 Схема работы алгоритмов обработки данных

К преимуществам архитектуры ccNUMA можно отнести: возможность построения системы с большим количеством процессоров (повышение производительности за счет уменьшения нагрузки на шину) и масштабируемость (простота разбиения системы на виртуальные независимые машины).

К недостаткам данной архитектуры можно отнести: падение производительности при обращении узла к нелокальной памяти, либо при обращении нескольких узлов к одному участку разделяемой памяти, а также сложность конструирования NUMA систем, связанная с необходимостью поддержания когерентности кэша и памяти. Это связано в первую очередь с большим количество операций (при чтении или записи), а также с асинхронностью выполнения некоторых операций в рамках одной транзакции (рис. 3). Влияние этих негативных факторов можно уменьшить, если перераспределить нагрузку по узлам для более оптимального использования ресурсов.

На данный момент все больше серверных решений переходят на архитектуру ccNUMA. Операционная система Linux активно поддерживает архитектуру ccNUMA начиная с версии ядра 2.6. В частности, в ядро включены различные политики выделения памяти (выделение страниц в памяти заданного узла), автоматическая миграция памяти и т.п. [1]. Существуют отдельные библиотеки, предоставляющие удобный интерфейс для управления политиками выделения памяти и распределением ресурсов в среде с NUMA архитектурой (см. libmuna, [6]).

II. Постановка задачи

Особенности архитектуры NUMA могут воздействовать на производительность алгоритмов обработки данных, хранящихся в оперативной памяти. Целью данной работы является проведение исследования влияния особенностей архитектуры NUMA на производительность алгоритмов блочной обработки данных. Даны алгоритмы блочной обработки данных (например, сжатия и шифрования), с известным интерфейсом и закрытой внутренней реализацией (черный ящик). Алгоритмы работают с тремя областями памяти (см. рис. 3): входных данных, выходных данных и промежуточных данных (буфер). Каждая из областей памяти может находиться на разных узлах, что неизбежно влияет на время работы алгоритма.

Для корректной оценки производительности конкретного алгоритма необходимо иметь возможность численно оценить влияние архитектуры на время его работы. Косвенную оценку можно получить, экспортируя из ядра таблицу весовых коэффициентов (distance table), оценивающую «расстояние» между узлами системы [1,7]. Эти коэффициенты, а так же данные о топологии узлов операционная система получает от ACPI (усовершенствованного интерфейса конфигурации и управления питанием), SRAT (статической таблицы близости ресурсов), а так же SLIT (System Locality Information Table), предоставленные прошивкой. Однако данной информации может быть недостаточно для определения эффективности работы конкретного алгоритма. В каждой системе велико влияние дополнительных факторов (таких как работа VMM при различных загрузках процессора и памяти), значения которых трудно оценить без экспериментальных исследований. Кроме того, степень влияния архитектуры зависит от того, насколько активно данный конкретный алгоритм работает с памятью. Для оценки влияния этих факторов предлагается разработать специальный стенд. Стенд представляет из себя программно-аппаратный комплекс, предназначенный для экспериментальной оценки эффективности работы алгоритмов обработки данных при различных показателях загруженности системы. Общая схема стенда изображена на рис. 4. Стенд состоит из следующих элементов:

1. Аппаратная часть — многопроцессорная система с NUMA архитектурой
2. Хранилище данных — внешний источник данных, с которого поступает обрабатываемая информация
3. Программная часть — утилита сбора эмпирических данных.

Программная часть, в свою очередь, включает в себя:

1. Пул памяти — менеджер оперативной памяти. Отвечает за выделение памяти на всех узлах и ее распределение при работе алгоритмов.
2. Модуль нагрузки — нагружает процессоры, кэш и память заданных узлов.
3. Модуль профилировки — замеряет время работы алгоритмов обработки данных в заданных временных единицах.

4. Библиотека алгоритмов — хранит тестируемые алгоритмы обработки данных.
5. Управляющий модуль — координирует работу системы в целом.

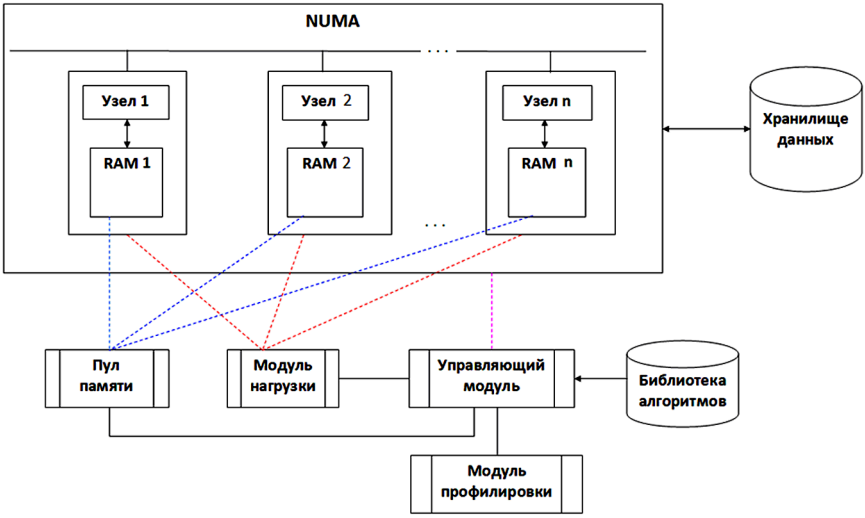


Рис. 4. Схема стенда

III. Описание эксперимента

Для получения эмпирических данных планируется провести следующий эксперимент.

1. На каждом из выбранных для эксперимента узлов предварительно выделяется 3 области памяти:
   a) область входных данных;
   b) область выходных данных;
   c) область промежуточных данных.
2. Во все области входных данных с внешнего хранилища загружаются одни и те же данные.
3. Для каждого алгоритма из библиотеки алгоритмов замеряется время его выполнения при различном расположении входных и выходных данных на узлах системы.
4. Пункт 3 повторяется при различной нагрузке на процессоры и память.

Полученные эмпирические данные помогут оценить, насколько архитектура NUMA влияет на время выполнения конкретного алгоритма. Использование модуля нагрузки позволит сравнивать результаты, полученные на ненагруженной системе с результатами, полученными при разных видах

нагрузки (нагрузка на процессор, память, кэш). Кроме того, появляется возможность сравнивать разные алгоритмы между собой.

Литература

1. *Christoph Lameter.* Local and Remote Memory: Memory in a Linux/NUMA System. Jun 20th, 2006.

2. “Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System”. *D. Molka, D. Hackenberg, R. Schöne, Matthias S. Müller*. 2009. 18th International Conference on Parallel Architectures and Compilation Techniques.

3. “THE AMD OPTERON NORTHBRIDGE ARCHITECTURE” (AMD site).

4. “CACHE Hierarchy And Memory Subsystem Of the AMD Opetron Processor”.

5. “Critical Factors in NUMA Memory Management”. *Jayashree Ramanathan* and *Lionel M. Ni*. Department of Computer Science Michigan State University East Lansing, MI48824-1027

6. http://linux.die.net/man/3/numa

7. *Tony Luck.* “Linux Scalability in a NUMA World”, Linux Magazine, September 14th, 2008 http://www.linux-mag.com/id/6868/

СОЗДАНИЕ И АНАЛИЗ СИСТЕМЫ РАСПРЕДЕЛЕННОГО ХРАНЕНИЯ ДАННЫХ

В. В. Черкалова, М. А. Саламатов, А. В. Сороцкий, А. Н. Сычев, Я. В. Андреев
(4 курс, каф. ИУС)

О. В. Александрова
(асп. каф. ИУС)

Целью данной работы было создание распределенного хранилища данных для дальнейшего его исследования. Основной задачей являлась разработка с минимальным использованием сторонних библиотек и объектно-ориентированного программирвоания. Большая часть работы реализована на языке C.

В работе получены новые научные знания о системах распределенного хранения информации.

В рамках нашего проекта было создано программное обеспечение, объединяющее несколько компьютеров в кластер, для организации распределенной базы данных, обеспечивающей высокий уровень надежности хранения информации.

Нами были поставлены и решены следующие задачи:

1. Создание иерархии внутри машин кластера, подразумевающей наличие нескольких клиентов на каждой из машин, демона, к которому эти клиенты отправляли бы запросы, и правил общения этих сущностей внутри сети, объединяющей машины в кластер.
2. Создание библиотеки, предоставляющей функции для обмена сообщениями между компьютерами по глобальной сети и между клиентами и управляющими сущностями на локальных машинах. Эта библиотека также должна организовать передачу файлов с одной машины на другую.
3. Создание клиентского приложения (далее — клиент), запускаемого на локальной машине и обеспечивающего доступ к файлам, находящимся в распределенной системе. Предусмотреть возможность запуска нескольких клиентов на каждой из машин. Разработать библиотеку, предоставляющую функции для работы с файлом на локальной машине.
4. Создание демона (программы, работающей в фоновом режиме и не имеющей пользовательского интерфейса), принимающего и обрабатывающего сообщения от каждого из локальных клиентов и отправляющего определенные сообщения всем участникам кластера.

При создании проекта мы не использовали дополнительные библиотеки, помимо стандартных, и использовали язык C, что позволило нам создать достаточно надежную систему.

В рамках теоретического исследования мы изучили существующие системы для создания кластеров, а также уровни для передачи сообщений в частности. Среди исследованных нами систем: «Lustre», «Open AIS», «CoroSync», «Open Clovis», «Google file system», «Transis» и другие.

Исследование показало, что созданная нами система обмена сообщениями является одной из наиболее эффективных, поскольку мы поодерживаем передачу форматированных сообщений, что позволяет сильно ускорить их прием и обработку. А это, в свою очередь, повышает эффективность системы в целом.

Также для реализации взаимодействия сущностей внутри кластера нами был изучен ряд алгоритмов, обеспечивающих высокий уровень надежности для подобных систем. Эти алгоритмы включают в себя механизм выбора координатора системы, механизм хранения и передачи базы данных о системе, алгоритмы обмена сообщениями и файлами.

Выводы. В результате нашей работы была получена полностью функционирующая система для распределенного хранения данных.

Литература

1. *Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal* and *P. Ciarfella.* The Totem single-ring ordering and membership protocol // ACM Transactions on Computer Systems. Vol. 13. No. 4 (November 1995). Pp. 311–342.

2. *W. C. H. Cheng, X. Jia* and *S. Kutty.* Towards fault-tolerant and synchronous multicast protocols for dis-tributed systems // Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems. Dijon, France (September 1996). Pp. 418–425.

3. *T. Tachikawa, H. Higaki, M. Takizawa, M. Gerla, M. T. Liu* and *M. Deen*. Flexible wide-area group communication protocols–International experiments // Proceedings of the 1998 Workshop on Architectural and OS Support for Multimedia Applications. Minneapolis, MN (August 1998). Pp. 105–112.

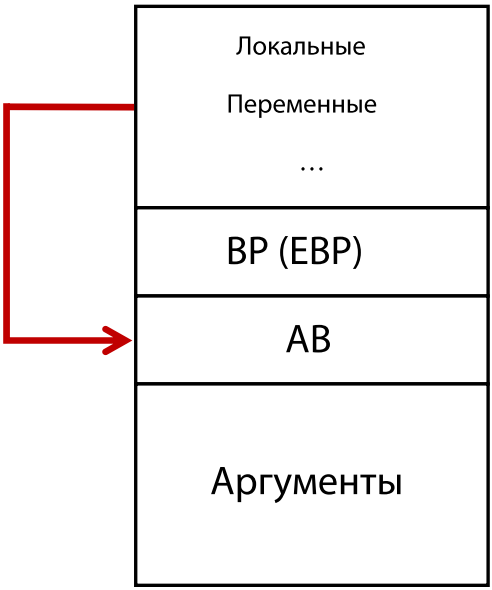
МЕТОДЫ ОБХОДА ЗАЩИТЫ ОТ ИСПОЛНЕНИЯ ДАННЫХ (DEP+ASLR) В ОПЕРАЦИОННЫХ СИСТЕМАХ СЕМЕЙСТВА WINDOWS

В. Д. Стремоухов

СПбГУИТМО, Санкт-Петербург

1. Введение

С момента, когда интернет распространился повсеместно и потеряли актуальность классические файловые вирусы, основным методом заражения целевой машины вредоносным кодом стало исполнение шеллкода на стеке/в куче процесса при реализации какой-либо уязвимости, связанной с переполнением буфера. Как правило, ошибка программиста уязвимого ПО заключается в отсутствии контроля за границами буфера для чего-то типа ASCIIZ-строки в области локальных переменных. Таким образом, при превышении размеров буфера происходит перезапись адреса возврата:



Так, «классическую атаку» условно можно разделить на следующие этапы:

- Поиск уязвимости.
- Формирование шелл-кода.
- Реализация уязвимости в эксплойте (формирование переполнения буфера).

- Формирование стека в эксплойте (шелл-код + подмена адреса возврата).
- Передача управления на шелл-код.

Поскольку ошибки в реализации у программистов будут всегда, очевидно необходимо было фундаментально закрыть данную «дыру», что и попытались сделать Intel и AMD (добавив NX (XD) бит предотвращения исполнения в качестве аттрибута страницы), и Microsoft, включив в XP SP3 технологию DEP (нас интересует прежде всего hardware DEP, как реализация NX (XD) бита, поскольку обход software DEP, по большому счёту, сложности не представляет).

2. Обход DEP

Основная уязвимость технологии hardware DEP, к сожалению, чисто архитектурная: реализуя поддержку DEP, в Microsoft хорошо представляли, какое количество сторонних приложений может столкнуться с проблемой несовместимости из-за постановки под вопрос факта исполняемости той или иной области памяти. Посему, кроме использования глобального системного флага, существует возможность включить или отключить использование DEP для каждого процесса. Отключить поддержку NX можно прямо во время выполнения процесса! Опустив подробности, коротко можно сказать, что для атаки достаточно снять NX-биты у страниц памяти, содержащих шеллкод, вызвав функцию NtSetInformationProcess с определёнными параметрами. Конечно, вызов функции из стека/кучи будет заблокирован той же DEP, но данная проблема легко и очевидно решается при помощи технологии ret2libc — соответствующим формированием стека и подменой адреса возврата на адрес то же NtSetInformationProcess (очевидно, что её код находится в исполняемой области. В докладе также были описаны другие способы обхода DEP). При этом важно, что для новой атаки почти не приходится «изобретать велосипед» — можно использовать те же шеллкоды и т. п., что и раньше.

Как уже было сказано, данная уязвимость является архитектурной и представляет собой компромисс между поддержкой старых приложений и защитой. Кроме того, атака возможна только в случае, если:

1) эксплойт «знает» 2 адреса: ret внутри атакуемой функции, начало функции LdrpCheckNXCompatibility (в случае Software DEP — один) и контролирует стек;
2) Атакуемый процесс может менять сам себе значения NX (XD) битов.

Второе условие, теоретически, можно сделать невозможным, в принципе запретив процессу менять значения NX-битов в своём виртуальном адресном пространстве. Вполне возможно, что в Microsoft ведутся разработки в данном направлении. Обычному же пользователю имеет смысл по возможности искючать из использования старое ПО и работать с DEP в режиме AlwaysOn.

3. Обход ASLR

Для исключения первого условия была разработана технология ASLR (Address Space Layout Randomization), суть которой в случайном размещении в виртуальной памяти системных библиотек при каждой загрузке ОС. Оставив за рамками статьи определённые проблемы реальной случайности адресов при использовании ASLR в её нынешнем виде (здесь исключительно проблемы реализации, в каком направлении двигаться очевидно), рассмотрим, по сути, единственный известный на текущий момент тип атак на DEP + ASLR — использование уязвимостей в just-in-time-компиляторах, коих предостаточно в современных браузерах. Пожалуй, наиболее ярким примером является найденная Дионисом Блазакисом возможность использовать уязвимость ActionScript для исполнения шеллкода в адресном пространстве браузера Internet Explorer. Схема атаки была подробно рассмотрена схема такой атаки. Вкратце механизм атаки выглядит так: в AS пишется скрипт, который содержит строку-шеллкод и реализует определённые уязвимости в самом языке, которые позволяют при смещении регистра EIP на 1 байт исполнять произвольный код (с некоторыми ограничениями). Там и вызывается VirtualProtect, который сбрасывает NX-бит payload-шеллкоду (также в докладе была показана возможность формирования целевого шеллкода внутри JIT-шеллкода).

4. Краткое резюме

- Технология DEP есть компромисс между обратной совместимостью и защитой
- DEP + ASLR в связке дают довольно высокий уровень защиты от эксплоитирования через переполнение буфера, что формирует высокие требования к стороннему ПО, в частности, к активным элементам браузеров

Литература

1. HeapSpray via ActionScript. http://roeehay.blogspot.com/2009/08/exploitation-of-cve-2009-1869.html

2. Dion Blazakis's white paper. http://www.semantiscope.com/research/BHDC2010/BHDC-2010-Paper.pdf

3. Getting Pointers from Leaky Interpreters. http://dion.t-rexin.org/notes/2009/10/29/getting-pointers-from-leaky-interpreters

5. Shellcode finding in windows 7. http://skypher.com/index.php/2009/07/22/shellcode-finding-kernel32-in-windows-7

6. JIT Shellcodes and exploits. http://www.dsecrg.com/files/pub/tools/JIT.zip

СИСТЕМА ХРАНЕНИЯ ДАННЫХ С ГРУППИРОВКОЙ ПО КАТЕГОРИИ

М. А. Тверьянович (e-mail: tvema@rambler.ru),
Д. В. Луцив (e-mail: dluciv@math.spbu.ru)

Санкт-Петербургский Государственный Университет

Проблематика

Данная работа посвящена теме хранения различных данных и их структуризации.

В настоящее время есть проблема ограниченности способов описания, определения данных/файлов в иерархической структуре, основные характеристики которой название, место в иерархии (каталогов), метаданные. Из этого вытекают трудности в нахождении искомых данных/файлов, поскольку память человека устроена так, что характеризует информацию не единым образом и потом при попытке вспомнить, как он ее назвал ранее или где она должна быть, скорее всего, потерпит неудачу или на это потребуется дополнительное время [2]. Так же происходит большая затрата времени на структурирование данных в соответствующую иерархию, что по сути является лишней работой.

Новый подход, который уже достаточно популярен — добавление меток к данным (метка или иначе краткое пользовательское описание каких-либо

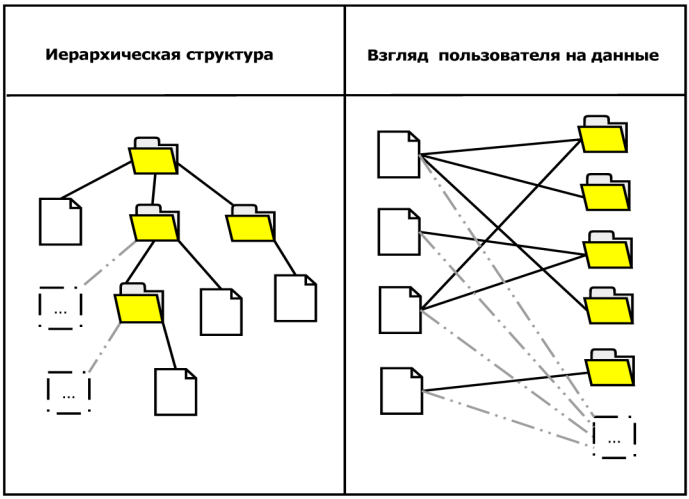


Схема 1. Подходы к структурированию данных

данных, которое, по мнению пользователя, характеризует их). Не потерян взгляд пользователя на данные и их содержимое. Кроме того наличие тех или иных меток у данных может разделять данные на различные категории, по которым впоследствии можно производить поиск.

Для системы хранения данных с их пользовательскими описаниями важна в первую очередь полнота описания этих данных и соответственно скорость, адекватность их поиска, а также удобство в использовании.

Архитектура

Основной частью архитектуры системы хранения данных с пользовательскими описаниями является хранилище данных. Само хранилище данных состоит из базы данных (не иерархической структуры), управляющего модуля для управления информацией о данных и API, посредством которого осуществляется работа с системой извне. Вместе со специализированным приложением — пользовательским интерфейсом — указанные выше компоненты были реализованы в прототипе и описаны в [4].

Сейчас разрабатываются оставшиеся компоненты системы, которые будут описаны ниже.

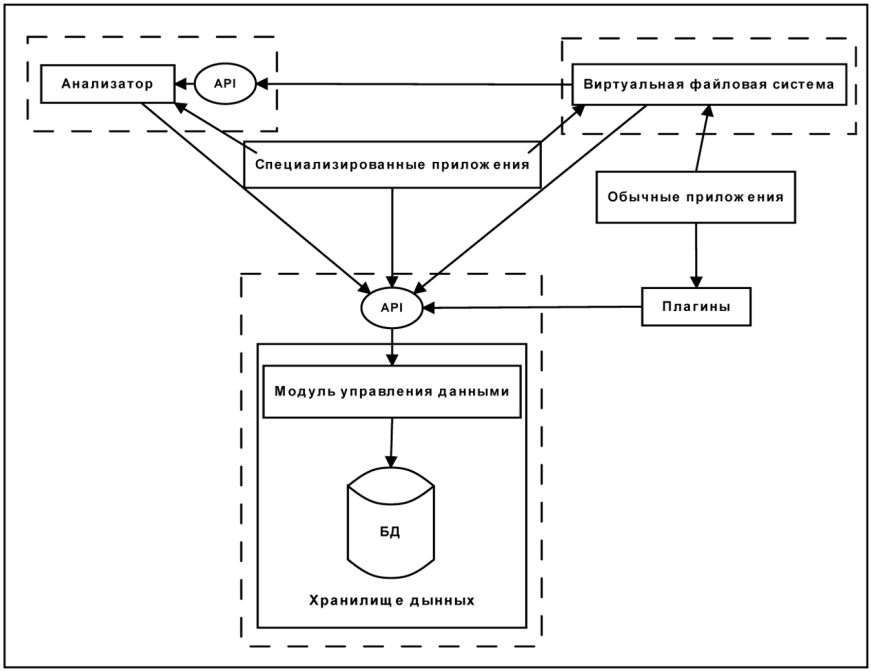


Схема 2. Архитектура системы

Виртуальная файловая система — модуль, отвечающий за имитацию обычной иерархической файловой системы для удобства пользователя и использования обычными программами. Путь в виртуальной файловой системе отображается в запрос к хранилищу данных. То есть, иными словами, при заходе в какой-либо каталог виртуальной файловой системы, происходит запрос к базе данных хранилища с критериями поиска, после чего там отображаются файлы и подкаталоги, полученные как результат выборки.

Анализатор отвечает за автоматическое определение возможных категорий/меток, к каким можно отнести данные. Анализатор работает на основе такой информации как содержимое данных и набор меток, связанный с этими данными. Анализатор должен иметь возможность обучения по мере работы с пользователем и при анализе базы данных. Благодаря анализатору можно попытаться получить более полное описание файла, даже если пользователь сам не достаточно полно опишет данные.

Различные специализированные приложения, написанные специально для работы с данной системой, могут обращаться к хранилищу данных напрямую через API. Обычные же приложения, те которые поддерживают плагины, аналогично могут работать посредством этих плагинов. Если же обычное приложение не поддерживает плагины, то, как говорилось выше, оно может работать с файлами из хранилища через виртуальную файловую систему.

Развитие системы

Хранилище данных

В реализованном прототипе был использован подход с использование реляционной базы данных в качестве варианта неиерархической структуры хранения данных [4]. Однако следует принять во внимание, что человек, если рассматривать подробнее, структурирует информацию аналогично достаточно сложной семантической сети [8]. Поэтому на данном этапе развития системы рассматривается создание хранилища, структурирующего данные на основе семантической сети и дедуктивных БД [5, 6]. При этом, поскольку человек пользуется для идентификации тех или иных данных словами, при построении семантической сети можно воспользоваться информацией из существующих сетей [1, 7].

Анализатор

В качестве подходов к анализу информации о данных, по которым в дальнейшем могут делаться те или иные выводы о принадлежности данных к той или иной категории, могут быть взяты различные методы выявления общего с уже другими категоризированными данными. Например, логический вывод (особенно при использовании дедуктивной базы данных), а также и иные

подходы, как, например, анализ корелляции встречаемости слов в текстах, байесовская классификация [3]. Последний метод представляется весьма подходящим в рамках рассматриваемой задачи, поскольку учитывает содержимое текстовых данных. Кроме того теоретически аналогичные подходы могут быть распространены и на анализ нетекстовых данных, где будет рассматриваться только набор меток, связанный с этими данными.

Список литературы

1. *Азарова И. В., Митрофанова О. А., Синопальникова А. А.* Компьютерный тезаурус русского языка типа WordNet // Материалы конференции «Диалог-2003». М., 2003.

2. *Белянин В. П.* Психолингвистика. Флинта: МПСИ, 2004. 232 с.

3. *Сегаран Т.* Программируем коллективный разум. Пер. с англ. СПб.: Символ-Плюс, 2008. 368 с.

4. *Тверьянович М. А.* Хранилище мультимедиа с группировкой данных по категории. Выпускная квалификационная работа. СПбГУ, 2010. 33 с.

5. *Minker J.* Foundations of deductive databases and logic programming. Los Altos, California: Morgan Kaufmann Publishers, 1988.

6. *Dart P. W., Zobel J.* Conceptual schemas applied to deductive databases // Information Systems. Vol. 13. Issue 3. Elsevier Science Ltd, 1988. Pp. 273–287.

7. *Miller G. A., Fellbaum C., Kegl J., Miller K. J.* WordNet: an electronic lexical reference system based on theories of lexical memory // Revue quebecoise de linguistique. 17(2). 1988. Pp. 181–213.

8. *Sowa J. F.* Semantic Networks. 2006. http://www.jfsowa.com/pubs/semnet.htm

Информационная безопасность

Молдовян
Александр Андреевич

д.т.н., профессор
зам. директора СПИИРАН по информационной безопасности

Фахрутдинов
Роман Шафкатович

с.н.с. СПИИРАН

CODA — НОВАЯ СИСТЕМА КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

М. В. Баклановский

Санкт-Петербургский государственный университет,
Кафедра системного программирования

Сегодня на рынке средств компьютерной безопасности существует серьёзная проблема — имеющиеся технологии защиты практически полностью выработали свой потенциал по скорости реагирования на вновь возникающие угрозы, в то время как темп атакующей стороны продолжает неуклонно расти. Через 2–3 года прогнозируется ситуация, когда ни один из представленных сегодня на рынке продуктов не сможет эффективно противостоять методам киберпреступников. Этим объясняется усиливающийся интерес к технологиям проактивной защиты (следующий шаг в развитии эвристических методов) и активный поиск решений в области искусственного интеллекта (например, CyberHelper Лаборатории Касперского). Элементы искусственного интеллекта — это эффективный путь решения данной проблемы, они смогут стать сглаживающим буфером между быстро возникающими угрозами и сравнительно медленно работающими людьми-аналитиками.

Наш проект позволит эффективно решить проблему безопасности с помощью комплексной системы защиты CODA, которая помимо основного функционала аналогичных современных инструментов предлагает новейший подход скоростного реагирования и необходимых дополнительных способов обороны в условиях быстродействующих типов угроз.

CODA (corporate oracle & dynamic authorization) — новая система компьютерной безопасности. Решает все традиционные для подобных систем задачи. Имеет дополнительные возможности по отслеживанию действий пользователей, по распределённому резервному копированию и другие. Отличается от аналогов самостоятельностью в принятии решений и быстротой реакции на новые формы несанкционированной активности.

Общее описание системы

Центральное место в системе занимает накопитель опыта и знаний далее оракул (по терминологии, предложенной Аланом Тьюрингом). Оракул умеет:

- очень быстро применять имеющиеся у него знания и предотвращать действия, которые носят нежелательный характер;
- самостоятельно определять подозрительные ситуации в процессе работы и осуществлять сдерживание активности тех объектов, действия которых показались ему небезопасными. Для сдерживания у него есть целый ар-

сенал средств, которые он применяет постепенно, в зависимости от активности сомнительного объекта;

- делиться своими сомнениями с другими оракулами и учитывать их мнение в своей дальнейшей работе.

Для принятия окончательного решения о степени опасности или полезности обнаруженной активности пока нужны специалисты-аналитики, но вынесенный ими вердикт очень быстро и взвешенно распространяется по сети оракулов.

Имеется 2 вида оракулов:

- Корпоративный оракул занимается обеспечением безопасности некоторой компьютерной сети и её узлов и имеет своего представителя на каждом узле защищаемой сети это устройство, которое имеет свой процессор, средства защиты и адаптер беспроводной сети, оно подключается к разъёму USB работающей машины, и через несколько минут машина оказывается под защитой системы безопасности;
- Оракул-консультант работает только с другими оракулами и своей группой аналитиков.

Запущенный в эксплуатацию экземпляр системы представляет собой централизованно управляемую наложенную сеть программно-аппаратных мультиагентных узловых комплектов (COSTAR), вживлённых в операционные системы сетевых машин и устройств и имеющих возможности автономной работы в течение сколь угодно больших промежутков времени и быстрого возврата в исходное сообщество по факту восстановления связи с ним.

Для случаев высокой концентрации сетевых машин и устройств (офис) можно, включив в поставку необходимое количество базовых станций, использовать вариант автоматически развёртываемой помехоустойчивой беспроводной сети системы безопасности. При этом

1) снижается нагрузка на корпоративную сеть;
2) повышается защищённость самой системы безопасности (т. е. понижаются риски повреждений частей системы и затрат на их восстановление);
3) упрощается процедура развёртывания системы и её переконфигурации в случае необходимости.

Одним словом, при использовании беспроводной сети понижается стоимость владения системой.

Каждый оракул должен иметь достаточное количество вычислительных и накопительных ресурсов, которое зависит от размеров системы, от требуемого уровня защиты и от ряда других причин. Проблема правильной оценки количества необходимых ресурсов снимается возможностями горячей замены модулей оракула и добавления новых модулей без его перезапуска. Эти же возможности упрощают процедуры технического обслуживания системы и, при необходимости, её обновлений.

Центр управления системой безопасности может состоять из одного или нескольких накопителей опыта и знаний — оракулов. При любом числе используемых оракулов едиными остаются службы:

1) наблюдения за работой системы (мониторинг и отчёты),
2) обучения системы (корректировка систем оценок, редактирование директив и стратегий),
3) запросов к внешним оракулам-консультантам,
4) конфигураций и тактического (в т. ч. ручного) управления.

При этом не выделяется дополнительный уровень логической иерархии, а вместо этого один из оракулов (например, первый, запущенный в эксплуатацию) назначается ответственным за работу единых служб. Такое назначение не даёт ему никаких преимуществ перед другими, работающими в системе оракулами, что приводит к высокой масштабируемости системы на уровне поэтапного подключения подразделений компании.

Корпоративные оракулы могут использовать друг друга в качестве консультантов для обмена накапливаемыми знаниями. Кроме того, с той же целью могут использоваться специализированные оракулы-консультанты, не имеющие своих сетей, но получающие входной информационный поток от своих клиентов (оракулов). Таким образом строится своего рода социальная сеть оракулов, по которой может очень быстро распространяться информация об обнаруживаемых опасностях.

Принципы построения системы

В системе используются как традиционные фон-неймановские вычислители, так и вычислительные модули на основе нейронных процессоров. Нейронные процессоры представляют собой обученные или самообучающиеся нейронные сети, реализованные в виде жёстких кристаллов или на основе программируемых логических интегральных схем (ПЛИС) с блоками оперативной памяти. Логика работы нейропроцессоров является расширением классической бинарной логики и подчиняет себе все архитектурные решения, используемые в системе.

Нейронные сети имеют архитектуру с практически идеальным параллелизмом, поэтому при реализации в «железе» дают прирост производительности на несколько порядков по принципу «чем больше сеть, тем больше прирост». Обычно такая эффективность влечёт за собой большие затраты на интерпретацию получаемых на выходе результатов. Однако, в нашем случае затрат удаётся не только избежать, но и превратить этот «недостаток» в основную особенность архитектуры всей системы.

Базовый принцип построения системы — многозначность как в оценках возникающих ситуаций, так и в решениях о выполнении воздействий.

Система считает все наблюдаемые объекты и даже собственные компоненты подозрительными. Ни про один из объектов она не «думает», что он абсо-

лютно безопасен или, что он безусловно опасен. Все и всегда под сомнением, а сомнение может иметь много значений. В логике воздействий на защищаемые объекты произведён отказ от устаревшей двузначной схемы «оставить/удалить». Вместо этого сформирован и используется большой взвешенный перечень возможных воздействий, направленных на создание различных препятствий и задержек в работе вызывающего сомнения объекта вплоть до полной остановки и удаления. При этом одновременно с воздействиями выполняются операции, направленные на создание возможности последующего отката к исходному (до начала работы сомнительного объекта) состоянию.

У базового принципа есть 2 следствия:

1. Система ведёт себя очень спокойно и ненавязчиво по отношению к пользователям.

Мнение пользователя о степени риска при выполнении им каких-либо действий система конечно же учтёт (например, повысит внутренний уровень опасности по отношению к указанному пользователем приложению), но задавать вопросы и выбрасывать на экран различные сообщения по своей инициативе она не будет. У неё есть временной запас nu — в случае возникновения сомнений достаточно высокого уровня она начнёт «выдавливать» подозрительное приложение сама, и делать это она может достаточно долго для того, чтобы дождаться либо каких-либо более явных проявлений злонамеренности, либо информации об обнаружении чего-то подобного в других системах. При этом она сама в состоянии послать запрос в аналитический отдел своей компании или компании,сопровождающей систему (правда, эту возможность можно отключить в конфигурации).

2. В арсенал возможностей системы включены различные формы резервного копирования.

Резервное копирование — это единственный надёжный способ защиты от потери данных, но его реализация как правило сопряжена с весьма времяёмкими процедурами, начиная от составления планов копирования и заканчивая обслуживанием архива копий. Наличие функционала резервного копирования в сочетании со знанием текущей загруженности и динамики использования корпоративных ресурсов по хранению данных делает систему возможным и естественным организатором этого сложного, но очень нужного дела.

Модульность архитектуры оракулов. Модули — это устройства со своими:

1) процессорами (в т. ч. нейропроцессорами),
2) встроенным программным обеспечением,
3) потребностями в ресурсах (такими, как питание, память и т. п.) и
4) функциональными возможностями.

Модульность обеспечивает масштабируемость и расширяемость оракулов. Специализированные модули-анализаторы могут запрашивать данные с узлов защищаемой сети, организовывать хранение промежуточных резуль-

татов их обработки на внутренних накопителях, предлагать системе варианты воздействий на защищаемые объекты и инициировать обмены мнениями о проблемных ситуациях с другими модулями и оракулами. Помимо возможностей модули имеют ещё и обязанности: проходить централизованное тестирование, использовать протоколы системы, предоставлять свои данные для отчётов и инспекций, выполнять конфигурационные директивы, подчиняться управляющим сигналам и многое другое.

Внутренняя структура оракула невидима из внешнего мира.

Система организует все необходимые внутренние и внешние информационные потоки и поддерживает внутреннюю адресацию модулей, одновременно скрывая её от глаз внешнего наблюдателя. Инициатива любого внешнего информационного обмена может исходить только изнутри системы, а все внутренние адресные метки на границе системы транслируются во временные внешние адреса (и обратно).

Наличие недоступного извне внутреннего информационного пространства даёт возможность отдельного и защищённого хранения атрибутов объектов защиты.

Описания объектов защиты (например, признаки конфиденциальности документов) являются закрытой информацией, и могут порождаться, храниться и использоваться внутри системы, имея одностороннюю привязку к своим объектам. При этом снаружи не появится информации не только о содержании этих описаний, но даже и об их наличии.

Вся внутрисистемная информация транспортируется по внешним каналам в зашифрованном виде.

Всё каналообразующее оборудование снабжено потоковыми криптопроцессорами. Это касается в первую очередь беспроводных каналов, но в случае использования сети общего назначения весь траффик на узлах сети всё равно заворачивается в устройство, являющееся частью системы безопасности, в котором выполняются и криптопреобразования, и трансляция адресов.

Механизмы инспекций используют сложные многоходовые протоколы обмена контекстно-зависимыми сообщениями, для которых задача вычисления правильного продолжения является алгоритмически неразрешимой.

Алгоритмы обслуживания всех интерфейсных стыков часто меняются. Эти изменения, конечно, подчинены некоторым закономерностям, но закономерности учитывают историю изменений. Не зная историю, можно попробовать её вычислить, но в нашем случае такое вычисление намного сложнее обычного перебора вариантов. По сложности эта задача относится к классу алгоритмически неразрешимых проблем.

Система экономит все используемые ей внешние ресурсы.

Агенты системы работают на всех узлах защищаемой сети, занимая их память и процессоры. Система использует сеть общего назначения даже

в случае беспроводной конфигурации. Все случаи использования ресурсов защищаемой системы насколько это возможно учитываются и контролируются, на всех узла выставляются ограничения по использованию ресурсов, и эти ограничения можно менять. Разумеется, такие изменения могут сильно усложнить решение основных задач системы, но за редчайшими исключениями она будет подчиняться ограничениям, хотя и отразит эти факты в отчётах о своей деятельности.

Система поддерживает множество ролей пользователей с разными возможностями наблюдения за работой системы и управления ей. Поддерживаются групповые уровни доступа для выполнения критических команд.

В системе используется расширение модели «управление доступом на основе ролей (RBAC)». Так право выполнения некоторых операции (например, остановка системы) может выдаваться не целиком, а по частям, и для выполнения такой операции необходимо одновременное (т. е. выраженное за небольшой промежуток времени) согласие нескольких пользователей.

Управление системой строится на основе рекомендаций и директив, которые транслируются в сценарии изменения настроек.

Системе передана значительная части функций, в прошлом выполнявшихся людьми. Понятно, что при этом очень сильно выросла сложность системы (число параметров, которым можно описать текущее состояние системы). Не все сочетания настроек являются допустимыми, и разрешением возникающих конфликтов занимается система. Не все переходы между состяниями могут выполняться мгновенно, и поэтому система строит последовательность переходов, ведущую к намеченной пользователем цели (сценарий). Система не выполняет одновремено разные сценарии, а объединяет их в один. Сценарии могут меняться в процессе выполнения, имеют ограниченное время жизни и не обязаны всегда завершаться успешно. При этом всегда есть возможность жёстко изменить состояние системы, но это критическая команда.

Архитектурное решение, предложенное нашей группой, по степени интеграции всех перспективных способов противостояния киберугрозам, не имеет аналогов на сегодняшнем рынке. Неслучайно в конкурсе на разработку новой системы комплексной защиты корпоративной сети, проводившемся Лабораторией Касперского в июне-октябре 2010 года, наша разработка заняла 1 место — разработка была представлена исследовательской группой:

- Баклановский Максим (СПбГУ);
- Жуков Николай (СПбГУ);
- Журавлёв Николай (УрГУ);
- Зеленчук Илья (УрГУ);
- Сабашный Вадим (Ланит-Терком);
- Татищев Виктор (Ланит-Терком).

АНТАГОНИСТИЧЕСКАЯ МОДЕЛЬ ИНФОРМАЦИОННОГО ПРОТИВОДЕЙСТВИЯ В ВОЕННОЙ СФЕРЕ. ПРОБЛЕМЫ, ПУТИ РЕШЕНИЯ

В. И. Емелин
ОАО «НИИ Вектор»

А. А. Молдовян
СПИИРАН

Информационное противоборство предполагает выполнение функций информационного нападения и защиты с каждой стороны. Целью каждой стороны при решении задач информационной безопасности (ИБ) является обеспечение процесса управления военными объектами, а при информационном нападении — дезорганизация процесса управления противника. Представленная на рис. 1 теоретико-множественная модель описывает такой процесс как антагонистическую игровую модель с двумя игроками, стратегиями которых являются: формирование и реализация мероприятий по информационным воздействиям (ИВ) и мероприятий по обеспечению ИБ автоматизированных систем управления (АСУ) военными объектами. Важным явлением, которое определяет существенные, но не в полном объеме учитываемые в существующей теории условия функционирования АСУ военными объектами, являются наличие предыстории, содержащей перечень информационных атак и нарушений требований информационной безопасности в течение всего периода жизненного цикла АСУ. Жизненный цикл АСУ военного назначения определяется как совокупность взаимосвязанных процессов, включающих последовательное изменение её состояний: от формирования исходных требований до окончания эксплуатации и утилизации комплекса средств автоматизации. На всем жизненном цикле выполняются действия по обеспечению информационной безопасности, причем с каждым этапом перечень защищаемой информации и объектов ее хранения последовательно расширяется, меняется модель угрозы, а также меняется набор задач обеспечения информационной безопасности. Со временем рост своеобразного послужного списка информационных атак свидетельствует о возможной утечке системной информации (ID пользователей; даты и времени входа и выхода; идентификаторов терминалов и т. д.).

При анализе состояний графа, описывающего процесс информационного противоборства взаимодействующих сторон, были выявлены следующие устойчиво проявляющиеся закономерности:

1) требования к конфиденциальности и доступности информации практически остаются неизменными на всех этапах жизненного цикла;

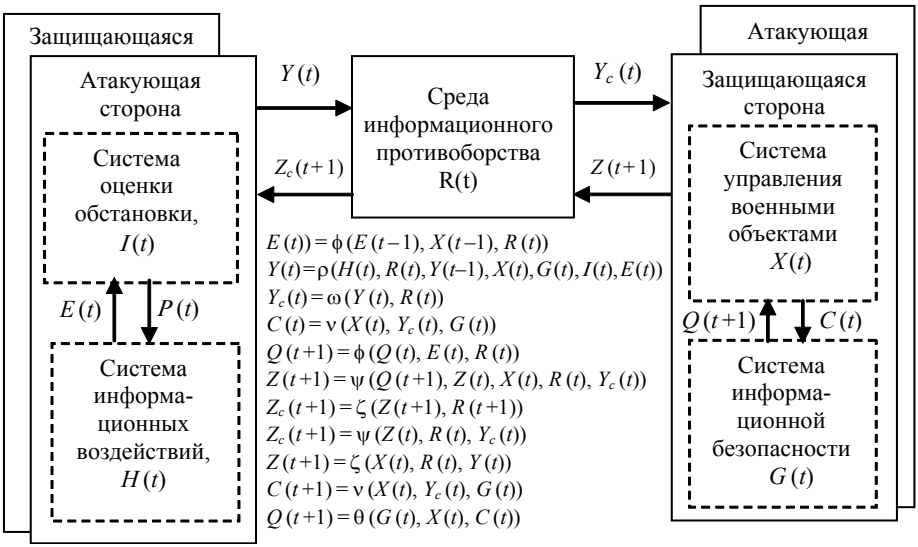


Рис. 1. Описание процесса информационного противоборства в пространстве состояний сторон:

$X(t)$ — состояние АСУ ВО; $G(t)$ — состояние системы ИБ; $R(t)$ — состояние среды; $I(t)$ — состояние системы оценки обстановки противоборствующей стороны; $P(t)$ — разведка данных о состоянии АСУ ВО; $E(t)$ — реализация мероприятий ИВ; $H(t)$ — состояние противоборствующей стороны; $Y(t)$ — информационное воздействие на АСУ ВО по результатам оценки обстановки; $Y_c(t)$ — информационное воздействие на АСУ с учетом влияния среды; $C(t)$ – обнаружение информационного воздействия на АСУ ВО; $Q(t+1)$ – реализация системы мер ИБ; $Z(t+1)$ — информационное воздействие на противника при решении задачи активного противодействия в $t+1$; $Z_c(t+1)$ – информационное воздействие на противника с учетом влияния среды в $t+1$

2) требования к целостности информации вырождаются на этапе эксплуатации АСУ ВН в требования целостности программного обеспечения (ПО);
3) временным интервалом, наиболее благоприятным для утечки, утраты информации и модификации ПО, является этап разработки АСУ ВН, что требует учета зарегистрированных деструктивных воздействий на этапе ее эксплуатации;
4) кардинальное изменение условий функционирования АСУ ВН (сдаточную команду сменяет экипаж, система работает в реальном масштабе времени управления объектами ВН), требует разработки новых методов оценки и обеспечения информационно-психологической и информационно-технической безопасности [1].

Для учета причинно-следственных зависимостей между разнородными событиями информационного противоборства, которые могут происходить на различных этапах жизненного цикла автоматизированных систем управ-

Рис. 2. Система показателей информационной безопасности на этапе эксплуатации АСУ ВН

ления, требуется комплексная взаимоувязанная система оценок (рис. 2). Использующаяся в настоящее время для оценки информационной безопасности система стандартов имеет следующие недостатки:

1) стандарты основаны на «пороговом воздействии» и не учитывают эффект слабых атак, выполненных на заключительной стадии информационного противоборства;

2) в стандартах не учитывается синергетическое влияние нескольких факторов;
3) стандарты редко применимы для учета уникальных условий протекания процессов.

Следует также отметить, что применяющиеся в настоящее время в стандартах критерии информационной безопасности (конфиденциальность, целостность, доступность) не связаны с критериями оценки управляемых процессов.

Созданная система оценок позволяет разрешить ряд существенных противоречий между требованиями практики и состоянием теории информационной безопасности. Наиболее острое из этих противоречий связано с появлением новых угроз информационной безопасности и отсутствием адекватных методов защиты от них. Системный анализ показывает, что в этом случае требуется новое комплексное решение по защите семантической составляющей информации от интеллектуальных помех, как нового вида информационного оружия. Требования нового подхода к обеспечению информационной безопасности АСУ ВН основаны на формализованном описании выявленных законов и закономерностей защиты, знание которых используется для повышения качества информации, и, в конечном итоге, для повышения боевой мощи военных объектов [2]. Следует отметить, что частные попытки решения отдельных задач безопасности на каждом этапе жизненного цикла, ориентированные на достижение целей данного этапа, не обеспечивают решение задачи в полном объеме. Теоретическая и практическая не проработанность этих вопросов, приводит к тому, что на этапе эксплуатации обслуживающий персонал может столкнуться с серьезными осложнениями, обусловленными тем, что произошла утечка, утрата или модификация информационного и программного обеспечения на ранних этапах жизненного цикла АСУ.

Литература

1. *Емелин В. И.* Метод оценки выполнения требований информационной безопасности пользователями автоматизированных систем / В. И. Емелин, А. А. Молдовян // Вопросы защиты информации. 2007. № 4 (79). С. 49–52.

2. *Емелин В. И.* Метод информационного управления для защиты баз данных автоматизированных систем в социально-экономической сфере деятельности / В. И. Емелин, А. А. Молдовян // Вопросы защиты информации. 2007. № 4 (79). С. 78–80.

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ И ИНФОРМАЦИОННАЯ УСТОЙЧИВОСТЬ КРИТИЧЕСКИХ СИСТЕМ

В. И. Емелин
ОАО «НИИ Вектор»

А. А. Молдовян
СПИИРАН

1. Основные тенденции развития фундаментальных понятий теории информационной безопасности

Под термином критические системы будем понимать [4] сложные компьютеризованные организационно-технические и технические системы, блокировка или нарушение функционирования которых потенциально приводит к потере устойчивости систем государственного управления и контроля, экологическим и технологическим катастрофам, утрате обороноспособности государства. Вопросам информационной безопасности (ИБ) критических систем в настоящее время уделяется большое внимание, результатом активного обсуждения ряда проблем является последовательное развитие теории информационной безопасности. Динамика происходящих изменений в теории ИБ в части определения ее основных понятий (объекта, функциональных требований, механизмов обеспечения безопасности) показана на рис. 1. Как следует из представленных результатов анализа, понятие целостности в последнее время периодически уточняется. Так, например, в ГОСТе Р ИСО/МЭК 17799-2005, введенным в действие с 01.01.2007, под целостностью понимается достоверность и полнота информации, а также методов ее обработки [5]. Такое определение является определенным шагом вперед, однако вносит элемент неоднозначности в понимание данного термина. С одной стороны, очевидно, что и под этим определением интуитивно угадывается требование обеспечения гарантий существования информации в исходном неизменяемом виде. С другой стороны, введение новой категории «достоверность и полнота информации» свидетельствует о намерении «вторжения на территорию» смыслового анализа информации.

Об этом, в частности, свидетельствуют определенные этим же ГОСТом требования о проверке корректности ввода-вывода данных (например, проверки на правдоподобие с целью определения, являются ли выходные данные приемлемыми). В этой связи следует отметить, что используемые в существующей теории ИБ понятие целостности не полной мере отражает ряд важных явлений, определяющих условия функционирования критических систем.

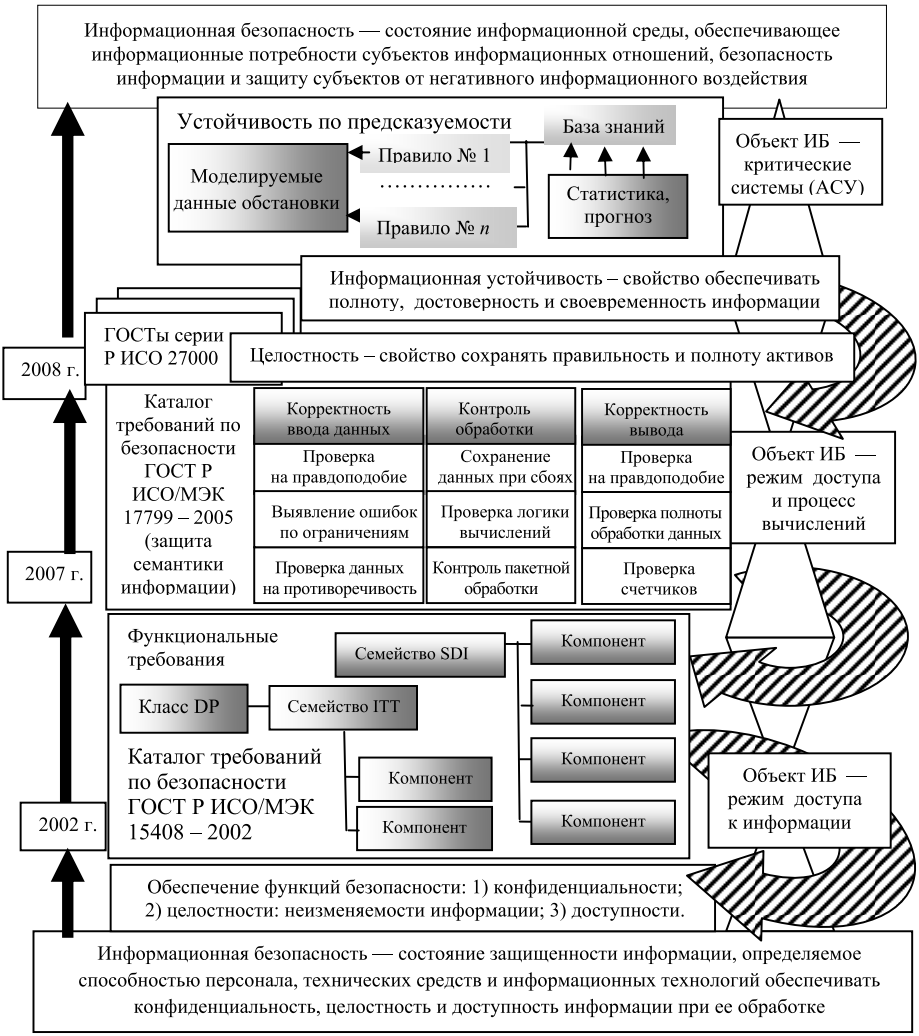


Рис. 1. Схема формирования витков информационного противоборства в процессе развития средств и методов защиты и нападения

2. Требования практики к развитию системы информационной безопасности критических систем

В Доктрине информационной безопасности РФ наиболее уязвимым объектом в условиях чрезвычайных ситуаций определяется система принятия решений по оперативным действиям (реакциям), связанным с возникно-

вением и развитием критических ситуаций. Включение фактора времени, одного из основных параметров любой эволюции, требует введение показателей, которые должны быть критичными к информационным воздействиям. В связи с этим, по нашему мнению, следует сохранить исторически сложившееся понятие целостности информации и дополнительно в перечень основных категорий информационной безопасности включить понятие информационной устойчивости. Это понятие определяется, как способность системы обеспечить заданный уровень качества информации (полнота, достоверность и оперативность) АСУ в условиях случайного или преднамеренного воздействия (искажения) на всех стадиях жизненного цикла. Указанное определение непосредственно вытекает из определения устойчивости управления [2], как характеристики системы, построенной на основе меры возмущающего воздействия, превышение которой ведет к выходу векторов ошибки управления за допустимые пределы. Важным условием существования этого понятия является требование целостного рассмотрения процесса на всех стадиях жизненного цикла. Слово «устойчивый» в рассматриваемом аспекте по отношению к информации означает, что система способна реагировать на изменения в возмущающих воздействиях и сохранять свое состояние на заданном качественном уровне на протяжении всего периода времени. Проблемную ситуацию при оценке устойчивости в условиях неравновесия [3] порождают уязвимости, которые возникают при информационном воздействии на средства добывания и передачи данных, что приводит к снижению полноты и достоверности информации, возникновению дезинформации, снижению оперативности и старению информации. Отметим, что процессы информационного противоборства имеют нелинейный характер: при плавном изменении параметров информационной атаки в некоторый момент времени происходит скачкообразное изменение информационной безопасности.

3. Метод оценки информационной устойчивости критических систем

Для выбора метода оценки устойчивости автоматизированной системы в условиях информационного противоборства определим понятие информационного потенциала, как совокупность информационных ресурсов (средств, методов и условий), которой соответствует определенный уровень качества информации. В соответствии с указанными требованиями информационный потенциал АСУ критических систем будем характеризовать следующими параметрами:

1) синтаксическим потенциалом, который определяет количество необходимой информации (элементов обстановки);
2) семантическим потенциалом, определяющим достоверность информации по элементам обстановки;

3) прагматическим потенциалом, который измеряется в оценочных единицах целевой функции.

Считается, что семантический и синтаксический потенциалы, определяющие достоверность D_i и полноту Ni информации, в рамках данной модели характеризуются нелинейной зависимостью по отношению к целевой функции. Переменные состояния D_i и Ni, от которых зависит значение потенциальной функции катастрофы сборки, по существу являются обобщенными координатами созданной модели информационной устойчивости АСУ критических систем.

Как показано в [1], катастрофа складки является структурно устойчивой локальной бифуркацией, возникающей при изменении одного управляющего параметра. Основное содержание этой зависимости определяет качество информации: если информация является полной и достоверной по всему перечню параметров, то будет принято наиболее рациональное управляющее решение R_1. В левой части рис. 2 изображен вариант моделирования процесса информационного противоборства в одномерном фазовом пространстве, когда в результате предпринимаемых действий последовательное повышение информационного потенциала приводит к скачкообразному изменению в выборе наиболее рационального плана действий R_2.

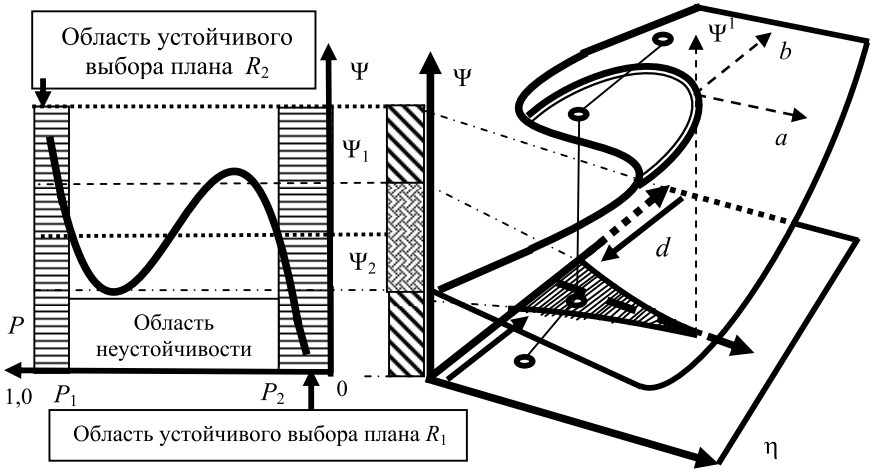


Рис. 2. Вариант моделирования процесса принятия решения складкой Уитни, моделирования обстановки сборкой Уитни

В указанной постановке оценка состояния информационного потенциала АСУ критической системы полностью описывается расположением обобщенной координаты в пространстве, описываемом потенциальной функцией

$$F(x, a, b) = \frac{1}{4}x^4 + \frac{1}{2}x^2 a + b.$$

В заключении еще раз отметим важность ведения работ в данном направлении в связи с наличием устойчивой тенденции смещения информационного противоборства в область смыслового анализа и синтеза информации.

Литература

1. *Арнольд В. И.* Теория катастроф. М.: Изд-во МГУ, 1983.

2. *Емелин В. И.* Антикризисное управление как метод обеспечения информационной безопасности социально-экономических систем / В. И. Емелин, Т. П. Гершкович, А. А. Молдовян // Научно-технические ведомости СПбГПУ. Санкт-Петербургское изд-во политехнического университета. 2007. № 3–1 (51).

3. *Пригожин И.* Время, хаос, квант: К решению парадокса времени. Пер. с англ. / Пригожин И., Стенгерс И. // М.: Прогресс, 1994. 266 с.

4. *Юсупов Р. М.* Наука и национальная безопасность. СПб.: Наука, 2006.

5. Национальный стандарт Российской Федерации. Информационная технология. Практические правила управления информационной безопасностью. ГОСТ Р ИСО/МЭК 17799-2005.

О ПРОГРАММНЫХ СРЕДСТВАХ КОНТРОЛЯ ЦЕЛОСТНОСТИ ИНФОРМАЦИИ

Д. М. Латышев

Санкт-Петербургский институт информатики и автоматизации РАН

Одним из факторов обеспечения безопасности информации, представленной в электронном виде, является защита её целостности.

Существует значительное количество программных средств для контроля целостности информации.

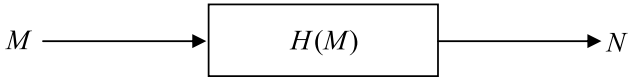
Объектами для контроля целостности могут выступать следующие данные:

- Содержимое файлов.
- Суммарное содержимое папки.
- Файловые атрибуты («Только для чтения», «Скрытый», «Архив», «Системный»).
- Время создания файла/папки.
- Время модификации файла.
- Права доступа.
- Размер файла/папки на диске.
- Количество альтернативных потоков NTFS.
- Содержимое альтернативных потоков NTFS.
- Данные реестра.
- Отдельные области памяти.
- Содержимое главной загрузочной записи (MBR).
- Содержимое логического диска.
- Содержимое физического диска.

Важно также отметить важность возможности контроля целостности содержимого файлов, вне зависимости от типа и формата файла, а также данных большого объема.

Один из способов проверки целостности информации является хранение эталонных копий, используемых для сравнения. Однако при необходимости частых проверок целостности данных большого объема такая процедура приводит к существенным временным задержкам, поэтому она применяется только в специальных случаях. Более технологичным является способ, основанный на вычислении по сложным законам некоторых контрольных сумм. Вычисляются контрольные суммы для массивов данных (например, файлов, каталогов), находящихся в так называемом эталонном состоянии. Эти эталонные контрольные суммы записываются и затем используются для проверки целостности информации. В этом случае проверка целостности состоит в вычислении по заданному алгоритму контрольной суммы для данного блока информации и сопоставлении полученного значения с эталонным

значением контрольной суммы. Выигрыш состоит в том, что теперь нет необходимости проводить сравнение двух больших массивов данных. Целостность проверяется путем сравнения, например, 128-битовых или 256-битовых контрольных сумм. Такие контрольные суммы называются защитными контрольными суммами (ЗКС) [1].

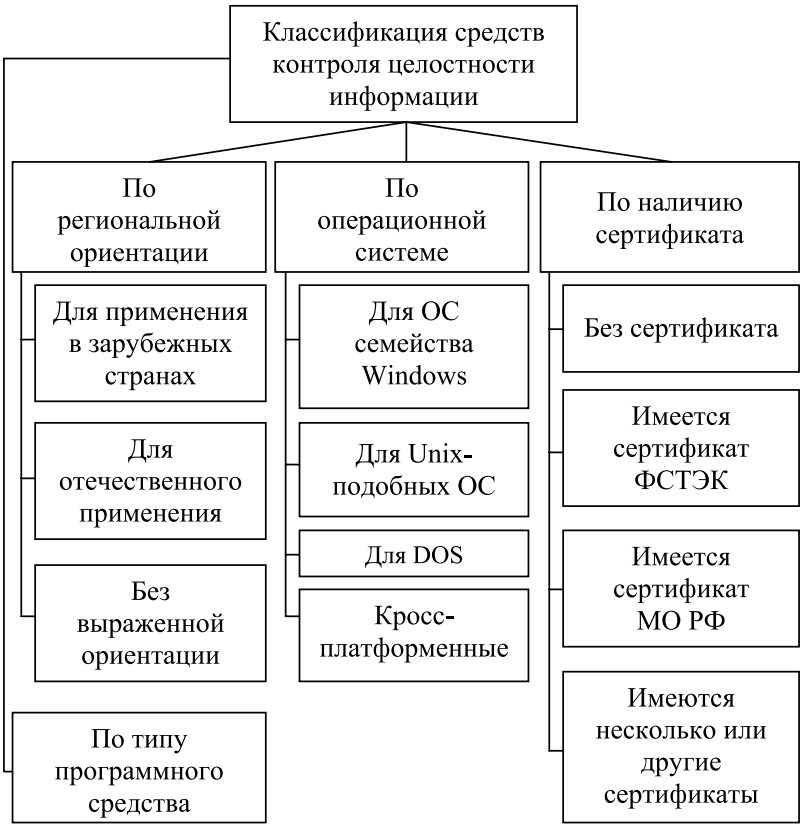


M — сообщение произвольной длины; *H*(*M*) — применение алгоритма вычисления ЗКС к *M*; *N* — сообщение фиксированной длины

Важнейшим типом ЗКС являются хэш-функции. К хэш-функциям предъявляются наиболее жесткие требования по сравнению с другими типами алгоритмов ЗКС. Различные программные средства контроля целостности используются для контроля различных объектов, но по большей части, они в том или ином случае используют контрольное суммирование и, в частности, хэш-функции. Среди используемых алгоритмов вычисления хэш-функций распространены MD5 и алгоритмы семейства SHA. Среди отечественных программных средств контроля целостности, распространено использование алгоритма ГОСТ Р 34.11-94. Таким образом, средства, позволяющие вычислять хэш-функции, можно использовать в той или иной степени для контроля целостности информации.

Можно выделить следующие типы программных средств, пригодных для контроля целостности информации:

- Программные средства широкого профиля, имеющие возможности для контроля целостности информации. Задачей данных средств, как правило, является комплексная защита автоматизированных рабочих мест, одним из аспектов которой является контроль целостности.
- Специализированные программные средства для контроля целостности информации широкого применения.
- Специализированные программные средства для контроля целостности информации частного применения. Отличием данных программных средств является то, что они могут применяться для ограниченного класса объектов.
- Программные средства, имеющие возможность вычисления хэш-функций.
- Специализированные программные средства для вычисления хэш-функций.
- Онлайн-калькуляторы хэш-функций.
- Программные библиотеки для контроля целостности информации или вычисления контрольных сумм.



В качестве рассмотрения, наибольший интерес представляют собой специализированные средства для контроля целостности информации широкого применения, так как они включают в себя наибольший спектр возможностей, необходимых для контроля целостности данных, и в то же время не содержат функции, усложняющие и выходящие за рамки данного рассмотрения.

Необходимо также пояснить классификацию по региональной ориентации средства. В зарубежных странах для средств контроля целостности данных играют роль иные моменты, нежели чем в отечественных продуктах. В частности, за рубежом, большое внимание уделяется способности программного средства контроля целостности использоваться для соответствия IT-систем таким стандартам, как PCI-DSS (Payment Card Industry Data Security Standard) и NIST 800-53. Для отечественных же программных средств большую роль играет наличие государственных сертификатов и возможность производить контрольное суммирование по алгоритму ГОСТ Р 34.11–94.

Среди зарубежных коммерческих продуктов выделяются Tripwire Change Audit [2][3], Verisys File Integrity Monitoring System, Xintegrity Professional, nCircle File Integrity Monitoring [4]. Из продуктов с открытым исходным кодом, можно упомянуть Open Source Tripwire [5], AIDE (Advanced Intrusion Detection Environment) [6], AFICK (Another File Integrity Checker) [7].

Из отечественных программных средств контроля целостности информации можно выделить следующие продукты:

- Семейство программных средств «ФИКС». Разработано ЗАО «ЦБИ-Сервис». Семейство включает в себя продукты «ФИКС» различных версий, «ФИКС-Unix 1.0», «ФИКС-DOS 1.0». На программные средства имеются сертификаты ФСТЭК [8].
- Программные средства «Трафарет» и «Трафарет 2.0». Разработаны ЗАО «Лаборатория противодействия промышленному шпионажу». На программные средства имеются сертификаты ФСТЭК [9].
- Средство контрольного суммирования «КС-ВС». Разработано ЗАО «НПО «ЭШЕЛОН». На программное средство имеется сертификат МО РФ [10].
- Средство контроля эталонного состояния «Янтарь». Было разработано «НИИ «Вектор». Особенностью данной разработки является то, что она ориентирована не на промышленное производство, а на использование в учебном процессе.

Литература

1. *Молдовян Н. А., Молдовян А. А.* Введение в криптосистемы с открытым ключом. Спб.: БХВ-Петербург, 2005.
2. http://www.tripwire.com/
3. http://soft.softline.ru/tripwire-inc/tripwire-change-audit/
4. http://www.windowsecurity.com/software/File-integrity-checkers/
5. http://sourceforge.net/projects/tripwire/
6. http://aide.sourceforge.net/
7. http://afick.sourceforge.net/
8. http://www.cbi-info.ru/groups/page-344.htm
9. http://www.labpps.ru/?part=company&subsection=13
10. http://npo-echelon.ru/production/83/4293

ПОСТРОЕНИЕ И АНАЛИЗ КРИПТОСХЕМ НАД КОНЕЧНЫМИ НЕКОММУТАТИВНЫМИ ГРУППАМИ ВЕКТОРОВ

А. А. Горячев

Санкт-Петербургский институт информатики и автоматизации РАН

В последнее время значительный интерес в области синтеза криптосхем с открытым ключом связан и использованием задач над некоммутативными группами и кольцами. Над некоммутативными алгебраическими структурами формулируются задачи поиска сопрягающего элемента и дискретного логарифмирования в скрытой коммутативной подгруппе, которые перспективны для построения протоколов открытого согласования секретного ключа и открытого шифрования. Построение быстродействующих криптосхем над некоммутативными группами и кольцами связано с применением конечных алгебраических структур такого типа. Ранее было показано, что применение конечных некоммутативных групп векторов, заданных над простым полем, обеспечивает уменьшение размера открытого ключа по сравнению со случаем синтеза криптосхем над конечными кольцами матриц размерности $n \times n$. Однако последние могут быть легко синтезированы для произвольных значений $n > 1$, а в случае векторов известны решения для сравнительно небольших значений их размерности m.

В настоящей работе рассматриваются особенности задания задачи дискретного логарифмирования в скрытой циклической подгруппе для синтеза на ее основе криптосхем с открытым ключом.

1. Задание конечных колец векторов

Конечные кольца векторов могут быть легко заданы над конечным векторным пространством путем определения над векторами некоторой ассоциативной операции умножения, используя стандартную операцию сложения векторов как сложение одноименных координат. При этом для заданного векторного пространства имеется достаточно большое число различных вариантов задания операции умножения векторов, которые определяют конечные кольца с различными свойствами. Вектора размерности m представляются в виде $(a, b, ..., q)$ или в виде $a\mathbf{e} + b\mathbf{i} + ... + q\mathbf{w}$, где $\mathbf{e}$, $\mathbf{i}$, ..., $\mathbf{w}$ — некоторые формальные базисные вектора; a, b, ..., q — координаты вектора, являющиеся элементами конечного поля $GF(p^s)$, где p — простое число (характеристика поля) и s – степень расширения поля ($s \geq 1$). Сложение векторов определяется по стандартной формуле:

$$(a, b, ..., q) + (x, y, ..., z) = (a + x, b + y, ..., q + z).$$

Одноразмерные вектора вида $e\mathbf{v}$, где $e \in GF(p^s)$ и $\mathbf{v} \in \{\mathbf{e}, \mathbf{i}, ..., \mathbf{w}\}$ — некоторый формальный базисный вектор, представляют собой компоненты вектора. Операция умножения векторов определяется как попарное перемножение всех компонентов векторов-сомножителей по формуле:

$$(a\mathbf{e} + b\mathbf{i} + ... + q\mathbf{w}) \circ (x\mathbf{e} + y\mathbf{i} + ... + z\mathbf{w}) = ax\mathbf{e} \circ \mathbf{e} + ay\mathbf{e} \circ \mathbf{i} + ... + az\mathbf{e} \circ \mathbf{w} +$$
$$+bx\mathbf{i} \circ \mathbf{e} + by\mathbf{i} \circ \mathbf{i} + ... + bz\mathbf{i} \circ \mathbf{w} + ... + qx\mathbf{w} \circ \mathbf{e} + qy\mathbf{w} \circ \mathbf{i} + ... + qz\mathbf{w} \circ \mathbf{w},$$

в которой произведения пар базисных векторов заменяются на однокомпонентный вектор, например $e\mathbf{v}$, задаваемый некоторой таблицей умножения базисных векторов (ТУБВ). Координаты однокомпонентных векторов, присутствующие в ТУБВ, называются структурными коэффициентами. Если определенная с помощью ТУБВ ассоциативная операция умножения является коммутативной (некоммутативной), то конечное векторное пространство является векторным конечным коммутативным (некоммутативным) кольцом.

Таблица 1

Задание некоммутативной ТУБВ для случая $m = 4$ (структурные коэффициенты ε и ρ равны –1)

Базисные векторы	Базисные векторы			
	e	**i**	**j**	**k**
e	**e**	**i**	**j**	**k**
i	**i**	$\rho\varepsilon$**e**	ρ**k**	ε**j**
j	**j**	ε**k**	$\rho\varepsilon$**e**	ρ**i**
k	**k**	ρ**j**	ε**i**	$\rho\varepsilon$**e**

2. Особенности криптосхем над конечными некоммутативными кольцами

Одним из перспективных примитивов для построения криптосистем с открытым ключом является задача дискретного логарифмирования в скрытой подгруппе некоммутативной конечной группы. Данная задача состоит в вычислении элемента некоммутативной группы X и числа x (пара этих элементов служит секретным ключом) в уравнении $Y = X \circ G^x \circ X^{-1}$ (элемент Y используется в качестве открытого ключа), где пара умножений на взаимно обратные элементы X и X^{-1} реализует операцию автоморфизма; G — элемент достаточно большого простого порядка (элементы G и Y считаются известными). Решение этого уравнения представляет собой самостоятельную трудную вычислительную задачу, отличную от задачи дискретного логарифмирования. При известном X возникает задача дискретного логарифмирования $Y' = X^{-1} \circ Y \circ X = G^x$. При известном x возникает задача поиска сопрягаю-

щего элемента X, которая в ряде работ рассматривается как вычислительно сложная в случае ее задания над некоммутативными группами специальных типов, однако наши результаты показывают, что в случае конечных некоммутативных групп матриц и векторов сложность задачи поиска сопрягающего элемента является полиномиальной. Тем не менее одновременное нахождение X и x является сложной задачей, несмотря на то, что существует очень большое число различных решений (если X и x — некоторое решение, то для произвольного элемента группы λ, коммутирующего со всеми другими элементами, пара λX и x также является решением). Причем задача дискретного логарифмирования непосредственно не сводится к решению двух независимых задач — дискретного логарифмирования и поиска сопрягающего элемента. При построении криптосхем требуется задать простой механизм, согласования секретных значений X_1 и X_2, которые должны быть выбраны из одной и той же коммутативной подгруппы. Этот вопрос решается заданием дополнительного известного элемента Q, такого что $Q \circ G \neq G \circ Q$, который также обладает большим простым порядком. В этом случае открытый ключ вычисляется как

$$Y = Q^w \circ G^x \circ Q^{-w},$$

где (w, x) — личный секретный ключ. В этом случае схема открытого согласования ключа работает следующим образом. Пусть два удаленных абонента имеют открытые ключи $Y_1 = Q^{w_1} \circ G^{x_1} \circ Q^{-w_1}$ и $Y_2 = Q^{w_2} \circ G^{x_2} \circ Q^{-w_2}$, где (w_1, x_1) и (w_2, x_2) — личные секретные ключи первого и второго абонентов. После открытого обмена открытыми ключами первый абонент вычисляет общий секретный ключ по формуле $K_{12} = Q^{w_1} \circ Y_2^{x_1} \circ Q^{-w_1}$, а второй – по формуле $K_{21} = Q^{w_2} \circ Y_1^{x_2} \circ Q^{-w_2} = K_{12}$. Комбинируя аналогичным образом операцию автоморфизма и операцию возведения в большую дискретную степень, можно построить алгоритмы коммутативного и открытого шифрования.

Литература

1. *Ko K. H., Lee S. J., Cheon J. H., Han J. W., Kang J. S., Park C.* New Public Key Cryptosystems Using Braid Groups // Advances in Cryptology — Crypto 2000 / Lecture Notes in Computer Science. Springer-Verlag, 2000. Vol. 1880. P. 166–183.

2. *Молдовян Д. Н.* Конечные некоммутативные группы как примитив криптосистем с открытым ключом // Информатизация и связь. 2010. № 1. С. 61–65.

3. *Молдовян Н. А.* Алгоритмы аутентификации информации в АСУ на основе структур в конечных векторных пространствах // Автоматика и телемеханика. 2008. № 12. С. 163–177.

ПРИНЦИПЫ СОЗДАНИЯ МОДЕЛЕЙ ОЦЕНКИ СИСТЕМ ЗАЩИТЫ ИНФОРМАЦИИ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Я. М. Гвоздик
ООО «Газинформсервис»

А. А. Молдовян
СПИИРАН

После принятия в Российской Федерации ГОСТ Р ИСО/МЭК ТО 19791-2008 [1] актуальным стала задача создания моделей оценки систем защиты информации автоматизированных систем (СЗИ АС), позволяющих получать объективные и корректные результаты. Данный государственный стандарт принят в развитие положений руководящего документа ФСТЭК (Гостехкомиссии) России «Безопасность информационных технологий…» [2] в части оценки регуляторов безопасности организационного и процедурного уровней. Перечисленные документы являются аналогами международных стандартов ISO/IEC 15408 и ISO/IEC TR 19791.

Сформированная, на основе перечисленных документов, система критериев оценки СЗИ АС имеет иерархическую структуру, состоящую из пяти уровней (рис. 1).

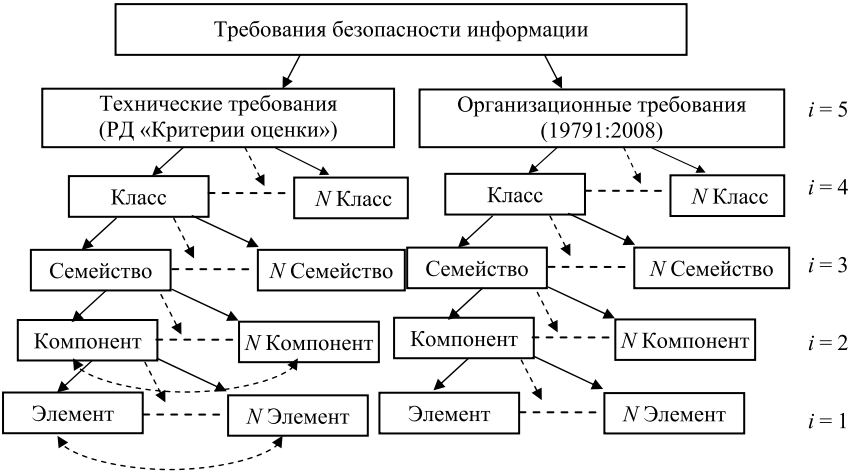


Рис. 1. Структура критериев оценки СЗИ АС

В качестве методологической основы оценки СЗИ АС можно предложить метод сводных показателей, который позволяет агрегировать оценки, проводимые на каждом уровне иерархии критериев, в обобщенную оценку.

При этом, должны выполняться ряд требований к исходным характеристикам, показателям и виду агрегирующих функций [3]:

- множество исходных характеристик должно быть минимально достаточным для оценивания СЗИ АС и измеряться по числовой шкале или шкале линейного порядка;
- показатели должны быть поляризованы, а агрегирующая функция должна учитывать важность каждого входящего показателя.

Исходя из общей структуры критериев и особенностей методов оценки критериев на каждом уровне, можно предложить следующие принципы создания модели многоуровневой оценки СЗИ АС.

Принцип 1 — первичность структуры критериев перед видом методов оценки. Суть принципа заключается в том, что модель оценки СЗИ АС создается как множество взаимоувязанных, согласованных на выбранной иерархической структуре критериев, частных методов оценки и в полной мере должна учитывать особенности оценки на каждом уровне.

Принцип 2 — согласованность входов вышестоящего уровня выходам нижестоящих уровней по типу передаваемых значений величин и их градации.

Принцип 3 — соответствие количества градаций оцениваемых величин уровню «уверенной» оценки эксперта.

Принцип 4 — рациональный выбор частного метода оценки в соответствии с градациями и инциденциями конкретного критерия.

В соответствии с предложенными принципами, модель многоуровневой оценки СЗИ АС состоит из четырёх взаимоувязанных процедур:

1. Процедура нечеткого экспертного оценивания элемента [4] (1-й уровень иерархии).
2. Процедура нечеткого продукционного оценивания показателей на основе оценок элементов (2-й уровень иерархии).
3. Процедура комплексного оценивания (3-й и 4-й уровни иерархии).
4. Процедура получения общей оценки на основе комплексного оценивания с использованием упрощенного метода анализа иерархий (5-й уровень иерархии).

Для создания процедуры нечеткого экспертного оценивания элементов необходимо решить ряд задач:

- выбрать шкалу, определить множество уровней шкалы и количественных значений проявлений признаков в рамках этих уровней;
- определить метод получения функции принадлежности;
- определить механизм получения значений проявлений признаков.

Для оценки степени выполнения требований безопасности на 1-ом уровне иерархии (элемент) в работе используются порядковые шкалы, элементы которых соответствуют вербальным уровням лингвистических шкал. При этом, оперировать приходится не со значениями несопоставимых между

собой, оцененных в разных шкалах и имеющих разные размерности признаков, а с безразмерными величинами — значениями функций принадлежности, что позволяет в процессе оценки корректно их обрабатывать.

Для экспертного оценивания степени выполнения требований безопасности необходимо определить множество вербальных уровней шкал.

В соответствии с выбранной шкалой экспертами оценивается степень выполнения требований безопасности на 1-ом уровне иерархии критериев оценки (элемент) с последующим построением соответствующих функций принадлежности.

Анализ методов формализации элементов шкал, применяемых для оценивания систем защиты информации АС, и соответственно метод формализации нечетких данных, полученных при оценивании этих характеристик экспертами, показал, что в большинстве случаев нам подходит следующий метод построения функции принадлежности.

Пусть имеются данные, полученные в результате оценивания у объектов качественной характеристики X в рамках вербальной шкалы с уровнями $X_l, l = \overline{1, m},\ m \geq 2$. Упорядочим их по возрастанию интенсивности проявления. Обозначим относительные частоты появления объектов, у которых интенсивность проявления X оценена уровнями $X_l, l = \overline{1, m},$ соответственно через $a_l, l = \overline{1, m},\ \sum_{l=1}^{m} a_l = 1$. Обозначим $\min(a_1, a_2)$ через b_1, $\min(a_{l-1}, a_l, a_{l+1})$, $l = \overline{2, m-2}$ через $b_l,\ l = \overline{2, m-2},$ а $\min(a_{m-1}, a_m)$ через b_{m-1}.

Тогда

$$\mu_1(x) \equiv \left(0,\ a_1 - \frac{b_1}{2},\ 0,\ b_1\right),$$

$$\mu_l(x) \equiv \left(\sum_{i=1}^{l-1} a_i + \frac{b_{l-1}}{2},\ \sum_{i=1}^{l} a_i + \frac{b_l}{2},\ b_{l-1},\ b_l\right),\quad l = \overline{2, m-2},$$

$$\mu_{m-1}(x) \equiv \left(\sum_{i=1}^{m-2} a_i + \frac{b_{m-2}}{2},\ 1 - a_m - \frac{b_{m-1}}{2},\ b_{m-2},\ b_{m-1}\right),$$

$$\mu_m(x) \equiv \left(1 - a_m - \frac{b_{m-1}}{2},\ 1 - a_m + \frac{b_{m-1}}{2} b_{m-1},\ 0\right).$$

Данный метод работает в условиях неполной информации, инвариантен относительно последовательности построения функций принадлежности элементов шкал, используемых для оценивания качественных характеристик. Построением функций принадлежности завершается первый этап оценки.

При практической СЗИ АС экспертами осуществляется лингвистическая оценка выполнения требований безопасности на уровне элементов, тем самым завершается выполнение первой процедуры.

Для реализации процедуры нечеткого продукционного оценивания показателей предлагается использовать алгоритм нечеткого логического вывода, позволяющий формировать адекватные оценочные модели на основе нечеткого логического вывода по Мамдани. Выбор модели нечеткого логического вывода обосновывается его большой распространенностью в практических приложениях, прозрачностью самого вывода и легкостью настроек параметров.

Нечеткий вывод в своей основе имеет базу знаний, формируемую экспертами в виде совокупности нечетких предикатных правил:

$$\Pi_i\text{: если } x \text{ есть } A_i\text{, тогда } z \text{ есть } B_i,$$

где x — входная переменная, z — переменная вывода, A_i и B_i — нечеткие множества, определенные соответственно на X и Z с помощью функций принадлежности $\mu_{A_i}(x)$ и $\mu_{B_i}(z)$, $i = 1, 2, \ldots, n$.

Указанный вывод в форме алгоритма Мамдани математически описывается следующим образом:

1. Введение нечеткости (fuzzification) — для заданного (четкого) значения аргумента $x = x_0$ находятся степени истинности для предпосылок каждого правила $a_i = \mu_{A_i}(x_0)$.
2. Нечеткий вывод по каждому правилу — находятся «усеченные» функции принадлежности для переменной вывода:

$$\mu^*_{B_i}(z) = \min_z (a_i, \mu_{B_i}(z)).$$

3. Композиция — с использование операции максимум (max) производится объединение найденных усеченных функций, что приводит к получению итогового нечеткого подмножества для переменной вывода с функцией принадлежности

$$\mu_\Sigma(z) = \mu_B(z) = \max_z [\mu^*_{B_1}(z), \mu^*_{B_2}(z), ..., \mu^*_{B_o}(z)].$$

4. Приведение к четкости (defuzzification) — для нахождения $z_0 = F(x_0)$ — обычно проводится центроидным методом — четкое значение выходной переменной определяется как центр тяжести для кривой $\mu_\Sigma(z)$, т. е.

$$z_0 = \frac{\int\limits_\Omega z \cdot \mu_\Sigma(z)\, dz}{\int\limits_\Omega \mu_\Sigma(z)\, dz},$$

где Ω — область определения $\mu_\Sigma(z)$.

Начиная с 3-го уровня иерархии (семейств) и выше, в работе предлагается оценку СЗИ АС осуществлять с использованием в качестве агрегирующих сверток — матричные свертки.

Особенностью матричных сверток является то, что любая произвольная свертка двух критериев на дискретном множестве может быть представлена в матричном виде, что позволяет тонко учесть знания эксперта.

Общая структура агрегирования отдельных критериев в комплексную оценку имеет вид дихотомического дерева, узлами которого являются матрицы свертки.

Построение дихотомической структуры производится экспертами исходя из следующих соображений. Определяется m_i — общее количество критериев нижнего уровня, сворачиваемых в данном критерии.

Для получения общей оценки СЗИ АС (5-й уровень иерархии) используется процедура упрощенного метода анализа иерархий (МАИ). Выбор данного метода для оценки обосновывается рядом факторов:

1. Обобщенная оценка включает в себя свертку более десятка классов. Построение эквивалентного бинарного дерева для матричной свертки повлечет за собой построение десятка промежуточных матриц свертки, большая часть которых будет лишена реального «физического» смысла, что приведет к необоснованной погрешности оценки.
2. Оценка взаимовлияния классов друг на друга требует от экспертов высокой квалификации, так как ошибки в оценках на этом уровне не могут быть компенсированы в процессе дальнейшей работы.
3. Привлечение экспертов высокой квалификации для построения большого количества бинарных матриц свертки не увеличивает точность оценки, по сравнению с простым ранжированием классов по их важности, что требуется для упрощенного метода анализа иерархий.
4. Использование известного метода анализа иерархий требует построения матрицы парных сравнений, что практически не реально для матриц размерностью больше семи (7×7), не говоря о матрицах (15×15).

Суть упрощенного МАИ [5] заключается в том, что для вычисления весов линейной аддитивной свертки обобщенной оценки используем только первую строку элементов a_{ij} матрицы парных сравнений, что в десятки раз проще в нашем случае. Далее, компоненты вектора весов $w = (w_1, w_2, \ldots, w_n)^T$ вычисляются по формуле

$$w_i = \frac{a_{1n}}{a_{1i}}, \quad i = 1, 2, ..., n,$$

что существенно упрощает промежуточные расчеты и не вносит дополнительную погрешность.

Предложенная модель оценки СЗИ АС разработана с использованием современного математического аппарата и позволяет повысить объективность и корректность оценки за счёт использования методов обработки трудно формализуемых данных предметной области.

Литература

1. ГОСТ Р ИСО/МЭК ТО 19791-2008 «Информационная технология. Методы и средства обеспечения безопасности. Оценка безопасности автоматизированных систем».

2. Руководящий документ ФСТЭК (Гостехкомиссии) России. «Безопасность информационных технологий. Критерии оценки безопасности информационных технологий».

3. *Хованов Н. В., Федотов Ю. В.* Модели учета неопределенности при построении сводных показателей эффективности деятельности сложных производственных систем // Научные доклады НИИ менеджмента СПбГУ. 2006. № 28. С. 1–37.

4. *Заде Л.* Понятие лингвистической переменной и его применение к принятию приближенных решений. М.: Мир, 1976. 165 с.

5. *Ногин В. Д.* Принятие решений при многих критериях (учебно-методическое пособие). СПб.: Ютас, 2007. 104 с.

ОБОСНОВАНИЕ НЕОБХОДИМОСТИ СОБЛЮДЕНИЯ КОНФИДЕНЦИАЛЬНОСТИ ВИДЕОДАННЫХ

Р. Ш. Фахрутдинов

СПИИРАН (Санкт-Петербург, Россия)

С широким распространением цифрового медиаконтента, остро встал вопрос его защиты от хищения, искажения и незаконного копирования. Видеоконференции, видеосвязь, видеонаблюдение, цифровое наземное, кабельное и спутниковое ТВ — вот далеко неполный перечень актуальных приложений, которые не могут быть конфиденциальны без применения в том или ином виде защиты в связи с тем, что большинство такого рода приложений использует сети общего назначения (например, Интернет или радиоканал в случае эфирного цифрового телевидения) для передачи данных. Необходимость достаточно надёжного закрытия данных диктуется и всё более производительными ПК, которые доступны злоумышленникам.

Особо нужно сказать об актуальности применении шифрования видеоинформации в военной технике. По сообщению газеты «The Wall Street Journal» [1], противостоящие силам НАТО в Ираке и Афганистане боевики использовали простую программу SkyGrabber для перехвата видеоинформации разведывательно-ударных беспилотных летательных аппаратов (БПЛА) MQ-1 Predator.

1. Селективное шифрование видеоданных

Существует два основных подхода к защите видеоинформации — полное и селективное шифрование видеопотока. Полное шифрование подразумевает предоставление для передачи канала данных, закрытого каким-либо криптографическим алгоритмом, либо криптографического закрытия непосредственно внутри транспортного протокола. При этом производится шифрование всего потока данных, в том числе и служебной информации. В отличии от полного, селективное шифрование обычно внедряется в сам алгоритм сжатия информации. Используется тот факт, что информационная плотность данных неоднородна, поэтому для достижения необходимого уровня безопасности данных, шифровать можно не всё.

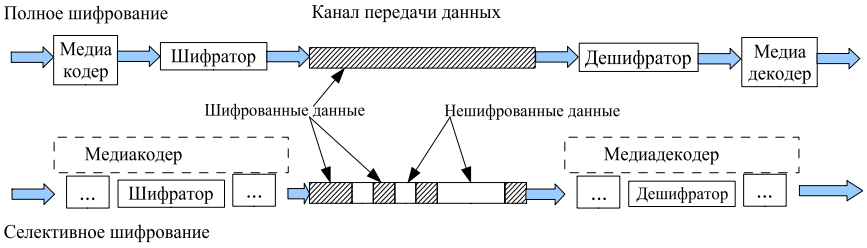


Рис. 1. Формирование шифртекста для передачи по каналу связи

При полном шифровании вначале выполняется сжатие (упаковка) входных данных с помощью медиакодера, после которого сжатые данные попадают на вход шифратора. После шифрования данные поступают на вход канала передачи данных.

Селективное шифрование производится непосредственно в медиакодере [2]. В этом процессе определяются данные, подлежащие шифрованию, в результате чего объём шифрованных данных оказывается меньше общего объёма данных. При декодировании производится обратный процесс, встроенный в декодер дешифратор выполняет выборочное расшифрование данных и на выходе получаем открытую последовательность видеоданных.

Основным достоинством селективного шифрования является то, что его можно с успехом использовать там, где есть готовая инфраструктура передачи данных, не адаптированная для полного шифрования и где такая адаптация не является экономически обоснованной, для применения в устройствах, которые не имеют встроенных аппаратных средств для полного шифрования или имеют недостаточную для этого производительность центрального процессора (например, КПК или мобильные телефоны). Основным недостатком селективного шифрования является более низкая, чем у полного шифрования стойкость. При этом сниженные требования по стойкости делают возможным разработки алгоритмов селективного шифрования, обеспечивающих высокую производительность. Существенным плюсом селективного шифрования (в большинстве реализаций) является его устойчивость к ошибкам передачи, так как транспортный поток не шифруется, не происходит (как в случае с блочным шифрованием) распределения ошибочного бита на целый блок данных, сохраняются механизмы защиты от искажения при передаче по каналам связи, встроенные в алгоритм шифрования и протокол передачи.

К другим минусами селективного шифрования является возможное снижение эффективности сжатия и зависимость реализации от конкретного медиакодека.

2. Схема алгоритма перестановок

В качестве примера селективного шифрования видеоданных приведём схему селективного шифрования данных с помощью перестановки блоков в процессе сжатия (например, DCT-блоков в алгоритмах сжатия MPEG2/MPEG4 или DWT-блоков в перспективных методах сжатия видео [3]), или до начала работы кодека.

В качестве входных данных используется набор несжатых видеокадров, на выходе кодека — поток сжатых видеокадров в специальном формате, который зависит от настроек кодека и особенностей его реализации. Перестановку блоков можно выполнить как на входе кодека (I), так и после квантования макроблоков (II).

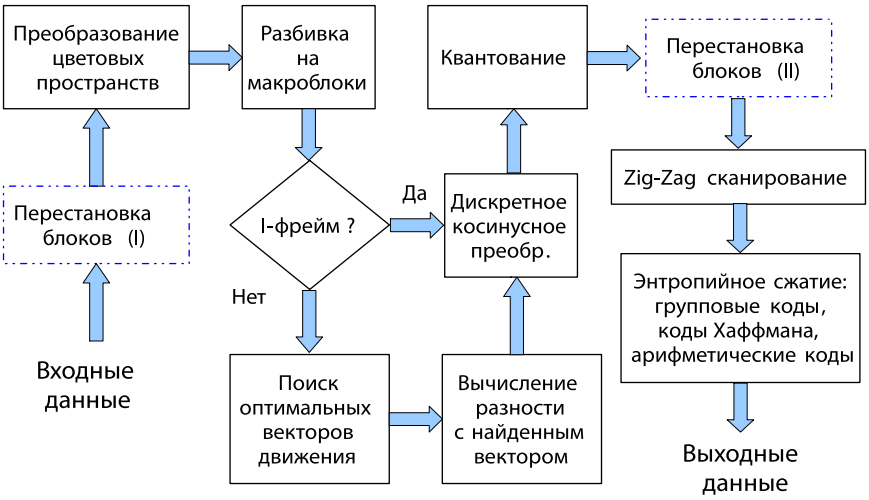


Рис. 2. Селективное шифрование с помощью перестановки блоков

Литература

1. *Siobhan Gorman, Yochi J. Dreazen, August Cole*. Insurgents Hack U.S. Drones, $26 Software Is Used to Breach Key Weapons in Iraq; Iranian Backing Suspected // The Wall Street Journal. December 17, 2009. http://online.wsj.com/article/SB126102247889095011.html

2. *Р. Ш. Фахрутдинов, Н. А. Молдовян.* Схема открытого распределения ключей и алгоритмы шифрования видеоданных // Вопросы защиты информации. 2010. № 1. С. 14–23.

3. *Ватолин Д., Ратушняк А., Смирнов М., Юкин В.* Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. М.: Диалог-МИФИ, 2003.

СИСТЕМНЫЙ ПОДХОД ПРИ РАЗРАБОТКЕ СЗИ

Р. Ш. Фахрутдинов, А. И. Галанов, А. Ю. Мирин, Д. К. Сухов

СПИИРАН (Санкт-Петербург, Россия)

Главный принцип системного подхода к построению средств защиты информации [1] — учет всех исходных требований, существующих угроз и влияющих на безопасность факторов при комплексном использовании наиболее эффективных мер, методов и средств защиты. Выделяются четыре аспекта рассматриваемого подхода:

- учет основополагающих требований, вытекающих из теории и практики защиты информации;
- тщательное и полное выполнение необходимых стадий разработки систем защиты с контролем качества промежуточных и итоговых результатов;
- решение всех базовых задач защиты информации;
- обеспечение гарантированности защиты за счет использования проверенных методов и алгоритмов.

К СЗИ изначально должны предъявляться следующие основополагающие требования, обеспечивающие максимальную степень защищенности автоматизированных систем:

- соответствие отечественным нормативным документам (в т. ч. требованиям ФСТЭК, ФСБ, МО) и международным стандартам (ISO 17799 — практические правила обеспечения безопасности, ISO 15408 — общие критерии оценки безопасности);
- многоуровневое построение системы защиты при корректности, полноте и непротиворечивости реализации всех приоритетных функций;
- централизованное управление средствами защиты, пользователями, рабочими станциями и ресурсами компьютерной сети на основе правил единой политики безопасности;
- централизованный контроль защищенности и поддержка принятия решений для снижения количества ошибок администрирования и своевременного реагирования на события, связанные с нарушениями информационной безопасности.

Анализ рисков

Процесс анализа рисков включает в себя:

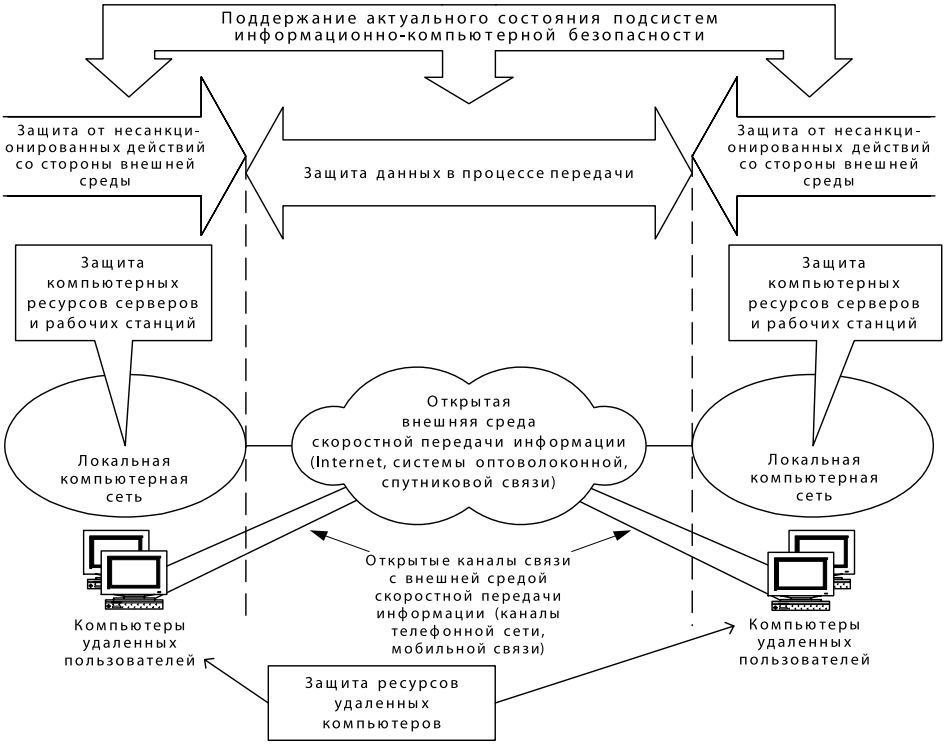
1. *Формирование модели защищаемых ресурсов* — идентификация и анализ защищаемых ресурсов, в качестве которых рассматривается все, что представляет ценность в АС.

2. *Формирование модели угроз* — составление полного списка угроз защищаемым ресурсам в АС.
3. *Формирование модели уязвимостей и модели нарушителя* — выявление уязвимостей и потенциальных нарушителей, которые делают возможной реализацию угроз.
4. *Формирование модели контрмер* — выработка контрмер, перекрывающих выявленные угрозы и предпринимаемых для уменьшения уязвимостей, используемых при реализации этих угроз.

Средства защиты при формировании контрмер должны обеспечивать решение базовых задач информационно-компьютерной безопасности:

1) защиты компьютерных ресурсов на уровне серверов и рабочих станций ЛВС;
2) защиты от несанкционированного межсетевого взаимодействия;
3) защиты передаваемой информации;
4) поддержания системы защиты в актуальном состоянии.

Базовые задачи информационно-компьютерной безопасности



Архитектура СЗИ «Аура»

В состав СЗИ [2], в соответствии с результатами анализа угроз, уязвимостей и атак, включены пять целевых подсистем и одна обеспечивающая подсистема.

К целевым подсистемам относятся:

1. Подсистема управления доступом к защищаемым ресурсам — предназначена для разграничения доступа пользователей к информации и ассоциированным с ней компьютерным ресурсам.
2. Подсистема регистрации и учета — предназначена для регулярного сбора, фиксации и выдачи по запросам сведений обо всех обращениях к защищаемым компьютерным ресурсам, а также доступе в компьютерную систему и выходе из нее.
3. Подсистема гарантированного уничтожения информации — предназначена для предотвращения доступа к остаточной информации в участках освобождаемой памяти, а также гарантированное удаление остаточной информации.
4. Подсистема обеспечения целостности — предназначена для обеспечения целостности средств защиты, программной среды, а также файлов и каталогов пользователей.
5. Подсистема администрирования — предназначена для управления подсистемами защиты в составе СЗИ, управления субъектами и объектами доступа, а также поддержания системы защиты в актуальном состоянии.

Подсистема глобального кодирования информации на носителях и виртуальных дисках является обеспечивающей подсистемой и предназначена для повышения общего уровня безопасности и эффективной реализации возможностей других подсистем, входящих в состав СЗИ.

Уровни усиления защиты ОС

1. *Уровень усиленной аутентификации.* Данный уровень обеспечивает идентификацию и эффективную проверку подлинности всех пользователей, допущенных в систему. Реализуется модулем аутентификации и нейтрализует угрозу несанкционированного входа в систему.
2. *Уровень контроля целостности программной среды.* Данный уровень обеспечивает работу в доверенной среде, исключающей наличие несанкционированных программ. Реализуется модулем контроля целостности и нейтрализует угрозу внедрения программных закладок.
3. *Уровень защиты от бесконтрольного доступа в процессе сеанса работы пользователя.* Данный уровень обеспечивает блокировку по тайм-ауту при отсутствии признаков активности пользователя или принудительную блокировку устройств ввода-вывода (клавиатуры, мыши и монитора) по электронному ключу или комбинацией клавиатуры. Реализуется

драйвером блокировки консоли и нейтрализует угрозу бесконтрольного несанкционированного доступа к информационным ресурсам при оставлении компьютера без присмотра в процессе сеанса работы пользователя.

4. *Уровень защиты ввода и вывода на отчуждаемый физический носитель информации.* Данный уровень препятствует бесконтрольному вводу информации со съемных носителей, а также бесконтрольному выводу информации на эти носители за счет криптографической привязки съемных носителей к защищаемому компьютеру (за счет прозрачного кодирования информации на съемных носителях по уникальному ключу защищаемого компьютера).
5. *Уровень защиты от несанкционированной загрузки ОС со съемных носителей.* Данный уровень обеспечивает недоступность информационных ресурсов и доверенной программной среды после несанкционированной загрузки ОС со съемных носителей. Реализуется драйвером прозрачного кодирования и нейтрализует угрозы обхода модулей защиты путем несанкционированной загрузки ОС со съемных носителей.
6. *Уровень криптографического закрытия защищаемых данных.* Данный уровень обеспечивает криптографическое закрытие защищаемой информации путем ее прозрачного кодирования. Реализуется драйверами прозрачного кодирования дисков и нейтрализует угрозы хищения носителей информации и обхода модулей защиты путем несанкционированной загрузки ОС со съемных носителей.
7. *Уровень защиты от доступа к остаточной информации.* Данный уровень обеспечивает гарантированное удаление информации в освобождаемых участках памяти. Реализуется модулем гарантированного уничтожения информации и нейтрализует угрозы хищения информации путем доступа к участкам освобожденной памяти.
8. *Уровень защиты от несанкционированных действий санкционированных пользователей.* Данный уровень обеспечивает реализацию принципа подотчетности путем установления соответствия между всеми входами в систему и реальными пользователями, а также учетом каждого действия и сопоставления каждого действия конкретному пользователю. Реализуется модулем регистрации и нейтрализует угрозы несвоевременного обнаружения несанкционированных действий санкционированных пользователей.

Литература

1. *Молдовян А. А., Юсупов Р. М.* Проблемы информатизации и вопросы информационной безопасности транспортной отрасли // Транспортная безопасность и технологии. 2010. № 3.

2. *Баранюк Т. Н., Заболотный А. П., Мирин А. Ю., Фахрутдинов Р. Ш.* Перспективная Система защиты информации от несанкционированного доступа «Аура» // Инновационная деятельность в Вооруженных силах Российской Федерации: Труды всеармейской научно-практической конференции. 10–11 декабря 2009, г. Санкт-Петербург. СПб.: ВАС, 2009. С. 60–62.

СОРЕВОВАНИЯ ПО КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ (CTF)

Н. Н. Журавлев (e-mail: znick@hackerdom.ru),
И. В. Зеленчук (e-mail: ilya@hackerdom.ru)

Уральский государственный университет им. А. М. Горького

Развитие глобальной сети Интернет повысило значимость специалистов в области информационной безопасности. Термин CTF [1] (Capture the flag, захват флага) означает командные соревнования, целью которых является развитие у участников навыков защищать компьютерные системы. Игра моделирует реальную компьютерную сеть находящейся под постоянной атакой, чтобы посмотреть, как участники игры справятся с этой ситуацией.

На соревновании командам выдается игровой образ, который представляет собой виртуальный сервер с несколькими игровыми сервисами. Каждый такой сервис представляет из себя сетевое приложение с определенным функционалом. Минимальный набор требований к игровому сервису:

- доступ по сети;
- возможность создания новых пользователей;
- аутентификация пользователей;
- каждый пользователь должен иметь возможность сохранить приватную, для него, информацию с последующим её просмотром.

Все игровые сервисы разрабатываются специально для каждого соревнования. Во время реализации сервиса разработчики встраивают в него несколько уязвимостей.

На протяжении игрового времени командам необходимо поддерживать свои сервисы в рабочем состоянии, обнаружить и устранить уязвимости не нарушив работоспособность сервисов. В то же время, используя знания о найденных уязвимостях, становится возможным использовать сервера других команд для обхода механизма безопасности сервиса и «захватить флаг» (некоторую приватную информацию). Баллы команде начисляются за:

- поддержание игрового сервиса в рабочем состоянии;
- отправку флагов, захваченных с сервисов команд-соперниц;
- отправку описания найденных уязвимостей с инструкцией по их устранению;
- решение дополнительных заданий.

Подобные соревнования набирают популярность в мире. Большинство организаторов и участников представители различных университетов. Наиболее популярными международными соревнованиями считаются iCTF [2] (Университет Санта-Барбары, Калифорния), RuCTFE [3] (Уральский государственный университет им. А.М.Горького) и C.I.P.H.E.R.[4] (Ахенский Технический Университет, Германия).

В России соревнования по защите информации становятся популярными именно в университетской среде. За последние два года прошли 5 соревнований:

- RuCTF — Уральский государственный университет им. А.М,Горького (УрГУ);
- UralCTF — Челябинский государственный университет;
- UfoCTF — Таганрогский Технологический Институт Южного Федерального Университета;
- LeetMore CTF — Национальный исследовательский университет информационных технологий, механики и оптики (СПБГУ ИТМО);
- SchoolCTF — Томский государственный университет (ТГУ);
- РусКрипто CTF — Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича.

Начиная с 2006 года на международных соревнованиях стали появляться Российские команды. Можно отметить следующие результаты:

- в 2007 году на iCTF команда HackerDom из УрГУ заняла 3-е место;
- в 2008 году на C.I.P.H.E.R. команды HackerDom из УрГУ заняла 1-е место;
- в 2008 году на iCTF команды SiBEars из ТГУ заняла 2-е место;
- в 2009 году на C.I.P.H.E.R. команды HackerDom из УрГУ заняла 2-е место;
- в 2010 году команда LeetMore из СПБГУ ИТМО прошла в финал крупных международных соревнований CODEGATE [5].

Для организации CTF соревнований требуются большие человеческие и технические ресурсы. В зависимости от продолжительности срока подготовки игрового образа и масштаба соревнований размер команды разработчиков может колебаться от 8 до 20 человек. При этом, команда в основном состоит из студентов, магистрантов, аспирантов и молодых преподавателей.

В команде разработчиков, как правило, выделяются следующие роли:

- менеджер проекта (тимлид) — несет ответственность за планирование, подготовку и реализацию игрового образа. Также занимается организацией команды;
- администратор игровой инфраструктуры — планирует и разворачивает игровую сеть;
- администратор проверяющей системы — отвечает за настройку, запуск и мониторинг проверяющей системы;
- администратор игрового образа — собирает игровой образ и убеждается в его корректной работе;
- разработчик сервиса — разрабатывает и реализовывает игровой сервис, пишет документацию и тесты.

CTF позволяет студентам, обучающихся компьютерным наукам, в игровой форме изучить новые технологии и повысить уровень своих навыков. На соревновании участники получают практический опыт в обнаружении и устранении уязвимостей в программном обеспечении. Подготовка к сорев-

нованиям способствует к занятию различными исследованиями и разработкой собственных программных продуктов. Немаловажным положительным эффектом является обучение студента работать в команде.

Литература

1. http://en.wikipedia.org/wiki/Capture_The_Flag#Computer_security
2. http://ictf.cs.ucsb.edu/
3. http://ructf.org/e/2010/
4. http://www.cipher-ctf.org
5. http://www.codegate.org

МЕТОД ПОИСКА ВРЕДОНОСНЫХ ПРОГРАММ НА ОСНОВЕ АНАЛИЗА ПРОЦЕССА РАСПРОСТРАНЕНИЯ

Р. Василенко

Абстракт. Количество вредоносных программ увеличивается с такой скоростью, что бороться с ними с помощью алгоритмов использующих антивирусные базы на стороне пользователя становится проблематично. Вирусописатели хорошо знакомы с общей схемой работы антивирусных компаний по выпуску новых детектирующих записей:

1. Поступление образца в антивирусную лабораторию.
2. Анализ образца (ручной или, что чаще, автоматический).
3. Создание детектирующей записи (эвристической или по бинарным маскам).
4. Тестирование детектирующей записи.
5. Выпуск баз обновлений.

Все данные этапы в разных антивирусных компаниях имеют свои особенности, но есть один общий факт — каждый этап занимает определенное время. В сумме с момента попадания образца в антивирусную лабораторию до выхода обновления проходит время t, которое обычно не меньше двух часов.

Зная это, вирусописатели оптимизируют свои алгоритмы выпуска вредоносных программ таким образом, чтобы максимально понизить эффективность выпущенных записей, что в худшем случае приводит к тому, что выпущенная антивирусной компанией запись «мертва», т. е. вредоносного образца, который она детектирует, уже нет у пользователей.

Общая модель

Из литературы [1–4] известно о некоторых подходах к решению обозначенной проблемы. Основная идея представленных систем в том, что используются статистические данные, полученные из множества различных источников, включая: анонимную информацию, предоставленную десятками миллионов пользователей, данные издателей ПО, а также анонимные данные от клиентов крупных предприятий. Все эти данные попадают на обработку в специальную систему, которая определяет «рейтинг опасности».

Обычно в таких системах используется такая информация, как: распространенность, время существования, и т. д. Однако в предложенных алгоритмах анализируются лишь количественные показатели, на определенный момент времени.

Предлагаемый мной подход отличается главным образом тем, что мы рассматриваем распространение вредоносной программы как развитие системы, таким образом, анализируя не только и не столько количественные показатели на определенный момент времени, сколько качественные изменения данных показателей во времени.

Развитие системы — это всегда развитие во времени, зависящее от многих факторов, однако, несмотря на их обилие, различные системы с разной степенью допущений и успешности поддаются моделированию.

Одним из законов Теории Развития Систем является закон, согласно которому все системы развиваются по *S*-образному графику.

По вертикали откладываются основные параметры системы, по которым можно оценивать ее развитость, эффективность и т. п. По горизонтали — время. График напоминает латинскую букву *S*, отсюда и название «*S*-образная кривая».

На реальных данных график выглядит намного менее гладко, но общая структура остается схожей и можно выделить четыре этапа:

I. Система начинает свою жизнь с медленного развития.
II. Система преодолела сложности первого этапа и начинается ее бурный рост.
II. Система находится на пике своего развития по основным параметрам, но при этом замедляет, а затем и вовсе прекращает свой рост.
IV. Старость, угасание, смерть. Значение основных параметров уменьшается, система перестает соответствовать требованиям современности.

Распространение нового файла по компьютерам пользователей так же можно считать некой системой изменяющейся во времени. При этом «Время» — будет время распространения файла по все большему числу компьютеров пользователей, а в качестве «Параметров» будет выступать общее количество экземпляров данного файла.

Анализ системы развития

На этапе построения модели мы имеем только набор статистических данных за определенный промежуток времени. Так же выдвигается гипотеза, основанная на предположении, что распространение вредоносных и чистых файлов, хоть и проходит указанные выше этапы, но обладает качественно разными показателями распространения.

Гипотеза

Способы распространения вредоносных и легальных программ отличаются друг от друга.

Встает вопрос о том, что мы не знаем закона развития нашей системы. Определение функции развития, основанное на предположениях, было бы не верно, т. к. на разных этапах жизни системы эти функции разные, да и определение это ставит нас в рамки этой функции, что при ошибочном предположении пагубно влияет на все остальные расчеты. Именно поэтому было принято решение использовать не идеальное приближение к неизвест-

ной функции, но рассматривать процесс распространения программы как некую разрывную линейно-кусочную функцию, которая на фиксированном участке анализа заменяется линейным приближением.

Алгоритм:

1. Выделяется общий временной интервал, на котором будет проводиться исследование.
2. Интервал делится на k одинаковых частей (рис. 1).
3. Определяется значение $f(n)$ (количество экземпляров файла) на границе каждого интервала.
4. Значения $f(n)$ соединяются прямыми.

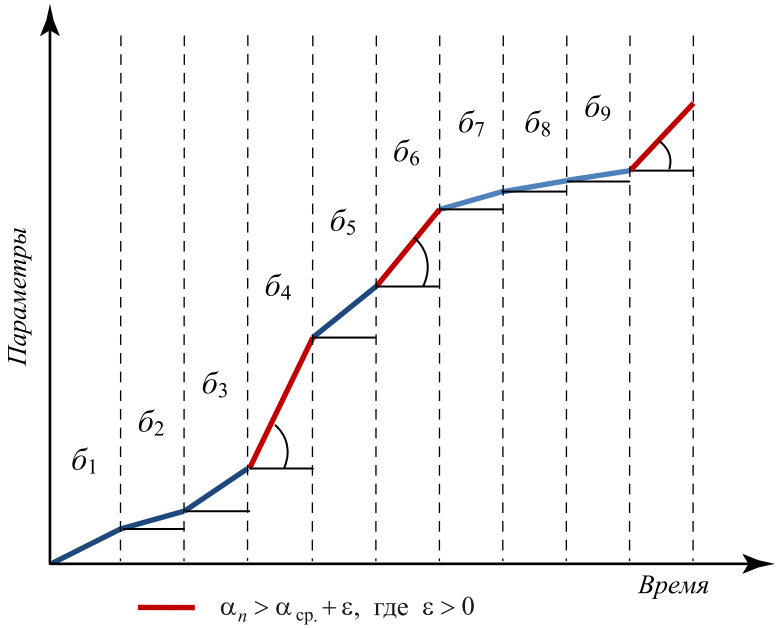


Рис. 1. График системы развития программы

Таким образом, угол $б_{i\,на}$ каждом интервале будет определять приближенное значение относительной скорости роста на данном интервале. Понятно, что чем меньше выбранный интервал, тем точнее приближение.

Определяется относительное изменение величины угла $(б_{i+1} - б_i)$ и относительный знак этого изменения при сравнении двух ближайших участков $(б_{i+1} - б_i) - (б_{i+2} - б_{i+1})$. В соответствии с этими величинами будет определять «ускорение» роста — вторая производная и ее знак (увеличение/уменьшение скорости изменения относительной скорости роста).

По данным параметрам можно набрать избыточную базу знаний, например в виде линейных матриц, как о процессах распространения заведомо легальных программ, так и о процессах распространения вредоносных программ. Сам процесс распространения можно выразить в виде паттерна, матричные элементы которого имеют определенные доверительные диапазоны, позволяющие проводить сравнение матриц, вычисляя статистически достоверную корреляцию подобия, разделяя процессы-паттерны на «безопасные», «подозрительные» и «опасные».

В ходе создания базы знаний, а так же проведения испытаний, признаки подобия процессов-паттернов могут уточняться введением дополнительных корреляций между первичными параметрами.

Заключение

Данный подход позволяет минимизировать вычислительные ресурсы под конкретную ресурсную ситуацию, либо важность задачи. Экономии вычислительных ресурсов может способствовать еще один важный фактор. В различный период времени значимой может становиться только определенная группа параметров, либо группа определенных корреляций между ними. По мере изменения «прогнозной ценности успешных групп», их можно менять на другие, с лучшими текущими показателями.

Литература

1. *Nachenberg C., Ramzan Z., Seshadri V.* Reputation: a new chapter in malware protection // Virus Bulletin Conference. 2009. Pp. 185–191.

2. US Patent No. 7,640,589 B1 Detection and minimization of false positives in anti-malware processing / Mashevsky Y. V., Namestnikov Y. V., Denishchenko N. V., Zelensky P. A., Chekunov I. G., Efremov A. A.; Заявлено 19.06.2009; Опубл. 29.12.2009.

3. US Patent No. 7,743,419 B1 Method and system for detection and prediction of computer virus-related epidemics / Mashevsky Y. V., Namestnikov Y. V., Denishchenko N. V., Zelensky P. A.; Заявлено 06.12.2009; Опубл. 22.06.2010.

4. European patent app. No. 07015353.1 A system that provides early detection, alert, and response to electronic threats / Elovici Y. M., Tachan G. O., Shabtai A. C.; Заявлено 06.08.2007; Опубл. 13.02.2008; Бюл. № 2008/07.

СОДЕРЖАНИЕ

Научное издание

СПИСОК-2011

Материалы
второй межвузовской научной конференции
по проблемам информатики

27–29 апреля 2011 г.,
Санкт-Петербург

Компьютерная верстка: *О. В. Шакиров*

Издательство «ВВМ»
190000, Санкт-Петербург,
ул. Декабристов, 6, лит. А, пом. 10н
E-mail: vvmpub@yandex.ru

Подписано к печати 10.10.11. Формат 60 × 90 1/16. Бумага офсетная.
Гарнитура Таймс. Печать цифровая. Печ. л. 29,75. Тираж 100 экз.
Заказ

Отпечатано в Отделе оперативной полиграфии
химического факультета СПбГУ
198504, Санкт-Петербург, Старый Петергоф, Университетский пр., 26
Тел.: (812) 428-4043, 428-6919