ИСП Институт Системного Программирования
Российской Академии наук

Труды Института Системного Программирования

Том 19

Москва 2010

ИСП **Учреждение Российской академии наук**
Институт системного программирования РАН

Труды Института Системного Программирования

Том 19

Под редакцией
академика РАН В.П. Иванникова

Москва 2010

УДК51

Труды Института системного программирования: Том 19.
/Под ред. В.П. Иванникова/ – М.: ИСП РАН, 2010. – 214 с.

В девятнадцатом томе Трудов Института системного программирования публикуются статьи, содержащие результаты работ, проводимых в Институте в 2009-2010 годах.

ISBN 978-0-543-57630-9

Содержание

П р е д и с л о в и е

19-й том Трудов института системного программирования включает 10 статей, посвященных современным проблемам систем управления данными, поддержки принятия решений, программирования в ограничениях, анализа программ. Две статьи посвящены актуальным для различных приложений алгоритмам упаковки прямоугольников в полосу.

В статье С.Д. Кузнецова «Год эпохи перемен в технологии баз данных» приводится обзор публикаций за 2009 г., посвященных новым подходам к управлению данными (в частности, в среде cloud computing). Анализ проектов, которым посвящены эти публикации, показывают, что идея пересмотра архитектур СУБД всерьез воспринята серьезными исследователями и разработчиками средств управления данными.

В статье того же автора «MapReduce: внутри, снаружи или сбоку от параллельных СУБД?» обсуждаются подходы к использованию технологии MapReduce в аналитических СУБД. Рассматриваются подходы, при которых MapReduce реализуется внутри ядра параллельной СУБД, используется в качестве коммуникационной инфраструктуры новой параллельной СУБД и применяется автономно в симбиотическом единстве с параллельной СУБД.

Также две статьи представили Л.Е. Карпов и В.Н. Юдин. Первая статья называется «Обмен данными в распределенной системе поддержки решений». Под «обменом данными авторы понимают расширение возможностей локальной системы поддержки решений с целью привлечения опыта, накопленного другими пользователями в подобных системах. Рассматриваются два варианта обмена в сети, где присутствуют несколько подобных систем: виртуальная интеграция (аналог консилиума) и консолидация (импорт знаний).

Во второй статье – «Многопараметрическое управление на основе прецедентов» – описываются основные черты нового проекта, связанного с решением задачи многопараметрического управления объектом со сложным взаимным влиянием воздействий в ситуациях, когда влияние внешних факторов может оказаться взаимозависимым и даже противоречивым, когда трудно или невозможно получить точную модель поведения объекта. Предлагается подход к математической формализации понятия управления на теории принятия решений, методов добычи данных и вывода по прецедентам.

Статья «Объектно-ориентированное программирование в ограничениях: новый подход на основе декларативных языков моделирования данных» представили В.А. Семенов, Д.В. Ильин, С.В. Морозов и О.В. Сидяка. ООСР сочетает две о парадигмы программирования: объектно-ориентированное программирование (ООР) и логическое программирование в ограничениях (CLP). До сих пор не существует единого понимания, какие конструктивные очертания может приобрести реализация этого подхода при дальнейшей

проработке и развитии. Ключевыми вопросами при этом остаются выразительность описания прикладной задачи в ограничениях и ее алгоритмическая разрешимость. В статье предлагается и обсуждается новый системный подход к реализации ООСР на основе использования декларативных языков моделирования данных.

В статье В.А. Семенова и О.В. Сидяки «Теоретические и экспериментальные оценки сложности методов локального распространения в задачах программирования в ограничениях» обсуждаются вопросы универсальности и эффективности методов локального распространения значений и степеней свободы применительно к задачам программирования в ограничениях. Отмечается важность построения и использования комбинированных алгоритмов, обеспечивающих надежное решение широких классов задач за полиномиальное время. Для предложенного комбинированного алгоритма проводятся серии вычислительных экспериментов, моделирующих системы ограничений переменной размерности с разным характером зависимостей по данным.

Статья С.А. Мартишина и М.В. Храпченко «Упаковка прямоугольников в полосу модифицированным методом Нелдера-Мида с использованием генетического алгоритма» посвящена исследованию задачи упаковки прямоугольников в полубесконечную полосу. Известно, что эта задача является NP-трудной. Предложен новый эвристический алгоритм упаковки с использованием модифицированного метода Нелдера-Мида, генетического алгоритма и линейного программирования. Проведенное экспериментальное исследование предложенного алгоритма на известных тестовых примерах демонстрирует его преимущества.

В статье Н.Н. Кузюрина и А.И. Поспелова «Вероятностный анализ одного алгоритма упаковки прямоугольников в полосу» предлагается и теоретически исследуется on-line алгоритм упаковки прямоугольников в полубесконечную полосу. Получено, что для незаполненной площади упаковок, порождаемых алгоритмом в среднем, справедлива оценка $O\left(N^{2/3}(\ln N)^{1/3}\right)$.

В.А. Падарян, М.А. Соловьев и А.И. Кононов представили статью «Моделирование операционной семантики машинных инструкций», в которой предлагается модель, позволяющая описывать операционную семантику машинных инструкций для широкого класса целевых архитектур. Особенностью модели является то, что она предназначена для обратного по сравнению с классической компиляторной задачей тракта преобразований, но в то же время модель позволяет выполнять над ней различные оптимизирующие преобразования. Для описания целевой машины применяются внешние спецификации. Рассмотрена прототипная подсистема интерпретации модели.

Статья И.И. Каретина и В.А. Макарова «Энергосберегающая оптимизация кода за счет использования отключаемых компонентов процессора»

посвящена рассмотрению вопросов снижения энергопотребления в процессе исполнения кода программ. Объектом исследования является техника отключения отдельных компонентов процессора. Описываются теоретические аспекты модификации исходного кода с целью повышения энергетического эффекта от отключения компонентов.

Наконец, в статье А.И. Гетьмана, Ю.В. Маркина, В.А. Падаряна и Е.И. Щетинин. «Восстановление формата данных» отмечается, что одной из распространенных практических задач анализа бинарного кода является восстановление структуры полученного программой сетевого сообщения или считанного файла. При работе аналитика с защищенным бинарным кодом трудоемкость восстановления формата данных становится недопустимо большой. Предлагается метод автоматизированного восстановления формата данных, базирующийся на динамическом анализе бинарного кода. Метод позволяет восстановить иерархическую структуру изучаемых данных и выявлять определенную семантику полей.

Академик РАН В.П. Иванников

Год эпохи перемен в технологии баз данных

Кузнецов С.Д.
kuzloc@ispras.ru

Не дай Вам Бог жить в эпоху перемен
Древняя китайская мудрость

Аннотация. В 2007 г. Майкл Стоунбрейкер и его коллеги опубликовали несколько статей, в которых убедительно утверждали о необходимости радикального пересмотра архитектур СУБД. Эти утверждения подтверждались экспериментами с несколькими новыми архитектурами, предназначенными для поддержки приложений потоковых данных, анализа данных, оперативной обработки транзакций и т.д. В 2008 г. эти тенденции были подтверждены в Клермонтском отчете о направлениях исследований баз данных. Наиболее интересные публикации 2009 г., краткому обсуждению которых посвящена статья, свидетельствуют о том, что идея пересмотра архитектур СУБД всерьез воспринята серьезными исследователями и разработчиками средств управления данными.

1. Введение

Года три назад в мире управления данными возникло течение, авторитетные представители которого утверждали о необходимости радикальных перемен; о том, что «безразмерные» архитектуры универсальных систем управления данными, возраст которых измеряется десятками лет, не в состоянии удовлетворять потребности многих новых приложений; что наступает новая эпоха архитектурно упрощенных специализированных систем управления данными, предельно эффективно и экономично поддерживающих приложения соответствующих достаточно узких предметных областей. Больше всех было слышно Майкла Стоунбрейкера (Michael Stonebraker), его соратников и учеников, ряд экспериментальных и коммерческих разработок которых подтверждал эти утверждения. Однако тогда доводы Стоунбрейкера и Ко о наличии революционной ситуации в области управления данными и потребности в коренном отказе от традиционных архитектур СУБД вызывали сомнения как у независимых экспертов, так и (безусловно!) у специалистов компаний, производящих универсальные СУБД.

Одним из основных событий 2008 г. явилась очередная встреча ведущих специалистов в области управления данных (из академических и коммерческих кругов), результатом которой явился Клермонтский отчет об исследованиях в области баз данных [18]. В подобных отчетах анализируется текущее положение дел и предлагается программа исследований и разработок на ближайшие годы. В Клермонтском отчете (хотя и гораздо более умеренных выражениях, чем в статьях Стоунбрейкера и его сподвижников) также отмечалась потребность в пересмотре архитектур систем управления данными, а также, в частности, указывалось на потребность исследований архитектур СУБД, предназначенных для поддержки «облачных» приложений.

В 2009 г. появилось несколько публикаций, свидетельствующих о ряде успешных разработок систем управления данными, которые основываются на новых архитектурах. Я переводил и комментировал эти публикации. Однако в целом они демонстрируют некоторую общую тенденцию, которую, по моему мнению, стоит проанализировать специальным образом. Подобной попытке анализа и посвящена данная статья.

В разд. 2 будет кратко рассмотрена недавняя предыстория вопроса: основные идеи статей Стоунбрейкера и Ко, соответствующие исследования и разработки, а также некоторые (особенно важные, по моему мнению) положения Клермонтского отчета. Следующие разделы основаны на отдельных публикациях разных авторов 2009 года. В разд. 3 обсуждаются эксперименты по сравнению эффективности технологий MapReduce и массивно-параллельных систем баз данных. В разд. 4 рассматривается новая архитектура СУБД, предназначенная для поддержки Web-приложений в облачной инфраструктуре. Разд. 5 посвящен обсуждению проблем аналитической обработке больших объемов данных. В разд. 6 описываются основные идеи перспективной системы аналитических баз данных, в которой сочетаются возможности создания аналитических приложений на основе SQL и MapReduce. В разд 7 кратко характеризуется новый проект, направленный на создание свободно доступной системы баз научных данных. Наконец, в заключительном восьмом разделе подводятся итоги анализа и приводятся заключительные замечания.

2. Недавняя предыстория

Здесь действительно приходится говорить о совсем короткой предыстории, которая началась для меня только в конце 2006-го-начале 2007 гг.

2.1. Предсказание конца эпохи

В 2006-2008 гг. разными авторами, в число которых почти неизменно входил Майкл Стоунбрейкер, было опубликовано несколько статей, в которых с разной степенью убедительности утверждалось о необходимости радикальных перемен в области управления данными. Все эти статьи переводились на русский язык и комментировались автором данной статьи [1-4].

Комментариям и замечаниям по поводу этих статей была посвящена моя небольшая заметка [5]. Цитируя (с небольшими изменениями) эту заметку, можно отметить, что основные положения указанных статей состоят в следующем:

- Архитектура современных SQL-ориентированных СУБД появилась более 30 лет тому назад, когда рынок систем управления данными был единым, не фрагментированным на специализированные секторы. В это время СУБД вынужденно делались «безразмерными», пригодными для использования в любой области приложений баз данных. Эта «безразмерность» присутствует сегодня в продуктах основных поставщиков. Плюсами основных SQL-ориентированных СУБД является надежность и общая высокая производительность. Минусы – сложность, объемность и высокие накладные расходы, свойственные универсальности.
- За прошедшие 30 с лишним лет рынок систем управления данными сильно фрагментировался. Стали известными большие секторы рынка, для которых очень существенна высокая производительность приложений, которая не достигается или достигается с недопустимо большими затратами при использовании «безразмерных» СУБД. Экономически целесообразной стала разработка специализированных систем, которые ориентируются на эффективную поддержку заранее известных сценариев использования.
- За эти же тридцать лет в области управления данными была выполнена громадная исследовательская работа, результаты которой можно успешно применять для разработки специализированных систем.
- В связи с быстро меняющимися требованиями рынка успешными могут быть только такие новые продукты, которые можно вывести на рынок достаточно быстро – через год или два после начала разработки. Это еще один довод в пользу специализированных систем, нацеленных на решение узкого класса задач. В таких системах требуются более простые языковые средства, упрощается оптимизация запросов и другие аспекты, являющиеся традиционным камнем преткновения «безразмерных» систем.

В последние десять лет Стоунбрейкер последовательно воплощает в жизнь эти идеи. Пробным шаром явилась разработка специализированных средств управления потоковыми данными. На основе исследований и разработок, выполненных в ряде университетов США, была создана компания и промышленная система StreamBase [6], которая была хорошо принята финансовыми компаниями с Уолл-Стрит. На этом этапе Стоунбрейкер практически не конкурировал с «безразмерными» СУБД, для которых рыночный сектор потоковых данных, по-видимому, был слишком узок (кстати, в настоящее время ситуация, похоже, изменилась: в том числе, можно отметить появление компонента Streaminsight [7] в составе ожидаемого релиза Microsoft SQL Server V2).

Следующая попытка Стоунбрейкера состояла в создании нового SQL-ориентированного средства поддержки хранилищ данных с хранением данных по столбцам. И в этом случае созданная компания и промышленная система Vertica [8] основывается на предыдущих университетских исследованиях и разработках, которые, в свою очередь, опираются на многолетние работы других исследователей. Понятно, что в этом случае Стоунбрейкер уже начинает потенциально конкурировать как с «безразмерными» СУБД, так и с СУБД, изначально ориентированными на поддержку хранилищ данных. И в одной из статей приводятся результаты тестовых испытаний, показывающие, что в некоторых сценариях использование приложения, основанного на использовании Vertica, демонстрирует производительность, на два порядка более высокую, чем при использовании «безразмерной» коммерческой СУБД.

Наконец, теперь Стоунбрейкер полностью выходит на тропу войны с «безразмерными» СУБД, покушаясь на их основной, традиционный сектор рынка – OLTP. В исключительно интересном, пока еще университетском проекте H-Store [9] получены результаты испытаний разрабатываемой системы на эталонном тестовом наборе TPC-C, демонстрирующие превосходство над «безразмерной» коммерческой СУБД почти на два порядка.

Кроме того, в одной из статей приводится краткая характеристика и показатели производительности экспериментальной системы ASAP, ориентированной на поддержку математических баз данных. Результаты тоже впечатляют, хотя опубликованных данных относительно общей организации и интерфейсов системы явно недостаточно, чтобы можно было хорошо понять принципы ее организации. Впрочем, это явно будет исправлено при разработке системы SciDB [10] (см. разд. 7), которая основывается на ASAP.

В 2007 г. у меня вызывали определенные сомнения как доводы Майкла Стоунбрейкера и его коллег о целесообразности и даже необходимости полномасштабного отказа от традиционных архитектур СУБД, так и некоторые технические аспекты конкретных систем и их характеристики, описываемые в упомянутых выше статьях. В частности, мне казалось, что к настоящему времени накоплено так много технологических возможностей, как аппаратных, так и программных, что большинство пользователей и разработчиков предпочтет потратить больше средств и/или пожертвовать некоторой долей производительности, чем перейти к использованию радикально других программных средств. Удачный опыт внедрения принципиально новых средств управления потоковыми данными не опровергает эти соображения. В этом случае речь идет об области приложений, которую принципиально не удовлетворяло существовавшее положение дел. Отчасти я был согласен и с патриархальными взглядами Марка Ривкина, высказанными им в статье [11].

2.2. Клермонтский отчет

Раз в несколько лет ведущие представители исследовательского и производственного сообщества баз данных проводят встречи, которые обычно длятся два дня. На этих встречах обсуждается и оценивается состояние дел в области баз данных и формулируются темы исследований, которые будут наиболее актуальны в ближайшие годы. По результатам встреч принято подготавливать и публиковать отчет. Такие отчеты пользуются высоким авторитетом в сообществе баз данных и оказывают серьезное влияние на развитие исследований и разработок. Все опубликованные отчеты доступны в Internet, для многих имеются переводы или пересказы на русском языке [12-17].

В мае 2008 г. состоялась новая встреча. Она проходила в отеле Claremont Resort в г. Беркли, штат Калифорния, и поэтому отчет получил название «Клермонтского». Джо Хеллерстейн (Joseph M. Hellerstein) подготовил и поддерживает специальный сайт (http://db.cs.berkeley.edu/claremont/), на котором и был опубликован первый вариант отчета [18]. Имеется его полный перевод на русский язык.

Я не буду здесь подробно пересказывать отчет, а лишь отмечу те его места, которые имеют отношение к дальнейшему материалу этой статьи.

2.2.1 Проблемы архитектур серверов баз данных

В качестве одной из наиболее актуальных тем исследований отмечается *пересмотр архитектуры серверов баз данных*. В отчете утверждается (здесь и ниже текст цитируется не вполне точно), что

«…у современных развитых коммерческих систем реляционных баз данных имеются хорошо известные ограничения. Обеспечивая широкий набор возможностей, они показывают пиковую производительность только для очень ограниченного набора режимов. В последнее десятилетие появилось много популярных задач, связанных с обработкой больших объемов данных, для которых реляционные СУБД обеспечивают плохое соотношение «цена/производительность», и при решении которых от использования РСУБД пришлось отказаться.»

Как видно, эти утверждения напрямую перекликаются с доводами Майкла Стоунбрейкера и его сторонников, рассмотренными в предыдущем подразделе.

С другой стороны, в отличие от общего «революционного» тона статей Стоунбрейкера и Ко, в отчете говорится, что

« имеются два разных направления: расширение диапазона применимости универсальных систем баз данных и радикальное повышение производительности путем разработки специализированных систем баз данных для конкретных прикладных областей. У обоих направлений имеются свои достоинства, и очевидная общность конечных целей подсказывает, что

работы в этих направлениях можно выполнять с взаимной пользой: специализированные методы можно повторно использовать в более универсальных системах, а использование архитектурных компонентов универсальных систем может позволить быстрее создавать прототипы новых специализированных систем.»

На самом деле, так и происходит. Наиболее развитые традиционные СУБД непрерывно развиваются за счет внедрения технологий, разработанных для специализированных (часто экспериментальных) систем (заметим, что это отнюдь не снижает их уровень сложности), а при разработке специализированных систем с новой архитектурой грамотные разработчики стремятся к максимальному использованию проверенных методов.

К числу наиболее важных исследовательских тем в этой области относятся:

1. разработка систем для кластеров многоядерных процессоров, в которых имеется ограниченный и неоднородный доступ к памяти вне кристалла;

2. использование удаленной основной и флэш-памяти в качестве среды персистентного хранения данных в дополнение к памяти на магнитных дисках;

3. разработка унифицированного подхода к постоянно выполняемой адаптации и самонастройке оптимизации запросов и физических структур хранения данных;

4. сжатие и шифрование данных на уровне хранения, интегрированное со структурой хранения и оптимизацией запросов;

5. разработка систем, опирающихся на не реляционные модели данных, вместо того, чтобы «впихивать» эти данные в таблицы;

6. нахождение компромиссов между согласованностью и доступностью для достижения лучшей производительности и масштабности уровня тысяч машин;

7. разработка СУБД, учитывающих потребление энергии, которые ограничивают энергопотребление без ущерба для масштабируемости.

Для целей обсуждения в этой статье основной интерес в этом списке представляет п. 6. Из остальных наиболее актуальным мне кажется п.2. В этой статье он подробно не обсуждается, но совершенно очевидно, что бурное развитие технологии флэш-памяти и «твердотельных дисков», доступность таких носителей с емкостью в сотни гигабайт создает новые перспективы для организации систем управления данными.

В связи с этим на меня произвела большое впечатление статья Гоца Грейфа «Правило пяти минут двадцать лет спустя, и как флэш-память изменяет правила» [19]. В этой статье автор, в частности, задает вопрос:

«Смогут ли традиционные системы, в которых вместо традиционных дисков применяется флэш-память, конкурировать со специализированными системами баз данных в основной памяти по производительности, общей стоимости владения, стоимости разработки и сопровождения, времени выхода на рынок и выпуска очередных релизов и т.д.?»

Другими словами, имеются основания предполагать, что применение флэш-памяти может существенно изменить как архитектуру, так и номенклатуру систем управления данными.

2.2.2 Декларативное программирование

Следующей важной темой исследований, выделяемой в Клермонтском отчете и имеющей отношение к данной статье, является *декларативное программирование для новых платформ*. В отчете говорится, что

«…актуальность этой проблемы возрастает буквально экспоненциально, поскольку программисты нацеливаются на все более сложные среды, включая многоядерные микропроцессоры, распределенные службы и платформы «облачных вычислений» (cloud computing). Неквалифицированным программистам требуется возможность легко писать надежные программы, масштабируемые при возрастании числа процессоров как в слабосвязанных, так и в сильносвязанных архитектурах».

Авторы отчета полагают, что подходы, облегчающие и упрощающие программирование приложений, которые ориентированы на обработку данных, окажут серьезное влияние на программирование в целом.

В качестве первого примера подобного подхода рассматривается MapReduce. Отмечается, что:

«…в подходе Map-Reduce привлекает его простота; он основывается на языковых методах и методах распараллеливания по данным, известных в течение десятилетий. Для исследователей баз данных значимость подхода Map-Reduce состоит в том, что он демонстрирует преимущества программирования с распараллеливанием по данным новым классам разработчиков».

Со своей стороны замечу, что этот подход действительно чрезвычайно популярен, особенно среди молодежи. С другой стороны, как показывают исследования, обсуждаемые в разд. 3, очень важно не переоценивать этот подход и относиться к нему, как к первому шагу на пути упрощения параллельной обработки данных. Интересно также отметить лояльное отношение в MapReduce среди некоторых разработчиков новых систем баз данных. В связи с этим см. разд. 6.

Вторым примером является появление новых декларативных языков, основанных на Datalog. В ряде сценариев использование подобного декларативного языка позволяет на порядки величин сократить объем кода, одновременно обеспечивая его распределенное или параллельное исполнение. Как отмечается в отчете, группы, выполняющие соответствующие исследования, очень слабо скоординированы, и возрождение декларативных языков в новых контекстах пока происходит практически стихийным образом. Тем не менее, сам факт реинкарнации Datalog кажется мне крайне интересным и потенциально многообещающим.

Наконец, третий пример происходит из области программирования корпоративных приложений. Сближению сред программирования и управления данными, устранению проблемы потери соответствия (impedance mismatch) способствует возникновение новых языковых расширений, таких как LINQ [20] и Ruby on Rails [21]. Однако в отчете отмечается, что:

«… в этих пакетах пока серьезно не решается проблема программирования с использованием нескольких машин. Для корпоративных приложений ключевым решением распределенной разработки является разделение логики приложения и данных между несколькими «звеньями» («tier»): Web-клиентами, Web-серверами, серверами приложений и серверной СУБД. Особо ценной здесь является независимость данных, позволяющая создавать программы без предварительного принятия долговременных решений о физическом размещении программ и данных в разных звеньях».

Как полагают авторы отчета, одним из существующих языков, который мог бы способствовать подобному декларативному программированию, является XQuery, поскольку формат XML часто используется на разных уровнях протоколов. Практическим подтверждением справедливости этого предположения является работа, рассматриваемая в разд. 4.

2.2.3 Управление данными в «облачной» инфраструктуре

Как отмечается в отчете, облачные сервисы могут повысить эффективность поставщиков приложений за счет сокращения объема требуемых начальных вложений и сокращения стоимости владения. Доступен ряд разнообразных облачных сервисов, включая прикладные сервисы (salesforce.com [22]), сервисы хранения (Amazon S3 [23]), вычислительные сервисы (Google App Engine [24], Amazon EC2 [25]) и сервисы данных (Amazon SimpleDB [26], Microsoft SQL Server Data Services [27], Google’s Datastore [28]).

Общей проблемой поставщиков облачных сервисов является потребность в компромиссе между функциональными возможностями и эксплуатационными расходами. Интерфейсы существующих облачных служб намного более ограничены, чем в традиционных системах баз данных. В них поддерживаются минималистские языки запросов и ограниченные гарантии согласованности данных. Это затрудняет программирование приложений, но позволяет создавать более предсказуемые службы, в которых обеспечиваются

соглашения об уровне обслуживания, трудно достижимые для полнофункциональных служб данных на основе языка SQL.

Следующая проблема – достижения высокого уровня управляемости. Решение этой проблемы для систем в облачной среде осложняется ограниченным человеческим вмешательством, сильно изменчивыми рабочими нагрузками и разнообразием совместно используемых инфраструктур. При решении проблемы требуется учитывать наличие различных подходов к виртуализации ресурсов – от виртуализации аппаратных средств до использования в службе данных одной СУБД для управления несколькими базами данных с разными схемами. Для достижения управляемости требуется развивать и совершенствовать методы самоуправления системами.

Серьезную проблему представляет масштабируемость будущих облачных систем управления данными. Технология, применяемая при разработке современных SQL-ориентированных СУБД, просто не позволяет масштабировать их до 1000 узлов. По всей видимости, в будущих системах нужно ограничивать требования к их транзакционности или учитывать семантику данных. Дополнительные исследования нужны для понимания реальной масштабируемости облачной инфраструктуры. В связи с этим см. разд 3 и 4 настоящей статьи.

Наконец, при совместном использовании физических ресурсов в облачной инфраструктуре требуется обеспечение безопасности и конфиденциальности данных, которые не могут гарантироваться за счет наличия физического разграничения машин или сетей.

Рассмотрим теперь, как утверждения, предположения и прогнозы, рассмотренные в этом разделе, проявляются в работах, опубликованных в 2009-м году.

3. MapReduce и параллельные системы баз данных

Молодежи свойственно увлекаться новыми идеями. Идея MapReduce [29], выдвинутая и реализованная сначала Google, а потом и сообществом open source в проекте Hadoop [30] почти мгновенно овладела молодыми массами. Причем даже теми представителями компьютерной молодежи, которые получили хорошее образование и последующий практический опыт в области систем управления базами данных.

Мне неоднократно приходилось слышать от молодых коллег, что они считают достоинствами MapReduce отсутствие схемы данных (в том числе, и отсутствие поддержки типов данных) и даже потребность в явном программировании конструкций, которые испокон веков поддерживались в СУБД на уровне высокоуровневых языковых конструкций языка SQL. Дополнительным стимулом к применению MapReduce была привязка этой технологии к «облачным» вычислениям, возможность практически бесплатно арендовать виртуальный кластер с большим числом узлов и развернуть на нем

свою MapReduce программу, почти автоматически достигнув громадной производительности своего приложения.

До поры до времени представители старшего и среднего поколений сообщества баз данных ограничивались ворчанием в адрес MapReduce, что, в свою очередь, еще больше привлекало молодежь к использованию соответствующих средств. Действительно, раз «старики» ворчат, значит, они просто не понимают, что средства управления данными их поколений просто устарели и нужно переходить к использованию новых, прогрессивных технологий. Больше других ворчали (и получали достойную отповедь от молодежи) Майкл Стоунбрейкер и Дэвид Девитт (David J. DeWitt), что, в частности, проявилось в написании и публикации ими в январе 2007 г. в коллективном блоке Database Column [31] заметок [32-33].

И вот, в 2008 г. ворчание стариков выразилось в инициировании ими чрезвычайно интересного проекта по практическому сравнению технологии MapReduce с технологиями параллельных СУБД категории sharing nothing. Результатам этого проекта посвящена статья [34].

3.1. Эксперименты и их результаты

Эксперименты, описываемые в этой статье, основывались на выполнении нескольких несложных аналитических задач на кластере из ста узлов с использованием двух параллельных СУБД категории sharing-nothing (применялись СУБД Vertica и некоторая СУБД с традиционным построчным хранением данных, именуемая в статье СУБД-X) и реализации MapReduce от Hadoop. Очень интересно, как авторы обосновывают свой выбор для экспериментов относительно небольшого кластера, тогда как сторонники MapReduce обычно говорят об абсолютных преимуществах этой технологии при использовании кластеров с тысячами узлов.

Как утверждается в статье, такая масштабная аппаратура не требуется современным высокоэффективным параллельным СУБД даже при поддержке предельных в настоящее время по объему петабайтных баз данных. Например, в конфигурации СУБД Teradata, применяемой в eBay, 72 узлов (в каждом узле два четырехъядерных процессора, 32 гигабайта основной памяти и 104 300-гигабайтных диска) оказывается достаточно для управления реляционными данными объемом около 2,4 петабайт. Так что с самого начала статьи, по сути дела, утверждается, что сверхвысокая масштабируемость параллельных СУБД в действительности не требуется (заметьте, как это перекликается с тезисом Клермонтского отчета о невозможности должного масштабирования современных СУБД в облачной инфраструктуре).

При том, что авторы статьи уделили тщательное внимание программированию тестовых аналитических задач для среды MapReduce, на кластере из 100 узлов СУБД-X показала среднюю производительность, большую соответствующего варианта для MapReduce в 3,2 раза, а СУБД Vertica оказалась в 2,3 раза быстрее СУБД-X. Авторы предполагают, те же относительные соотношения

сохранились бы и на кластере с 1000 узлами, хотя такое оборудование просто не требуется для поддержки параллельными системами баз данных актуальных сегодня баз данных петабайтного масштаба.

3.2. В чем причины?

Преимущество в производительности, демонстрируемое параллельными СУБД, оказывается возможным за счет использования технологий, совершенствуемых в течение десятилетий:

- индексация на основе B-деревьев для оптимизации выборки данных;
- новые механизмы хранения данных (в частности, поколоночное хранение);
- эффективные методы сжатия данных, позволяющие выполнять операции прямо над сжатыми данными;
- параллельные алгоритмы выполнения операций над реляционными данными.

С другой стороны, при использовании MapReduce потребовалось существенно меньшие времени на загрузку данных: иногда это происходило в три раза быстрее, чем в случае Vertica, и 20 раз быстрее, чем для СУБД-X (в основном, из-за отсутствия схемы данных). Существенно проще происходит установка и конфигурирование MapReduce, чем аналогичные действия для параллельных СУБД. Авторы статьи приходят к выводу, что это является одним и основных факторов, привлекающих к MapReduce разработчиков приложений. Другими факторами является простота использования (SQL часто оказывается слишком сложным для начинающих программистов аналитических приложений; с другой стороны примитивность MapReduce приводит к росту трудозатрат на разработку приложений) и большая устойчивость к сбоям отдельных узлов кластера.

Принципиальным отличием MapReduce от систем баз данных является отсутствие схемы. В результате возникает потребность разбора записей на стадии выполнения приложения, уменьшается польза от сжатия данных. Все это является одной из причин снижения производительности. Кроме того, при отсутствии схемы невозможно обеспечить унифицированное хранение информации, требуемой для оптимизации декларативных запросов: об имеющихся индексах, о применяемых способах разделения таблиц, о гистограммах, описывающих распределения данных и т.д.

Общим выводом статьи является потребность в совершенствовании обеих технологий. MapReduce нуждается в более выразительных средствах программирования, чтобы можно было снизить упомянутые выше трудозатраты на разработку. В параллельных СУБД требуются более совершенные методы распараллеливания определяемых пользователями функций. Первыми признаками грядущей интеграции SQL и MapReduce являются системы Greenplum [35] (см. разд. 6) и *n*Cluster компании Aster Data [36]. Хочу также заметить, что в августе 2009 г. поддержка MapReduce

появилась и в компании Vertica [37], являющейся, как уже говорилось, детищем ранее неутомимого борца с этим подходом Майкла Стоунбрейкера.

4. Cloud Computing и новые критерии оптимальности архитектуры СУБД

Хотя «облачные» вычисления (cloud computing) постепенно входят в практику все большего числа разработчиков приложений, остается открытым вопрос, *какими должны быть средства управления данными «в облаках»*? Некоторые люди полагаются на использование распределенных файловых систем и подхода map/reduce, другие говорят о целесообразности использования массивно-параллельных систем баз данных (см. разд. 2), третьи пытаются приспособить к облачной инфраструктуре традиционные SQL-ориентированные СУБД (например, SQL Azur [38] от Microsoft; кстати, после появления в Microsoft проекта массивно-параллельной СУБД Madison (в настоящее время соответствующий продукт называется SQL Server 2008 R2 Parallel Data Warehouse [39]), возможно, компания присоединится к лагерю параллельных облачных СУБД).

Свой подход к «облачным» СУБД предлагают Даниела Флореску (Daniela Florescu) и Дональд Коссман (Donald Kossmann) в статье [40]. В качестве краткой справки заметим, что эти авторы наиболее известны своим активным участием в разработке стандарта языка XQuery и другими работами, связанными с языком XML.

Статья, по сути, состоит из двух основных частей. В первой части вводятся и обосновываются новые критерии оптимальности СУБД, соответствующие современной специфике (наличию облачной инфрастуктуры и крупных центров данных с дешевыми аппаратными средствами). Во второй части предлагается и объясняется архитектура СУБД, соответствующая этим критериям.

4.1. Новые критерии оптимизации

Традиционные критерии оптимизации, по мнению авторов, выглядят следующим образом:

> При заданном наборе аппаратных ресурсов и гарантировании полной согласованности данных (т.е. при поддержке ACID-транзакций) требуется минимизировать время ответа за запросы и максимизировать пропускную способность системы по отношению к поступающим запросам.

Они считают, что в новых условиях они должны быть такими:

> При заданных требованиях к производительности приложений (пиковая пропускная способность, предельно допустимое время ответа) требуется минимизировать требуемые аппаратные ресурсы и максимизировать согласованность данных.

Основным показателем, нуждающемся в оптимизации, является оценка денежных *затрат*. Наиболее важный вопрос состоит в том, сколько требуется машин, чтобы удовлетворить требования к производительности на заданной рабочей нагрузке. Аппаратные ресурсы – это теперь не одноразовая инвестиция; это отдельная существенная статья ежемесячных расходов IT.

Для большинства приложений *производительность* является заданным ограничением, а не целью оптимизации. Требуется, чтобы поддерживалась заранее заданная пиковая рабочая нагрузка. Имеется возможность поддержки, по существу, любой рабочей нагрузки, вопрос только в том, сколько это будет стоить, и чем дешевле, тем лучше.

Насущной потребностью любой современной информационной системы является неограниченная *масштабируемость*. Это требование удовлетворяется, если расходы на IT растут линейно по мере роста бизнеса, и этот рост ничем не ограничивается. У традиционных СУБД функция стоимости является ступенчатой, а масштабируемость – ограничена.

Для многих организаций *предсказуемость* приложений настолько же обязательна, что и масштабируемость. Имеется в виду предсказуемость и производительности, и стоимости. Оптимизация 99% операций более важна, чем оптимизация в среднем. Прилагаются огромные усилия для сокращения расходов на администрирование и достижения более устойчивой производительности СУБД, но радикальных успехов достичь до сих пор не удалось.

Транзакции в стиле ACID, обеспечивающие полную *согласованность* данных, и сервис-ориентированная архитектура, помогающая структурировать и развивать приложения, плохо совместимы. Более того, большинству приложений просто не требуется полная согласованность. Более приоритетным является требование стопроцентной доступности данных. В современных приложениях согласованность данных является целью оптимизации, а не ограничением, как в традиционных системах баз данных.

Гибкость – это возможность настройки программной системы к индивидуальным требованиям пользователя. По мере усложнения приложений гибкость как цель разработки становится все более важной. В традиционных системах OLTP это требование отсутствовало, так что гибкость не являлась целью оптимизации в традиционных системах управления данными. Понятно, что гораздо проще обеспечить должные производительность и масштабируемость, если система не обладает гибкостью.

4.2. Новая архитектура СУБД

В традиционной архитектуре СУБД (рис. 1) все запросы инициируются пользователями на уровне представления, например, с использованием Web-браузера. Логика приложения программируется на среднем уровне; на среднем же уровне поддерживаются такие функциональные средства, как

Web-сервер. Все управление данными производится на низшем уровне с использованием СУБД.

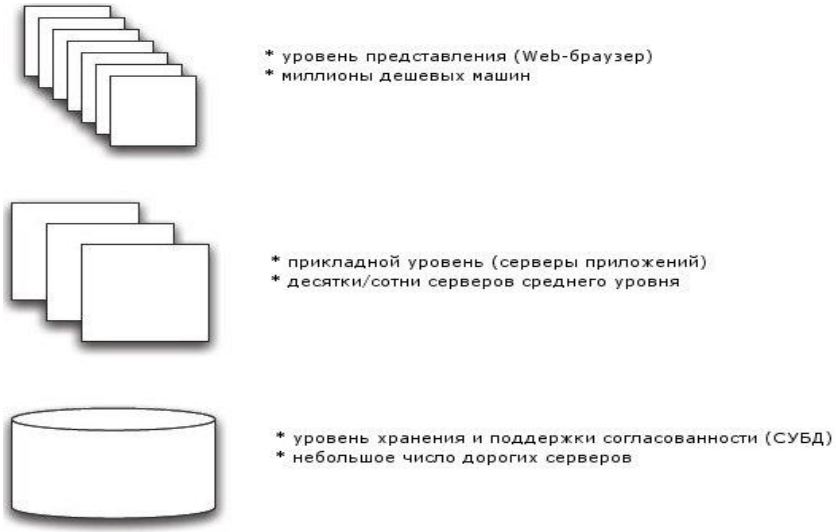


Рис. 1. Традиционная архитектура СУБД

Эта архитектура практически идеально отвечает критериям оптимальности традиционных СУБД. Согласованность данных поддерживается СУБД на нижнем уровне. Основная цель оптимизации (минимизация времени отклика и максимизация пропускной способности) достигается за счет применения на всех уровнях ряда отработанных методов (индексация, кэширование, разделение и т.д.). Верхние два уровня почти неограниченно масштабируются (масштабируемость нижнего уровня ограничена).

Однако целям новой оптимизации традиционная архитектура не удовлетворяет. Во-первых, она рассчитана на обеспечение ожидаемой пиковой производительности, которая может на несколько порядков превышать реальную среднюю производительность. В результате б*о*льшая часть дорогостоящих аппаратных ресурсов простаивает. То же можно сказать о многих компонентах программного обеспечения СУБД, за которые приходится платить даже в том случае, когда они не требуются приложению. Во-вторых, традиционная архитектура не обеспечивает предсказуемость, поскольку при наличии многопользовательского доступа трудно понять, что происходит на уровне СУБД. В-третьих, как уже отмечалось, на нижнем уровне архитектуры масштабируемость ограничена. Наконец, не поддерживается гибкость, поскольку на всех трех уровнях используются разные модели данных и программирования (XML/HTML и скриптовые языки

на верхнем уровне, объектно-ориентированные подход на среднем уровне и SQL – на нижнем уровне).

Двумя принципами разработки приложений в трехзвенной архитектуре являются *контроль* и *передача запросов*. Первый принцип означает, что весь доступ к данным полностью контролируется СУБД, а второй – что как можно большая часть логики приложений должна выполняться поблизости от данных, т.е. внутри СУБД (на основе хранимых процедур, определяемых пользователями типов данных и функций), По мнению авторов, эти два принципа подрывают масштабируемость, предсказуемость и гибкость, приводят к удорожанию и усложнению СУБД (хотя вполне согласуются с целями традиционной архитектуры).

Предлагаемая авторами архитектура показана на рис. 2. Здесь на нижнем уровне поддерживается только крупномасштабное распределенное хранение данных на основе, например, облачной службы Amazon S3. Согласованность данных обеспечивается на среднем уровне, причем эта согласованность не является строгой. Она поддерживается за счет применения всеми серверами приложений общих соглашений при чтении и записи данных через службу нижнего уровня.

Рис. 2. Новая архитектура

На всех уровнях можно использовать дешевую аппаратуру и обеспечивать неограниченную масштабируемость. На каждом уровне допускается выход из строя любого узла: на нижнем уровне это возможно из-за репликации и слабой согласованности данных; на верхних – за счет работы узлов без сохранения состояния.

На основе этой новой архитектуры в компании 28msec [41] реализуется система Sausalito. В качестве службы хранения данных используется Amazon S3, на среднем уровне применяются сервисы прикладного уровня Amazon EC2. Вся традиционная функциональность управления базами данных реализуется на этом же уровне на основе языка XQuery с расширениями, обеспечивающими возможность модификации данных (понятно, что в данном случае речь идет об XML-данных). Тот же язык используется для разработки Web-приложений, на поддержку которых и ориентирована система Sausalito.

4.3. Все ли так хорошо с новой архитектурой?

На мой взгляд, статья написана очень последовательно и логично. Правда, несколько смущает сходство предлагаемой архитектуры приложений баз данных с архитектурами файл-серверных СУБД. Фактически, в подходе Флореску и Коссмана Amazon S3 выполняет роль файл-сервера, а вынесение службы запросов и других функций СУБД на уровень приложения напоминает организацию систем, существовавших до появления клиент-серверных архитектур, например, Informix SE.

И, конечно, что там не говори насчет отказа от принципа «передачи запросов», некоторые сомнения вызывает передача по Internet от узлов Amazon S3 в узлы серверов приложений, как минимум, XML-документов целиком (а может быть и коллекций XML-документов). Непонятно, как при этом удается гарантировать, что время ответа на запрос не превышает заданные ограничения (если, конечно, не считать, что пользователи могут спокойно подождать и несколько минут).

Кроме того, я не уверен, что разработчики приложений придут в полный восторг от необходимости использования XQuery не только для запросов XML-данных, но и для написания логики приложений. Для системы это, конечно, очень удобно, а на месте разработчиков я бы, пожалуй, предпочел использовать для программирования что-нибудь более привычное.

Хотя, конечно, здесь нужно учитывать, что Флореску и Коссман входят в число родоначальников XQuery, и с их точки зрения, по-видимому, программирование приложений на XQuery является вполне естественным и удобным. Кстати, по этому поводу полезно почитать статью [42].

А может быть, я напрасно сомневаюсь, этот возврат к архитектурному прошлому закономерен, очень большие XML-документы в Web-приложениях не встречаются, а разработчики Web-приложений с энтузиазмом перейдут от программирования на PHP и Python к программированию на XQuery. Авторы достаточно убедительно демонстрируют, что их «новая» архитектура (а любое

«новое», как известно, – это хорошо забытое «старое») обеспечивает неограниченное масштабирование приложений, гибкость системы и предсказуемость ее поведения. В конце концов, в старые и добрые времена файл-серверных СУБД не было облачных вычислений, не было Web-приложений и требования к системам баз данных были другими.

5. Как справиться с большими данными?

В области управления данными ежегодно публикуется множество статей – в сборниках трудов многочисленных конференций, в специализированных журналах и изданиях универсальной софтверной тематики. Однако подавляющее большинство этих статей касается чрезвычайно узкой тематики, понятной только специалистам, профессионально занимающимся аналогичными вопросами. Лишь немногие люди решаются публично выразить свою более общую точку зрения, затрагивающую проблемы области в целом. И круг таких людей очень ограничен: Майкл Стоубрейкер, раньше – Джим Грей (Jim Gray), который, к всеобщему несчастью, пропал в океане зимой 2007 г. [43], Кристофер Дейт (Chris Date), может быть, еще несколько человек. Новые люди в этом «элитном» сообществе появляются крайне редко.

Поэтому меня сразу заинтересовала статья Адама Якобса «Патологии больших данных» [44]. Привлекли амбициозное название статьи, а также неизвестность и явное нахальство автора, решившегося высказаться на такую тему. Как удалось выяснить, первой специализацией автора статьи была лингвистика, а степень PhD он получил в области вычислительной нейропсихологии. Ему приходилось заниматься аналитическими исследованиями больших объемов данных, и с начала 2000-х он работает в компании 1010data Inc. [47], где руководит разработкой аналитической СУБД Tenbase [46].

Статья Якобса настолько меня затронула, что я написал по ее поводу отдельную заметку «О точности диагностики патологий» [45], в которой серьезно (и, полагаю, заслуженно) раскритиковал автора. Не буду здесь пересказывать эту заметку, а остановлюсь только на том, что связывает статью Адама Якобса с основной темой моей статьи.

На основе своих рассуждений Якобс приводит следующее «метаопределение» больших данных:

> Большими являются данные, размер которых вынуждает нас выходить за пределы проверенных временем методов, широко распространенных в данное время.

В середине 1980-х гг. приходилось иметь дело с наборами данных, которые были настолько масштабными, что для их обработки требовалось использовать специальные робототехнические решения, автоматизирующими работу с тысячами магнитных лент. В 1990-е гг. для анализа больших данных не хватало возможностей Microsoft Excel и персональных компьютеров, и в

этом случае использовались рабочие станции с ОС Unix и более серьезным программным обеспечением. В настоящее время большими являются такие данные для обработки которых оказываются недостаточными средства традиционных SQL-ориентированных СУБД и настольных статистических пакетов; требуется массивно-параллельное программное обеспечение.

Как видно, по мнению автора, с проблемой больших данных всегда приходится сталкиваться именно при анализе данных; в области оперативной обработки транзакций такой проблемы нет. И решить проблему больших данных в каждый период времени можно только за счет отказа разработчиков программных систем от традиционных, типовых решений, за счет понимания истинной природы имеющихся аппаратных средств и привлечения всего многообразия ранее созданных методов и алгоритмов. Мне кажется, что в этом идеи Якобса полностью созвучны идеям Стоунбрейкера, хотя он вообще на него не ссылается.

Что касается конструктивных предложений, то в статье Якобса я их вижу две:

- несмотря на совершенствование устройств внешней памяти (включая устройства флэш-памяти), их можно эффективно использовать только при последовательном доступе; поэтому следует избегать неоптимальных схем доступа к внешней памяти;
- в распределенных системах баз данных репликация должна служить не только целям повышения уровня доступности и отказоустойчивости системы; для повышения эффективности системы баз данных следует поддерживать реплики данных с разным физическим представлением.

Судя по всему, эти предложения используются при разработке упомянутой выше распределенной СУБД Tenbase [46]. Как я уже отмечал, многие высказывания Якобса заслуживают критики. Еще раз отсылаю читателей к своей заметке [45].

6. Новый взгляд на место аналитиков в системе баз данных

С идеями Стоунбрейкера интересным образом сплетается статья [48]. Главным автором статьи, безусловно, является Джо Хеллерстейн, профессор Калифорнийского университета в Беркли, научный руководитель компании Greenplum [35], в которой работают остальные авторы статьи, соратник Майкла Стоунбрейкера по разработке СУБД Postgres.

В статье утверждается, что традиционный, «ортодоксальный» подход к организации корпоративных хранилищ данных (Enterprise DataWarehouse, EDW), основанный классиками этого направления Эдгаром Коддом (Edgar Codd), Биллом Инманом (Bill Inmon) и Ральфа Кимболла (Ralph Kimball), не соответствует реалиям настоящего времени. В этом традиционном подходе главным является тщательное проектирование и развитие схемы EDW, служащей основой интеграции корпоративных данных. Сервер баз данных,

поддерживающий EDW, является основным вычислительным средством, центральным, масштабируемым механизмом корпоративной аналитики. EDW контролируется специально назначаемыми сотрудниками IT, которые не только сопровождают систему, но и тщательно контролируют доступ к ней.

В настоящее время хранение данных обходится настолько дешево, что за счет собственного бюджета базу данных громадного масштаба может иметь даже небольшое подразделение корпорации. Число внутрикорпоративных источников данных непрерывно возрастает, включая журналы Web-серверов, архивы электронной почты и т.д. Все отчетливее понимается важность аналитики, разной аналитики в разных подразделениях одной и той же компании. Для аналитики нужны как можно более свежие данные, целесообразна поддержка отдельных механизмов сбора и анализа данных для разных подразделений.

6.1. Вся власть – аналитикам!

Исходя из этого изменения ландшафта анализа данных, авторы предлагают подход MAD, являющийся акронимом от *magnetic (магнетичность), agile (гибкость)* и *deep (основательность)*. В совокупности эти три характеристики подхода образуют *MAD skills*, новые, *МОГучие способности* анализа данных.

По мнению авторов, ортодоксальные EDW «отталкивают» новые источники данных: для интеграции в хранилище данных заново возникшего источника данных требуется, вообще говоря, изменить схему EDW, настроить (или даже создать) процедуру ETL (Extract-Transform-Load) и применить эту процедуру. Весь этот процесс может длиться месяцами, а может и вообще не закончиться. В результате аналитики остаются без данных, анализ которых мог бы принести бо́льшую пользу компании. Предлагаемая концепция *магнетичного* хранилища данных означает, что данные должны становиться доступными для анализа сразу после появления нового источника. Аналитики должны сами решать, что для них важнее, полная очищенность и согласованность данных или же быстрота доступа к ним. Диктовать условия доступа к данным должны не администраторы EDW, а пользователи, т.е. аналитики.

Процессы развития хранилища данных, должны быть *быстрыми* и *гибкими.* Должна допускаться возможность быстрого изменения физического и логического содержимого аналитической базы данных. В частности, это означает, что у «могучего» хранилища данных может отсутствовать ортодоксальная жесткая схема (хотя ее могут поддерживать сами аналитики). Может отсутствовать и ортодоксальная процедура ETL: аналитики сами должны решать, какой уровень согласованности данных им требуется.

В современных СУБД поддерживаются только минимальные средства, требуемые для анализа данных (типа CUBE BY). Серьезные статистические пакеты (SAS, Mathlab, R) выполняются на рабочих станциях, что ограничивает объем анализируемых данных и требует их передачи по сети. *Основательность* нового подхода к аналитике означает возможность

разработки самими аналитиками и размещения поблизости от данных, внутри аналитической СУБД статистических пакетов любой сложности. При этом аналитикам не должен навязываться конкретный стиль разработки этих пакетов. Например, им должны быть равно доступны технологии SQL и MapReduce.

6.2. Новые возможности за счет старых средств

В соответствии с подходом MAD в компании Greenplum на основе свободно доступных исходных текстов СУБД Postgres (и PostgreSQL) разработана параллельная аналитическая СУБД, пригодная, в частности, для использования в среде cloud computing. Насколько я понимаю, основные усилия были затрачены на разработку массивно-параллельного варианта системы, поскольку Postgres никогда не была параллельной системой. Естественно, о качестве этой части работы, не испытав ее в реальных производственных условиях, говорить невозможно.

Но зато все остальные расширения (поддержка статистических пакетов вблизи от данных, реализация MapReduce) основываются на использовании стандартных средств Postgres, которая с самого начала разработки в середине 1980-х гг. задумывалась Стоунбрейкером, как расширяемая система. Определяемые пользователями типы данных и функции, возможность определения новых структур хранения и методов доступа, естественно, позволяют сравнительно легко добавлять в систему новые, даже совсем не ожидавшиеся возможности. В связи с этим меня всегда интересовал вопрос, насколько безопасны эти расширения, не могут ли они подорвать общую работоспособность системы? Ответ на этот вопрос в контексте Postgres (и позднее в контексте Informix Universal Server) я получить так и не смог. Не дают его и авторы данной статьи.

В заключение этого раздела отметим сходства и различия подхода Хеллерстейна с подходов Флореску и Коссмана, обсуждавшимся в разд. 4. В обоих случаях критикуется традиционная архитектура, в обоих случаях предлагается некоторая новая архитектура, более пригодная для использования в современных условиях, в частности, на основе инфраструктуры cloud computing. Основным отличием является, на мой взгляд, отношение к принципу «передачи запросов». Флореску и Коссман жертвуют этим принципом ради достижения масштабируемости и предсказуемости (а также в угоду SOA), а Хеллерстейн, по сути, целиком на нем основывается. Конечно, в первом случае речь идет о Web-приложениях обработки данных, а во втором – об аналитике. Но в любом случае идея физического приближения вычислений к данным всегда была притягательной: зачем передавать по сети данные, если можно передавать существенно менее объемные результаты их обработки?

7. Научные базы данных и проект SciDB

У ученых разных специальностей (физиков, химиков, астрономов, социологов и т.д.) исторически существуют сложные взаимоотношения с миром баз данных. Это видно, например, при анализе проектов Российского фонда фундаментальных исследований, связанных с созданием научных информационных систем. Им неудобны главенствующие в мире SQL-ориентированные СУБД, предназначенные, главным образом, для поддержки разных видов бизнеса.

В свое время об этом много думал и писал Джим Грей (см. например, [49]). В его честь по инициативе, прежде всего, Майкла Стоунбрейкера и Дэвида Девитта в начале 2009 г. образован проект SciDB (http://www.scidb.org/). К настоящему времени (начало декабря 2009 г.) по поводу этого проекта опубликованы две статьи: «Requirements for Science Data Bases and SciDB» [50] и «A Demonstration of SciDB: A Science-Oriented DBMS» [51]. На русский язык эти статьи не переводились.

В число основных проектировщиков SciDB, помимо Стоунбрейкера и Девитта, входят, в частности, такие известные в мире баз данных люди, как Сэм Мэдден (Sam Madden), Дэвид Дайер (David Maier), Дженнифер Вайдом (Jennifer Widom) и Стэн Здоник (Stan Zdonik). Разработчиков пока на вид меньше, чем проектировщиков, но среди них российские программисты Павел Велихов и Роман Симаков. Проект выполняется в стиле open source (хотя никаких исходных текстов на сайте проекта пока нет) и рассчитан на два года. Проект поддерживается спонсорами, включая компании Vertica и eBay. Других источников финансирования, похоже, пока нет, хотя в начале проекта говорилось о возможной финансовой поддержке со стороны National Science Foundation.

Как отмечалось в разд. 2, проекту SciDB предшествовал университетский проект ASAP, из которого заимствуются многие идеи. Среди основных характеристик ожидаемой системы на текущий момент можно выделить следующее:

- Используется модель данных, основанная на популярных среди ученых разных специальностей вложенных многомерных массивах.
- Поддерживаются примитивные операции, ориентированные на научные расчеты, такие как смещение координатной сетки.
- Для всех хранимых данных обеспечивается информация об их происхождении, т.е. из какого источника данных они взяты.
- Обеспечивается возможность хранения, выборки и обработки неточных данных.
- Имеется возможность обработки данных без их загрузки в базу данных.

Как видно, проект SciDB вполне соответствует идеям Стоунбрейкера: быстро реализуется специализированная система. Предполагается возможность использования SciDB в облачной инфраструктуре. Насколько этот проект

будет успешным, покажет ближайший год (второй, завершающий год выполнения проекта).

8. Заключение

Как показывает 2009 г., хотим мы этого или не хотим, в мире баз данных происходят серьезные изменения. В той или иной степени все наиболее интересные, на мой взгляд, события 2009-го года связаны с идеями Майкла Стоунбрейкера и Клермонтского отчета, опубликованными в 2007-2008 гг. Специализация систем управления данными, поиск их новых архитектур, более приспособленных к текущим реалиям, становится относительной нормой.

Однако нужно учитывать, что 2009-й год был кризисным. Известно, что кризис – это хорошее время для новых начинаний и плохое время от зрелого бизнеса. Поэтому мы еще не услышали ответ на новые инициативы в области управления данными со стороны компаний, являющихся основными производителями СУБД. Будет ли этот ответ, и каким он будет, покажет ближайшее будущее.

В любом случае, мы живем в очень интересное время, время перемен в области управления данными. Займет ли новый мир место старого, привычного и для многих вполне удобного мира, или же два мира будут мирно существовать, покажет наступающее новое десятилетие.

Литература

[1] A Conversation with Michael Stonebraker and Margo Seltzer (http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=489), ACM Queue, Volume 5, Number 4, May/June 2007. Перевод: Беседа Марго Зельцер с Майклом Стоунбрейкером (http://citcity.ru/16453/).

[2] Michael Stonebraker, Uğur Çetintemel. «One Size Fits All»: An Idea Whose Time Has Come and Gone (http://www.cs.brown.edu/~ugur/fits_all.pdf). Перевод: Майкл Стоунбрейкер, Угур Кетинтемел. «Один размер пригоден для всех»: идея, время которой пришло и ушло (http://citforum.ru/database/articles/one_size_fits_all/).

[3] Michael Stonebraker, Chuck Bear, Uğur Çetintemel, Mitch Cherniack, Tingjian Ge, Nabil Hachem, Stavros Harizopoulos, John Lifter, Jennie Rogers, and Stan Zdonik. One Size Fits All? – Part 2: Benchmarking Results (http://nms.csail.mit.edu/~stavros/pubs/osfa.pdf). Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR), January 7-10, 2007, Asilomar, California, USA. Перевод: Майкл Стоунбрейкер, Чак Беэ, Угур Кетинтемел, Мич Черняк, Тиньян Ге, Набил Хачем, Ставрос Харизопулос, Джон Лифтер, Дженни Роджерс, Стэн Здоник. Пригоден ли один размер для всех? Часть 2: результаты тестовых испытаний (http://citforum.ru/database/articles/one_size_fits_all_2/).

[4] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, Pat Helland. The End of an Architectural Era (It's Time for a Complete Rewrite) (http://citforum.ru/database/articles/end_of_arch_era/). Proceedings of VLDB, 2007, Vienna, Austria. Перевод: Майкл Стоунбрейкер, Сэмюэль Мэдден, Дэниэль

Абади, Ставрос Харизопулос, Набил Хачем, Пат Хеллэнд. Конец архитектурной эпохи, или Наступило время полностью переписывать системы управления данными (http://citforum.ru/database/articles/end_of_arch_era/).

[5] Сергей Кузнецов. Универсальность и специализация: время разбивать камни? (http://citforum.ru/database/articles/time_to_break_stones/).

[6] StreamBase (http://www.streambase.com/).

[7] Streaminsight (http://www.microsoft.com/sqlserver/2008/en/us/R2-complex-event.aspx)

[8] Vertica (http://www.vertica.com/).

[9] H-Store (http://db.cs.yale.edu/hstore/).

[10] SciDB (http://scidb.org/).

[11] Марк Ривкин. Тенденции развития универсальных коммерческих СУБД. (http://citforum.ru/database/articles/trends/).

[12] Future Directions in DBMS Research – The Laguna Beach Participants (http://www.icsi.berkeley.edu/pubs/techreports/tr-88-001.pdf). SIGMOD Record 18(1): 17-26, 1989. Пересказ на русском языке: Будущие направления исследований в области баз данных: десять лет спустя (http://www.citforum.ru/database/articles/future_01.shtml).

[13] Abraham Silberschatz, Michael Stonebraker, and Jeffrey D. Ullman. Database Systems: Achievements and Opportunities (http://infolab.stanford.edu/~hector/lagi.ps). CACM 34(10): 110-120, 1991.

[14] Abraham Silberschatz, Michael Stonebraker, Jeffrey D. Ullman. Database Research: Achievements and Opportunities Into the 21st Century (http://www.cs.rmit.edu.au/~zahirt/Teaching/oodb/Papers/view.ps). SIGMOD Record 25(1): 52-63 (1996). Перевод: Базы данных: достижения и перспективы на пороге 21-го столетия (http://citforum.ru/database/classics/nfs_report/).

[15] Avi Silberschatz, Stan Zdonik et al., Strategic Directions in Database Systems – Breaking Out of the Box (http://www.cs.rmit.edu.au/~zahirt/Teaching/oodb/Papers/view.ps). *ACM Computing Surveys*, Vol. 28, No. 4 (Dec 1996), 764-778. Перевод: Стратегические направления в системах баз данных (http://citforum.ru/database/classics/nsf_report2/).

[16] The Asilomar Report on Database Research (http://www.sigmod.org/record/issues/9812/asilomar.html). SIGMOD Record 27(4): 74-80, 1998. Перевод: Асиломарский отчет об исследованиях в области баз данных (http://citforum.ru/database/digest/asil_01.shtml).

[17] The Lowell Database Research Self-Assessment (http://research.microsoft.com/~Gray/Lowell/). CACM 48(5): 111-118, 2005. Пересказ на русском языке: Крупные проблемы и текущие задачи исследований в области баз данных (http://www.citforum.ru/database/articles/problems/).

[18] The Claremont Report on Database Research (http://db.cs.berkeley.edu/claremont/claremontreport08.pdf). Перевод: Клермонтский отчет об исследованиях в области баз данных (http://citforum.ru/database/articles/claremont_report/).

[19] Goetz Graefe. The Five-minute Rule: 20 Years Later and How Flash Memory Changes the Rules (http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=549),

ACM QUEUE, July/August 2008. Перевод: Гоц Грейф «Правило пяти минут двадцать лет спустя, и как флэш-память изменяет правила» (http://www.citforum.ru/database/articles/five_minute_rule/

[20] LINQ (http://msdn.microsoft.com/en-us/netframework/aa904594.aspx)

[21] Ruby on Rails (http://www.rubyonrails.ru/)

[22] salesforce.com (http://www.salesforce.com/)

[23] Amazon S3 (http://aws.amazon.com/s3)

[24] Google App Engine (http://code.google.com/appengine/)

[25] Amazon EC2 (http://aws.amazon.com/ec2)

[26] Amazon SimpleDB (http://www.amazon.com/SimpleDB-AWS-Service-Pricing/b?ie=UTF8&node=342335011)

[27] Microsoft SQL Server Data Services (http://www.microsoft.com/sql/dataservices/default.mspx)

[28] Google's Datastore (http://www.groovie.org/2008/04/13/google-datastore-and-the-shift-from-a-rdbms)

[29] Jeffrey Dean, Sanjay Ghemawat «MapReduce: Simplifed Data Processing on Large Clusters» (http://labs.google.com/papers/mapreduce-osdi04.pdf), Proceedings of the Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004

[30] MapReduce в Hadoop (http://hadoop.apache.org/mapreduce/)

[31] Database Column (http://databasecolumn.vertica.com/)

[32] Michael Stonebraker, David J. DeWitt. MapReduce: A major step backwards (http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/)

[33] Michael Stonebraker, David J. DeWitt. MapReduce II (http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/)

[34] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis (http://cs-www.cs.yale.edu/homes/dna/papers/benchmarks-sigmod09.pdf). Proceedings of the 35th SIGMOD International Conference on Management of Data, 2009, Providence, Rhode Island, USA. Перевод: Эндрю Павло, Эрик Паулсон, Александр Разин, Дэниэль Абади, Дэвид Девитт, Сэмюэль Мэдден, Майкл Стоунбрейкер. Сравнение подходов к крупномасштабному анализу данных (http://citforum.ru/database/articles/mr_vs_dbms/)

[35] Greenplum (http://www.greenplum.com/)

[36] Eric Friedman, Peter Pawlowski, John Cieslewicz «SQL/MapReduce: A practical approach to selfdescribing, polymorphic, and parallelizable userdefined functions» (http://www.asterdata.com/resources/downloads/whitepapers/sqlmr.pdf), Proceeding of the VLDB '09 August 2428, 2009, Lyon, France

[37] MapReduce в Vertica (http://www.vertica.com/company/news/vertica-analytic-database-broadens-reach-with-flexstore)

[38] SQL Azur (http://www.microsoft.com/windowsazure/sqlazure/)

[39] SQL Server 2008 R2 Parallel Data Warehouse (http://www.microsoft.com/sqlserver/2008/en/us/parallel-data-warehouse.aspx)

[40] Daniela Florescu, Donald Kossmann. Rethinking Cost and Performance of Database Systems (http://www.sigmod.org/sigmod/record/issues/0903/p43.articles.florescu.pdf). SIGMOD Record, Vol. 38, No. 1, March 2009. Перевод: Переосмысление стоимости и производительности систем баз данных (http://citforum.ru/database/articles/rethinking/)

[41] 28msec (http://www.28msec.com/)

[42] Martin Kaufmann, Donald Kossmann. «Developing an Enterprise Web Application in XQuery» (http://www.28msec.com/download/enterprise_webapps.pdf)

[43] Jim Gray. (http://en.wikipedia.org/wiki/Jim_Gray_%28computer_scientist%29)

[44] Adam Jacobs. The Pathologies of Big Data (http://queue.acm.org/detail.cfm?id=1563874). ACM Queue, Vol. 7, Issue 6, July 2009. Перевод: Адам Якобса «Патологии больших данных» (http://citforum.ru/database/articles/pathology/)

[45] Сергей Кузнецов. О точности диагностики патологий (http://citforum.ru/database/articles/pathology_diagnostics/

[46] Tenbase (http://www.1010data.com/solutions.dw.tenbase.html)

[47] 1010data Inc. (http://www.1010data.com/)

[48] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, Caleb Welton. MAD Skills: New Analysis Practices for Big Data (http://db.cs.berkeley.edu/jmh/papers/madskills-032009.pdf). Proceedings of the VLDB'09 Conference, Lyon, France, August 24-28, 2009). Пер.

[49] Перевод: Джеффри Коэн, Брайен Долэн, Марк Данлэп, Джозеф Хеллерстейн и Кейлэба Велтон «МОГучие способности: новые приемы анализа больших данных» (http://citforum.ru/database/articles/mad_skills/)

[50] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, Gerd Heber. Scientific Data Management in the Coming Decade (http://www.sigmod.org/sigmod/record/issues/0512/p34-article-gray.pdf), SIGMOD Record, Vol. 34, No. 4, Dec. 2005. Перевод: Управление научными данными в следующем десятилетии (http://www.citforum.ru/database/articles/scientific_data/

[51] 50. Requirements for Science Data Bases and SciDB (http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_26.pdf). Proceedings of the Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009

[52] A Demonstration of SciDB: A Science-Oriented DBMS (http://scidb.org/Documents/SciDB-VLDB09-paper.pdf). Proceedings of the VLDB ‘09, August 2428, 2009, Lyon, France

MapReduce: внутри, снаружи или сбоку от параллельных СУБД?

Кузнецов С.Д.
kuzloc@ispras.ru

Аннотация. Обсуждаются подходы к использованию технологии MapReduce в аналитических СУБД. Рассматриваются подходы, при которых MapReduce реализуется внутри ядра параллельной СУБД, используется в качестве коммуникационной инфраструктуры новой параллельной СУБД и применяется автономно в симбиотическом единстве с параллельной СУБД. В качестве примера применения первого подхода анализируются особенности организации массивно-параллельных СУБД Greenplum Database и nCluster компаний Greenplum и Aster Data Systems соответственно. Второй подход применяется в проекте HadoopDB университетов Yale и Brown. Наконец, третий подход развивается в компании Vertica.
Ключевые слова. Массивно-параллельные аналитические СУБД, MapReduce

1. Введение

Как отмечалось в Клермонтском отчете [1], "*... сбор, интеграция и анализ данных больше не считаются расходами на ведение бизнеса; данные – это ключ к достижению эффективности и прибыльности бизнеса. В результате быстро развивается индустрия, поддерживающая анализ данных*". Если к концу прошлого века программные средства, пригодные для организации хранилищ данных и выполнения над ними оперативного анализа, можно было пересчитать по пальцам одной руки (IBM DB2, Teradata, Sybase IQ, Oracle, частично Microsoft SQL Server, причем только в DB2 и Teradata поддерживалась массивно параллельная архитектура без общих ресурсов между узлами (sharing nothing) и только в Sybase IQ использовалось поколоночное хранение таблиц (column-based store)), то с начала нового тысячелетия активизировалось направление специализированных аппаратно-программных систем, полностью ориентированных на поддержку хранилищ данных и/или анализа данных (Data Warehouse Appliance или Analytic Appliance; в дальнейшем для соблюдения точности и для краткости я буду обозначать это направление и относящиеся к нему системы аббревиатурой DWAA). Основной целью этого направления являлось и является создание аппаратно-программных средств, которые были бы существенно дешевле средств поддержки хранилищ данных, предлагаемых поставщиками универсальных СУБД, но при этом обеспечивали бы не меньшую, а

желательно, бо́льшую производительность и масштабируемость при работе со сверхбольшими хранилищами данных.

1.1. Аналитические параллельные СУБД сегодня

Как отмечается в [2], в действительности направление DWAA появилось еще в 1980-е гг., и соответствующие пионерские продукты были созданы в компании Britton Lee Inc. [3], которая в 1989 г. была сначала переименована в ShareBase Corporation, а затем поглощена компанией Teradata [4], которая к этому времени тоже придерживалась подхода DWAA. Аппаратно-программное решение, основанное на ассоциативной адресации элементов хранения данных, имелось у компании ICL (Content Addressable File Store [5]). Однако на рынке систем поддержки хранилищ данных на основе подхода DWAA с тех пор осталась только Teradata.

Возрождение направления DWAA в начале 2000-х, безусловно, связано с упомянутым выше ростом заинтересованности компаний в сравнительно недорогих и эффективных решениях, направленных исключительно на поддержку хранилищ данных и их анализа. Вокруг этого направления стали возникать софтверные стартапы, первым из которых стала компания Netezza [6], основавшая свое эффективное DWAA-решение на использовании программируемых вентильных матриц (Field Programmable Gate Array, FPGA) и процессоров PowerPC. Использование FPGA в контроллерах магнитных дисков позволяет осуществлять "на лету" первичную фильтрацию данных, а применение PowerPC вместо процессоров Intel (по утверждению компании) позволяет снизить энергопотребление и расходы на охлаждение.

С тех пор появилось еще около десяти новых компаний, ориентирующихся на разработку DWAA с применением (почти всегда) разновидностей массивно-параллельной архитектуры (MPP) "sharing-nothing":

- Vertica Systems [7] – MPP, поколоночное хранение таблиц;
- DATAllegro Inc. [8], недавно поглощенная Microsoft, которая основала на продукте этой компании проект Madison, ставший основой SQL Server 2008 R2 Parallel Data Warehouse [15], – MPP, система основана на использовании СУБД Ingres [16] (тем самым, таблицы хранятся по строкам);
- Greenplum [9] – MPP, система основана на использовании СУБД PostgreSQL [17] (тем самым, таблицы хранятся по строкам);
- Aster Data Systems [10] – MPP, таблицы хранятся по строкам;
- Kognitio [11] – MPP, таблицы хранятся по строкам;
- EXASOL AG [12] – MPP, поколоночное хранение таблиц;
- Calpont Corporation [13] – MPP, поколоночное хранение таблиц, система (InfiniDB) внешне схожа с MySQL;
- Dataupia Corporation [14] – MPP, таблицы хранятся по строкам;

- Infobright [15] – поколоночное хранение таблиц, система основана на MySQL, ориентирована на использование многоядерных процессоров, массивный параллелизм не используется;
- Kickfire [16] – поколоночное хранение таблиц, используется специальная аппаратура, ускоряющая выполнение SQL-запросов, система создана на основе MySQL и не основана на массивно-параллельной архитектуре.

Подход DWAA постепенно проникает и в продукты основных поставщиков SQL-ориентированных СУБД. Как отмечалось выше, разработка компании DATAllegro стала основой массивно-параллельного варианта Microsoft SQL Server (SQL Server 2008 R2 Parallel Data Warehouse), а компания Oracle обеспечивает специализированное массивно-параллельное хранилище табличных данных Oracle Exadata Storage Server [21], позволяющее значительно ускорить работу основной СУБД.

У разных решений категории DWAA имеются свои интересные технические особенности, заслуживающие более глубокого обсуждения, анализа и сравнения. Их можно классифицировать и сравнивать по разным критериям. Однако это не является целью данной статьи. Некоторую попытку такого анализа представляет собой обзор [22]. Значительный рост интереса к направлению DWAA, к специализированным СУБД вообще и к СУБД Vertica в частности вызвала статья [23].

1.2. При чем здесь MapReduce?

В этой своей статье я сосредоточусь на частном, но очень важном в настоящее время вопросе взаимоотношений технологий массивно-параллельных аналитических СУБД и MapReduce [24]. При рассмотрении этого вопроса контекст DWAA является вполне естественным, поскольку практически все СУБД, созданные на основе подхода DWAA, являются массивно-параллельными без использования общих ресурсов. Эти системы создавались в расчете на использование в кластерной аппаратной архитектуре, и они сравнительно легко могут быть перенесены в "облачную" среду динамически конфигурируемых кластеров.

Поэтому появление "родной" для "облачной" среды технологии MapReduce и в особенности энтузиазм по части ее использования, проявленный многими потенциальными пользователями параллельных СУБД, очень озаботили представителей направления DWAA. Сначала авторитетные представители сообщества баз данных и одновременно активные сторонники подхода DWAA Майкл Стоунбрейкер (Michael Stonebraker) и Дэвид Девитт (David J. DeWitt) старались убедить общественность в том, что MapReduce – это технология, уступающая технологии параллельных баз данных по всем статьям [25-26]. Потом была проведена серия экспериментов, продемонстрировавшая, что при решении типичных простых аналитических задач MapReduce уступает в производительности не только поколоночной СУБД Vertica, но и

традиционной массивно-параллельной СУБД с хранением таблиц по строкам [27].

Все приводимые доводы и результаты экспериментов были весьма солидными и убедительными, и вряд ли кто-нибудь из людей, знакомых с обеими технологиями, сомневается в том, что MapReduce не вытеснит параллельные СУБД, и что эти технологии будут благополучно сосуществовать в "облаках" и в среде кластерных архитектур вообще. Однако возникает другой вопрос: а нет ли в технологии MapReduce каких-либо положительных черт, которых не хватает параллельным СУБД? И можно ли каким-либо образом добавить эти черты в параллельные СУБД, сохранив их основные качества: декларативный доступ на языке SQL, оптимизацию запросов и т.д. (Кстати, понятно, что у параллельных СУБД имеется масса положительных черт, которыми не обладает MapReduce, но похоже, что добавление их к MapReduce изменило бы суть этой технологии, превратив ее в технологию параллельных СУБД.)

И на эти два вопроса удалось получить положительный ответ. В нескольких проектах, связанных с направлением DWAA, удалось воспользоваться такими преимуществами MapReduce, как масштабируемость до десятков тысяч узлов, отказоустойчивость, дешевизна загрузки данных, возможность использования явно написанного кода, который хорошо распараллеливается. Сразу следует заметить, что пока ни в одном проекте не удалось воспользоваться сразу всеми этими преимуществами, но даже то, чего уже достигли исследователи и разработчики, позволяет добавить в параллельные СУБД важные качества, которыми они до сих по не обладали.

Мы рассмотрим три подхода к интеграции технологий MapReduce и параллельных СУБД, предложенных и реализованных специалистами компаний Greenplum [28] и Aster Data [29], университетов Yale и Brown [30], а также компании Vertica [31] соответственно, которые можно было бы назвать:

- MapReduce внутри параллельной СУБД;
- СУБД внутри среды MapReduce и
- MapReduce сбоку от параллельной СУБД.

В общих словах, первый подход ориентирован на поддержку написания и выполнения хранимых на стороне сервера баз данных пользовательских функций, которые хорошо распараллеливаются в кластерной (в том числе, в "облачной") среде. Т.е. в данном случае используется преимущество MapReduce по применению явно написанного кода и его распараллеливанию.

Второй подход направлен на использование MapReduce в качестве инфраструктуры параллельной СУБД, в качестве базовых компонентов которой используются традиционные не параллельные СУБД. Применение MapReduce позволяет добиться неограниченной масштабируемости получаемой системы и ее отказоустойчивости на уровне выполнения запросов.

Наконец, при применении третьего подхода MapReduce используется для выполнения процедуры ETL (Extract, Tansform, Load) над исходными данными до их загрузки в систему параллельных баз данных. В этом случае используется преимущество MapReduce в отношении дешевой загрузки данных до их обработки.

Основные разделы статьи организованы следующим образом. В разд. 2 приводится общий обзор технологии MapReduce. Разд. 3 посвящается обсуждению деталей интеграции технологий MapReduce и параллельных СУБД путем встраивания средств MapReduce в СУБД. В разд. 4 описываются основные приемы, используемые для создания параллельной СУБД на основе MapReduce. В разд. 5 обсуждаются проблемы загрузки данных в аналитические базы данных и преимущества, которые можно получить при выполнении процедуры ETL на основе MapReduce. Разд. 6 содержит заключение.

2. MapReduce: модель и реализации

Программная модель MapReduce была придумана несколько лет тому назад в компании Google [24], и там же была выполнена первая реализация этой модели на основе распределенной файловой системы той же компании GFS (Google File System) [32]. Эта реализация активно используется в программных продуктах самой Google, но является сугубо проприетарной и недоступна для использования вне Google.

Альтернативная, свободно доступная реализация Hadoop MapReduce [33] (с открытыми исходными текстами) была выполнена в проекте Hadoop [34] сообщества Apache [35]. Она основана на использовании распределенной файловой системы HDFS (Hadoop Distributed File System) [36], также разработанной в проекте Hadoop. Реальную популярность MapReduce принесла именно реализация Hadoop в силу своей доступности и открытости, а широкое использование Hadoop MapReduce в различных исследовательских и исследовательских проектах приносит несомненную пользу этой системе, стимулируя разработчиков к ее постоянному совершенствованию.

Однако реализация Hadoop MapReduce полностью основана на спецификациях Google, и поэтому каноническим описанием технологии была и остается статья [24]. Заметим, что при этом в документации Hadoop MapReduce [37] используется терминология, несколько отличная от [24]. В этом разделе из уважения к первенству Google я буду использовать термины из [24], а в следующих разделах там, где будет иметься конкретно реализация Hadoop MapReduce, будет использоваться терминология Hadoop (это не должно привести к путанице).

2.1. Общая модель программирования MapReduce

В этой модели вычисления производятся над множествами входных пар "ключ-значение", и в результате каждого вычисления также производится некоторое множество результирующих пар "ключ-значение". Для представления вычислений в среде MapReduce используются две основные функции: *Map* и *Reduce*. Обе функции явно кодируются разрабочиками приложений в среде MapReduce.

Функция *Map* в цикле обрабатывает каждую пару из множества входных пар и производит множество промежуточных пар "ключ-значение". Среда MapReduce групирует все промежуточные значения с одним и тем же ключом *I* и передает их функции *Reduce*.

Функция *Reduce* получает значение ключа *I* и множество значений, связанных с этим ключом. В типичных ситуациях каждая группа обрабатывается (в цикле) таким образом, что в результате одного вызова функции образуется не более одного результирующего значения.

2.2. Реализация в распределенной среде

Реализации MapReduce от Google и Hadoop ориентированы на использование в кластерной распределенной среде со следующими основными характеристиками:

- узлы среды выполнения MR-приложений обычно представляют собой компьютеры общего назначения с операционной системой Linux;
- используется стандартное сетевое оборудование с адаптерами, расчитанными на скорости передачи в 100 мегабит в секунду или 1 гигабит в секунду, но средняя пропускная способность существенно ниже;
- кластер состоит из сотен или тысяч машин, так что вполне вероятны отказы отдельных узлов;
- для хранения данных используются недорогие дисковые устройства, подключенные напрямую к отдельным машинам;
- для управления данными, хранящимися на этих дисках, используется распределенная файловая система;
- пользователи представляют свои задания в систему планирования; каждое задание состоит из некоторого набора задач, которые отображаются планировщиком на некоторый набор узлов кластера.

2.2.1. Выполнение MR-приложения

Вызовы *Map* распределяются по нескольким узлам кластера путем разделения входных данных на *M* непересекающихся групп (split). Входные группы могут параллельно обрабатываться на разных машинах. Вызовы *Reduce* распределяются путем разделения пространства ключей промежуточных ключей на *R* частей с использованием некоторой функции разделения

(например, функции хэширования). Число разделов *R* и функция разделения задаются пользователем.

Выполнение MR-программы происходит следующим образом. Сначала среда MapReduce расщепляет входной файл на *M* частей, размер которых может задаваться пользователем. Затем сразу в нескольких узлах кластера запускается основная программа MapReduce. Один из экземпляров этой программы играет специальную роль и называется *распорядителем (master)*. Остальные экземпляры являются *исполнителями (worker)*, которым распорядитель назначает работу. Распорядитель должен назначить исполнителям для выполнения *M* задач *Map* и *R* задач *Reduce*.

Исполнитель, которому назначена задача *Map*, читает содержимое соответствующей группы, разбирает пары "ключ-значение" входных данных и передает каждую пару в определенную пользователем функцию *Map*. Промежуточные пары "ключ-значение", производимые функцией *Map*, буферизуются в основной памяти. Периодически буферизованные пары, разделяемые на *R* областей на основе функции разделения, записываются в локальную дисковую память исполнителя. Координаты этих сохраненных на диске буферизованных пар отсылаются распорядителю, который, в свою очередь, передает эти координаты исполнителям задачи *Reduce*. *I*-ый *Reduce*-исполнитель снабжается координатами всех *i*-ых областей буферизованных пар, произведенных всеми *M Map*-исполнителями.

После получения этих координат от распорядителя исполнитель задачи *Reduce* с использованием механизма удаленных вызовов процедур переписывает данные с локальных дисков исполнителей задачи *Map* в свою память или на локальный диск (в зависимости от объема данных). После переписи всех промежуточных данных выполняется их сортировка по значениям промежуточного ключа для образования групп с одинаковым значением ключа. Если объем промежуточных данных слишком велик для выполнения сортировки в основной памяти, используется внешняя сортировка.

Далее *Reduce*-исполнитель организует цикл по отсортированным промежуточным данным и для каждого уникального значения ключа вызывает пользовательскую функцию *Reduce* с передачей ей в качестве аргумента значения ключа и соответствущее множество значений. Результирующие пары функции *Reduce* добавляются в окончательный результирующий файл данного *Reduce*-исполнителя. После завершения всех задач *Map* и *Reduce* распорядитель активизирует программу пользователя, вызывавшую MapReduce.

После успешного завершения выполнения задания MapReduce результаты размещаются в *R* файлах распределенной файловой системы (имена этих результирующих файлов задаются пользователем). Обычно не требуется объединять их в один файл, потому что часто полученные файлы используются в качестве входных для запуска следующего MR-задания или в

каком-либо другом распределенном приложении, которое может получать входные данные из нескольких файлов.

2.2.2. Отказоустойчивость

Поскольку технология MapReduce предназначена для обработки громадных объемов данных с использованием сотен и тысяч машин, в ней обязательна должна присутствовать устойчивость к отказам отдельных машин.

2.2.2.1. Отказ исполнителя

Распорядитель периодически посылает каждому исполнителю контрольные сообщения. Если некоторый исполнитель не отвечает на такое сообщение в течение некоторого установленного времени, распорядитель считает его вышедшим из строя. В этом случае все задачи *Map*, уже выполненные и еще выполнявшиеся этим исполнителем, переводятся в свое исходное состояние, и можно заново планировать их выполнение другими исполнителями. Аналогично распорядитель поступает со всеми задачами *Reduce*, выполнявшимися отказавшим исполнителем к моменту отказа.

Завершившиеся задачи *Map* выполняются повторно по той причине, что их результирующие пары сохранялись на локальном диске отказавшего исполнителя и поэтому недоступны в других узлах. Завершившиеся задачи *Reduce* повторно выполнять не требуется, поскольку их результирующие пары сохраняются в глобальной распределенной файловой системе. Если некоторая задача *Map* выполнялась исполнителем *A*, а потом выполняется исполнителем *B*, то об этом факте оповещаются все исполнители, выполняющие задачи *Reduce*. Любая задача *Reduce*, которая не успела прочитать данные, произведенные исполнителем *A*, после этого будет читать данные от исполнителя *B*.

2.2.2.2. Отказ распорядителя

В реализациях MapReduce от Google и Hadoop какая-либо репликация распорядителя не производится. Считается, что поскольку распорядитель выполняется только в одном узле кластера, его отказ маловероятен, и если он случается, то аварийно завершается все выполнение MapReduce. Однако в [24] отмечается, что несложно организовать периодический сброс в распределенную файловую систему всего состояния распорядителя, чтобы в случае отказа можно было запустить его новый экземпляр в другом узле с данной контрольной точки.

2.2.2.3. Семантика при наличии отказов

Если обеспечиваемые пользователями функции *Map* и *Reduce* являются детерминированными (т.е. всегда выдают одни и те же результаты при одинаковых входных данных), то при их выполнении в среде распределенной реализации MapReduce при любых условиях обеспечивает тот же результат,

как при последовательном выполнении всей программы при отсутствии каких-либо сбоев.

Это свойство обеспечивается за счет атомарности фиксации результатов задач *Map* и *Reduce*. Каждая выполняемая задача записывает свои результаты в частные временные файлы. Задача *Reduce* производит один такой файл, а задача *Map* – *R* файлов, по одной на каждую задачу *Reduce*. По завершении задачи *Map* исполнитель посылает распорядителю сообщение, в котором указываются имена *R* временных файлов. При получении такого сообщения распорядитель запоминает эти имена файлов в своих структурах данных. Повторные сообщения о завершении одной и той же задачи *Map* игнорируются.

При завершении задачи *Reduce* ее исполнитель атомарным образом переименовывает временный файл результатов в окончательный файл. Если одна и та же задача *Reduce* выполняется несколькими исполнителями, то для одного и того же окончательного файла будет выполнено несколько операций переименования. Если в используемой распределенной файловой системе операция переименования является атомарной, то в результате в файловой системе соохранятся результаты только какого-либо одного выполнения задачи *Reduce*.

2.2.3. Резервные задачи

Чаще всего к увеличению общего времени выполнения задания MapReduce приводит наличие "отстающих" ("straggler") – узлов кластера, в которых выполнение одной из последних задач *Map* или *Reduce* занимает необычно долгое время (например, из-за некритичной неисправности дискового устройства).

Для смягчения проблемы "остающих" в MapReduce применяется следующий общий механизм. Когда задание близится к завершению, для всех еще не завершившихся задач назначаются дополнительные, резервные исполнители. Задача считается выполненной, когда завершается ее первичное или резервное выполнение. Этот мезанизм настраивается таким образом, чтобы потребление вычислительных ресурсов возрастало не более чем на несколько процентов. В результате удается существенно сократить время выполнения крупных MR-заданий.

2.3. Расширенные средства

В большинстве приложений MapReduce оказывается достаточно просто написать свои функции *Map* и *Reduce*. Однако в ряде случаев оказываются полезными некоторые расширения базовых функциональных возможностей.

2.3.1. Функция разделения

Пользователи MapReduce явно указывают требуемое число задач *Reduce R* (и соответствующих результирующих файлов). Данные распределяются между

этими задачами с использованием некоторой функции разделения от значений промежуточного ключа. По умолчанию используется функция хэширования (например, "*hash*(*key*) mod *R*"). Однако пользователи MapReduce могут специфицировать и собственные функции разделения.

2.3.2. Гарантии упорядоченности

Внутри каждого раздела, передаваемого задаче *Reduce*, данные упорядочены по возрастанию значений промежуточного ключа. Это позволяет задачам *Reduce* производить отсортированные результирующие файлы.

2.3.3. Функция-комбинатор

В некоторых случаях в результатах задачи *Map* содержится значительное число повторящихся значений промежуточного ключа, а определенная пользователем задача *Reduce* является коммутативной и ассоциативной. В таких случаях пользователь может определить дополнительную функцию-комбинатор (*Combiner*), выполняющую частичную агрегацию таких данных до их передачи по сети. Функция *Combiner* выполняется на той же машине, что и задача *Map*. Обычно для ее реализации используется тот же самый код, что и для реализации функции *Reduce*. Елинственное различие между функциями *Combiner* и *Reduce* состоит в способе работы с их результирующими данными. Результаты функции *Reduce* записываются в окончательный файл результатов. Результаты же функции *Combiner* помещаются в промежуточные файлы, которые впоследствии пересылаются в задачи *Reduce*.

2.3.4. Форматы входных и результирующих данных

В библиотеке MapReduce поддерживается возможность чтения входных данных в нескольких разных форматах. Например, в режиме "text" каждая строка трактуется как пара "ключ-значение", где ключ – это смещение до данной строки от начала файла, а значение – содержимое строки. В другом распространенном формате входные данные представляются в виде пар "ключ-значение", отсортированных по значениям ключа. В каждой реализации формата входных данных известно, каким образом следует расшеплять данные на осмысленные части, которые обрабатываются отдельными задачами *Map* (например, данные формата "text" расщепляются только по границами строк).

Пользователи могут добавить к реализации собственные форматы входных данных, обеспечив новую реализацию интерфейса *reader* (в реализации Hadoop – *RecordReader*). *Reader* не обязательно должен читать данные из файла, можно легко определить *reader*, читающий данные из базы данных или из некоторой структуры в виртуальной памяти.

Аналогичным образом, поддерживаются возможности генерации данных в разных форматах, и имеется простая возможность определения новых форматов результирующих данных.

Думаю, что для общего ознакомления с технологией MapReduce и для понимания следующих разделов статьи этой информации достаточно. Кроме того, она позволяет понять, какие особенности модели и реализаций MapReduce обеспечивают масштабируемость технологии до десятков тысяч узлов, ее отказоустойчивость, дешевизну загрузки данных и возможность использования явно написанного кода, который хорошо распараллеливается.

3. MapReduce внутри параллельной СУБД

Очевидны преимущества клиент-серверных организаций СУБД: в такой архитектуре сервер баз данных поддерживает крупную базу данных, которая сохраняется в одном экземпляре и доступна большому числу приложений, выполняемых прямо на стороне клиентов или в промежуточных серверах приложений. Однако даже при использовании реляционной (или, правильнее, SQL-ориентированной) организации баз данных, когда от клиентов на сервер баз данных отправляются высокоуровневые декларативные запросы, в обратную сторону, от сервера к клиенту, пересылаются результирующие данные, вообще говоря, произвольно большого объема.

Естественно, возникает вопрос: не окажется ли дешевле, чем пересылать данные с сервера на клиент для их дальнейшей обработки, переместить требуемую обработку данных на сервер, ближе к самим данным. Насколько мне известно, в явном виде идея перемещения вычислений на сторону сервера была высказана в статье Лоуренса Роува (Lawrence A. Rowe) и Майкла Стоунбрейкера (Michael R. Stonebraker) [38], хотя в более скрытой форме намеки на эту идею можно найти и в более ранних статьях М. Стоунбрейкера и др. [39-40], еще не имевших непосредственного отношения к СУБД Postgres.

С того времени, как поддержка определяемых пользователями хранимых процедур, функций и методов, типов данных и триггеров появилась во всех развитых SQL-ориентированных СУБД, соответствующие языковые средства специфицированы в стандарте языка SQL. Более того, возникла новая проблема выбора – одну и ту же функциональность приложения можно реализовать на стороне сервера, на сервере приложений и на клиенте. Однозначных методологий и рекомендаций, способствующих простому выбору, не существует. Например, очевидно, что если услугами одного сервера пользуется несколько приложений, то перегрузка сервера хранимыми процедурами и функциями, реализующими функциональность одного приложения, может нанести ущерб эффективности других приложений.

Тем не менее, во всех традиционных серверных организациях СУБД возможность переноса вычислений на сторону сервера существует и не очень сложно реализуется. Однако в параллельных СУБД (в особенности, категории

sharing-nothing) дела обстоят гораздо хуже. Выполнение SQL-запросов распараллеливается автоматически оптимизитором запросов. Но оптимизатор запросов не может распараллелить определенную пользователем процедуру или функцию, если она написана не на SQL, а на одном из традиционных языков программирования (обычно с включением вызовов операторов SQL).

Конечно, технически можно было бы такие процедуры и функции вообще не распараллеливать, а выполнять в каком-либо одном узле кластера. Но тогда (a) в этом узле пришлось бы собрать все данные, требуемые для выполнения процедуры или функции, для чего потребовалась бы массовая пересылка данных по сети, и (b) это свело бы на нет все преимущества параллельных СУБД, производительность которых основывается именно на параллельном выполнении.

С другой стороны, невозможно обязать распараллеливать свои программы самих пользователей, определяющих хранимые процедуры или функции (например, на основе библиотеки MPI [41], которую принято использовать для явного параллельного программирования на основе обмена сообщениями в массивно-параллельной среде). Во-первых, это слишком сложное занятие для разработчиков приложений баз данных, которые часто вообще не являются профессиональными программистами. Во-вторых, при таком явном параллельном программировании требовалось бы явным же образом управлять распределением данных по узлам кластера.

Несмотря на эти трудности, какая-то поддержка механизма распараллеливаемых определяемых пользователями процедур и функций в параллельных аналитических системам баз данных все-таки требуется, поскольку без этого аналитики вынуждены выполнять анализ данных на клиентских рабочих станциях, постоянно пересылая на них из центрального хранилища данных данные весьма большого объема. Другого способа работы у них просто нет. И как показывает опыт двух производственных разработок, для обеспечения возможностей серверного программирования в массивно-параллельной среде систем баз данных с пользой может быть применена модель MapReduce.

Речь идет о параллельных аналитических СУБД Greenplum Database компании Greenplum [9] и *n*Cluster компании Aster Data Systems [10]. Общим в подходах обеих компаний является то, что модель MapReduce реализуется внутри СУБД, и возможностями этих реализаций могут пользоваться разработчики аналитических приложений. Различие состоит в том, как можно пользоваться возможностями MapReduce: в Greenplum Database – наряду с SQL, а в *n*Cluster – из SQL. Рассмотрим эти подходы подробнее.

3.1. Greenplum – MapReduce наравне с SQL

Сначала немного поговорим об общей философии компании Greenplum, приведшей ее, в частности, к идее поддержки технологии MapReduce наряду с технологией SQL. По мнению идеологов Greenplum и основных архитекторов

Greenplum Database [28], возрастающий уровень востребованности хранилищ данных и оперативного анализа данных, возможность и целесообразность использования требуемых аппаратных средств в масштабах отдельных подразделений компаний приводят к потребности пересмотра "ортодоксального" подхода к организации хранилищ данных.

3.1.1. MAD Skills: новый подход к организации хранилищ данных и аналитике

Предлагается и реализуется новый подход к анализу данных, который идеологи (и маркетологи!) компании связывают с аббревиатурой *MAD*. Здесь, конечно, имеется интересная игра слов, которую трудно выразить на русском языке. С одной стороны, *mad* применительно к технологии означает, что эта технология слегка безумна и уж во всяком случае не ортодоксальна. С другой стороны, *mad skills* означает *блестящие способности*, а значит, предлагаемая технология, по мнению ее творцов, обладает новыми полезнейшими качествами. Но в Greenplum *MAD* – это еще и аббревиатура от *magnetic*, *agile* и *deep*.

Magnetic (магнетичность) применительно к хранилищу данных означает, что оно должно быть "притягательным" по отношению к новым источникам данных, появляющимся в организации. Данные из новых источников должны легко и просто включаться в хранилище данных с пользой для аналитиков. В отличие от этого, при использовании традиционного ("ортодоксального") подхода к организации хранилища данных, для подключения нового источника данных требуется разработка и применение соответствующей процедуры ETL, а возможно, и изменение схемы хранилища данных, в результате чего подключение нового источника данных часто затягивается на месяца, а иногда и вовсе кончается ничем.

Agile (гибкость) – это предоставляемая аналитикам возможность простым образом и в быстром темпе воспринимать, классифицировать, производить и перерабатывать данные. Для этого требуется база данных, логическая и физическая структура и содержание которой могут постоянно и быстро изменяться. В отличие от этого, традиционным хранилищам данных свойственна жесткость, связанная с потребностью долгосрочного тщательного проектирования и планирования.

Deep (основательность) означает, что аналитикам должны предоставляться средства выполнения произвольно сложных статистических алгоритмов над всеми данными, находящимися в хранилище данных, без потребности во взятии образцов или выборок. Хранилище данных должно служить как основательным репозиторием данных, так и средой, поддерживающей выполнение сложных алгоритмов.

Я не буду больше распространяться здесь про MAD-аналитику (более развернутое обсуждение см. в [28], а остановлюсь только на том аспекте, который привел к реализации системы с поддержкой интерфейсов и SQL, и

MapReduce. Как считают разработчики Greenplum Database хозяевами будущего мира анализа данных должны стать аналитики. Фактически, на это направлены все аспекты MAD-аналитики. В частности, это означает всяческую поддержку написания и использования в среде хранилища данных разнообразных аналитических алгоритмов.

Как отмечалось в подразделе 1.1, параллельная СУБД Greenplum Database делалась на основе СУБД PostgreSQL, являющейся законной наследницей Postgres. Помимо своих прочих достоинств, Postgres была первой *расширяемой* СУБД (этот аспект системы впервые явно подчеркивался в [42]). Пользователи Postgres могли определять собственные процедуры и функции, типы данных и даже методы доступа к структурам внешней памяти. Эти возможности расширений системы были переняты и развиты в PostgreSQL. Наряду с традиционным в Postgres языком C, для программирования серверных расширений в PostgreSQL можно использовать, в частности, популярные скриптовые языки Perl и Python [43].

В Greenplum Database на основе этих возможностей расширений системы обеспечена расширенная среда, позволяющая на уровне языка SQL оперировать такими математическими объектами, как векторы, функции и функционалы. Пользователи могут определять собственные статистические алгоритмы и в полуавтоматическом режиме распараллеливать их выполнение по данным в массивно-параллельной среде (что часто является очень нетривиальной задачей). Однако в любом случае при использовании такого подхода к анализу данных пользователям-аналитикам приходится иметь дело с декларативным языком SQL, а как считают идеологи Greenplum, для многих аналитиков и статистиков SQL-программирование является обременительным и неудобным.

В качестве альтернативы аналитическому SQL-программированию в Greenplum Database обеспечивается полноправная реализация MapReduce, в которой обеспечивается доступ ко всем данным, сохраняемым в хранилище данных. При использовании MapReduce аналитики пишут собственный понятный для них процедурный код (можно использовать те же Perl и Python) и понимают, как будет выполняться их алгоритм в массивно-параллельной среде, поскольку это выполнение опирается на простую модель MapReduce.

3.1.2. Реализация MapReduce в Greenplum Database

Для иллюстрации общей организации Greenplum Database воспользуемся Рис. 1, позаимствованным из [44].

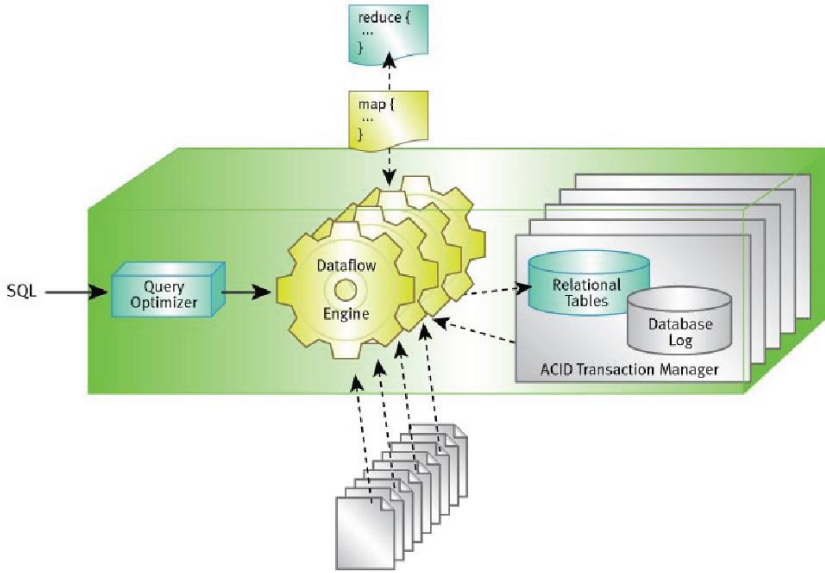


Рис. 1. Общая организация Greenplum Database

Как показывает этот рисунок, ядром системы является процессор потоков данных (Dataflow Engine). С его разработки началась реализация Greenplum Database, причем основные цели состояли в том, чтобы (a) заменить соответствующий компонент ядра PostgreSQL, чтобы обеспечить массивно-параллельное выполнение запросов и (b) обеспечить базовые функциональные возможности, требуемые для поддержки модели MapReduce. В результате SQL-ориентированная СУБД и MapReduce работают с общим ядром, поддерживающим массивно-параллельную обработку данных, и механизмы SQL и MapReduce обладают интероперабельностью.

Как отмечалось выше, функции *Map* и *Reduce* в среде Greenplum Database можно программировать на популярных скриптовых языках Python и Perl. В результате оказывается возможным использовать развитые программные средства с открытыми кодами, содержащиеся в репозиториях Python Package Index (PyPi) [45] и Comprehensive Perl Archive Network (CPAN) [46]. В составе этих репозиториев находятся средства анализа неструктурированного текста, статистические инструментальные средства, анализаторы форматов HTML и XML и многие другие программные средства, потенциально полезные аналитикам.

В среде Greenplum Database приложениям MapReduce обеспечивается доступ к данным, хранящимся в файлах, предоставляемым Web-сайтами и даже

генерируемым командами операционной системы. Доступ к таким данным не влечет накладных расходов, ассоциируемых с использованием СУБД: блокировок, журнализации, фиксации транзакций и т.д. С другой стороны, эффективный доступ к данным, хранимым в базе данных, поддерживается за счет выполнения MR-программ в ядре Greenplum Database. Это позволяет избежать расходов на пересылку данных.

Архитектура Greenplum Database с равноправной поддержкой SQL и MapReduce позволяет смешивать стили программирования, делать MR-программы видимыми для SQL-запросов и наоборот. Например, можно выполнять MR-программы над таблицами базы данных. Для этого всего лишь требуется указать MapReduce, что входные данные программы должны браться из таблицы. Поскольку таблицы баз данных Greenplum Database хранятся разделенными между несколькими узлами кластера, первая фаза *MAP* выполняется внутри ядра СУБД прямо над этими разделами.

Как и в автономных реализациях MapReduce, результаты выполнения MR-программ могут сохраняться в файловой системе. Но настолько же просто сохранить результирующие данные в базе данных с обеспечением транзакционной долговечности хранения этих данных (см. компонент ACID Transaction Manager на Рис. 1). В дальнейшем эти данные могут анализироваться, например, с применением SQL-запросов. Запись результирующих данных в таблицы происходит параллельным образом и не вызывает лишних накладных расходов.

Поскольку источником входных данных для MR-программы может служить любая таблица базы данных Greenplum, в частности, в качестве такой таблицы можно использовать представление базы данных, определенное средствами SQL. И наоборот, MR-программу можно зарегистрировать в базе данных как представление, к которому можно адресовать SQL-запросы. В этом случае MR-задание выполняется "на лету" во время обработки SQL-запроса, и результрующие данные в конвейерном режиме передаются прямо в план выполнения запроса.

3.2. Aster Data – MapReduce как основа нового механизма функций, определяемых пользователями

Как и у компании Greenplum с ее *MAD Skills*, у компании Aster Data имеется свой слоган *Big Data, Fast Insight*, который, по сути, означает то же самое – превращение массивно-параллельного хранилища данных в аналитическую платформу. И для этого тоже используется технология MapReduce, встроенная в СУБД. Однако, в отличие от Greenplum, эта технология применяется не для обеспечения альтернативного внешнего способа обработки данных, а для реализации нового механизма хорошо распараллеливаемых (по модели MapReduce), самоопределяемых и полиморфных табличных функций, определяемых пользователями и вызываемых из операторов выборки SQL [30].

3.2.1. Предпосылки и преимущества использования механизма SQL/MapReduce

По мнению основных разработчиков СУБД *n*Cluster, декларативный язык SQL во многом ограничивает использование аналитических СУБД. С одной стороны, несмотря на постоянное наращивание аналитических возможностей этого языка, для многих аналитиков их оказывается недостаточно. С другой стороны, эти возможности постепенно становятся такими сложными и непонятными, что зачастую становится проще написать процедурный код, решающий частную аналитическую задачу. Наконец, оптимизаторы запросов SQL-ориентированных СУБД постоянно отстают от развития языка, и планы сложных аналитических запросов могут быть весьма далеки от оптимальных, что приводит к их недопустимо долгому выполнению, а иногда и аварийному завершению.

Эти проблемы частично решаются за счет поддержки в SQL-ориентированных СУБД механизма функций, определяемых пользователями (User-Defined Function, UDF). Такие функции позволяют пользователям решать внутри сервера баз данных свои прикладные задачи путем написания соответствующего процедурного кода. Однако традиционные механизмы UDF разрабатывались в расчете на "одноузловые" СУБД, и по умолчанию предполагается чисто последовательное выполнение UDF. Как упоминалось в этом разделе ранее, автоматическое распараллеливание последовательного кода в массивно-параллельной среде с разделением данных является сложной нерешенной проблемой.

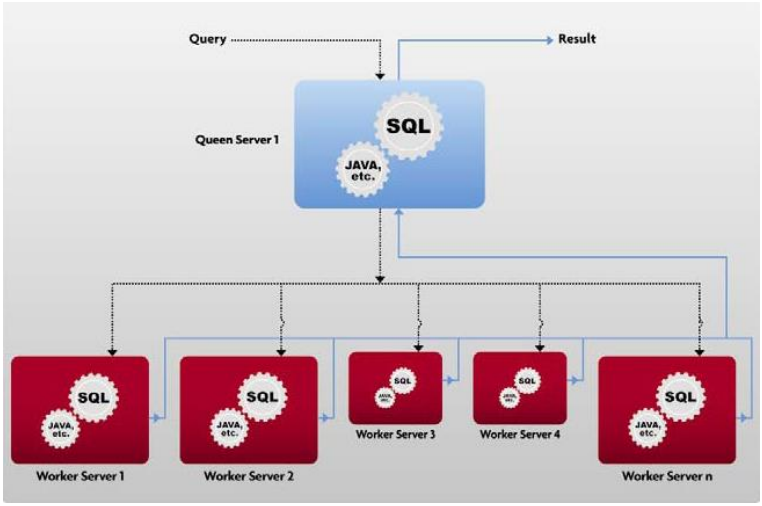


Рис. 2. Совместное выполнение SQL-запросов и SQL/MapReduce-функций в nCluster

В Aster Data для обеспечения механизма естественно распараллеливаемых UDF разработана инфраструктура SQL/MapReduce, поддерживаемая внутри SQL-ориентированной массивно-параллельной СУБД *n*Cluster (см. Рис. 2, позаимствованный из [47]). Организация среды SQL/MapReduce обеспечивает следующие возможности:

- можно эффективно выполнять в "реляционном" стиле операции над таблицами с использованием SQL, а "нереляционные" задачи и оптимизации – возлагать на явно программируемые процедурные функции;
- поскольку функции выполняются над согласованными данными из таблиц базы данных, обеспечивается согласованность вычислений;
- оценочный (cost-based) оптимизатор может принимать решения о способе выполнения SQL-запросов, содержащих вызовы SQL/MapReduce-функций, на основе достоверной статистики данных;
- пользователи *n*Cluster могут формулировать SQL-запросы с использованием высокоуровневых средств анализа данных, воплощенных в SQL/MapReduce-функциях.

SQL/MapReduce-функции можно программировать как на традиционных языках программирования (Java, C#, C++), так и скриптовых языках (Python, Ruby). При этом, независимо от используемого языка программирования, эти функции являются *самоописываемыми* и *полиморфными*. Поскольку здесь идет речь о *табличных* функциях (т.е. функциях, входными параметрами и результатами которых являются таблицы), то это означает, что одна и та же функция может принимать на вход таблицы с разными схемами (функция настраивается на конкретную схему входной таблицы на этапе формирования плана запроса, содержащего ее вызов) и выдавать таблицы также с разными схемами (функция сама сообщает планировщику запроса схему своего результата на этапе формирования плана запроса). Это свойство SQL/MapReduce-функций упрощает процедуру их регистрации в системе (в частности, не требуется выполнение специального оператора SQL `CREATE FUNCTION`, в котором описывались бы схемы входной и выходной таблиц) и способствует повторному использованию кода.

Синтаксические особенности определения SQL/MapReduce-функций и их семантика делают эти программные объекты естественным образом параллелизуемыми по данным: во время выполнения для каждой функции образуются ее экземпляры, параллельно выполняемые в узлах, которые содержат требуемые данные. Вызовы функций подобны подзапросам SQL, что обеспечивает возможность композиции функций, при которой при вызове функции вместо спецификации входной таблицы можно задавать вызов другой функции. Наконец, внешняя эквивалентность вызова SQL/MapReduce-функции подзапросу позволяет применять при формировании плана SQL-запроса с вызовами таких функций обычную оценочную оптимизацию на

основе статистики, а также динамически изменять порядок выполнения функций и вычисления настоящих SQL-подзапросов.

В следующем пункте без излишних деталей обсуждаются синтаксические конструкции, семантика и особенности реализации, обеспечивающие отмеченные свойства SQL/MapReduce-функций (более подробное описание см. в [30]).

3.2.2. Синтаксис, семантика и особенности реализации SQL/MapReduce

Обсудим, как вызываются SQL/MapReduce-функции, как они определяются, и как обрабатываются SQL-запросы с вызовами таких функций.

3.2.2.1. Синтаксис вызова SQL/MapReduce-функции

Вызов SQL/MapReduce-функции может присутствовать только в качестве элемента списка ссылок на таблицы раздела `FROM` SQL-запроса. Синтаксис вызова показан на Рис. 3.

```
SELECT ...
FROM function_name(
  ON table-or-query
  [PARTITION BY expr, ...]
  [ORDER BY expr, ...]
  [clausename(arg, ...) ...]
  )
...
```

Рис. 3. Синтаксис вызова SQL/MapReduce-функции

В разделе `ON`, который является единственным обязательным разделом вызова, указывается любой допустимый SQL/MapReduce-запрос (SQL-запрос, вызов SQL/MapReduce-функции или просто имя таблицы). Во время формирования плана запроса, содержащего вызов SQL/MapReduce-функции, схемой входной таблицы этого вызова считается схема результата запроса, указанного в разделе `ON`.

Раздел `PARTITION BY` указывается только в вызовах SQL/MapReduce-функций над разделами (аналоге функции *Reduce* исходной модели MapReduce, см. ниже). В этом случае в разделе `PARTITION BY` указывается список выражений, на основе значений которых производится разделение таблицы, специфицированной в разделе `ON`. При наличии раздела `PARTITION`

BY в вызове может содержаться и раздел ORDER BY, указывающий на потребность в сортировке входных данных до реального вызова функции. Наконец, вслед за разделом ORDER BY можно указать произвольное число дополнительных разделов со специальными аргументами. Имена этих разделов и значения аргументов передаются в SQL/MapReduce-функцию при ее инициализации.

3.2.2.2. Модель выполнения SQL/MapReduce-функций

В среде SQL/MapReduce используется модель выполнения функций, являющаяся обобщением модели MapReduce. Функция SQL/MapReduce может быть либо *функцией над строками (row function)*, либо *функцией над разделами (partition function)*. Функции первого типа является аналогами функций *Map* классической модели MapReduce, а функции второго типа – аналогами функций *Reduce*. Поскольку, как отмечалось ранее, в разделе ON вызова SQL/MapReduce-функции может содержаться вызов другой SQL/MapReduce-функции, в среде SQL/MapReduce допускается любое число и любой порядок вызовов функций *Map* и *Reduce*, а не только жесткая последовательность *Map-Reduce*, допускаемая классической моделью.

При выполнении функции над строками каждая строка входной таблицы обрабатывается ровно одним экземпляром этой функции. С точки зрения семантики каждая строка обрабатывается независимо, поэтому входная таблица может разделяться по экземплярам функции произвольным образом, что обеспечивает возможности параллелизма и масштабирования. Для каждой строки входной таблицы функция над строками может не производить ни одной строки, а может произвести несколько строк.

При выполнении функции над разделами каждая группа строк, образованная на основе спецификации раздела PARTITION BY вызова функции, обрабатывается ровно одним экземпляром этой функции, и этот экземпляр получает все группу целиком. Если в вызове функции содержится раздел ORDER BY, то экземпляры функции получают разделы в уже упорядоченном виде. С точки зрения семантики каждая строка обрабатывается независимо, что обеспечивает возможности параллелизма на уровне разделов. Для каждого входного раздела функция над строками может не производить ни одной строки, а может произвести несколько строк.

3.2.2.3. Реализация SQL/MapReduce-функций

Как отмечалось в предыдущем пункте, для реализации SQL/MapReduce-функций можно использовать разные языки, но все они являются объектно-ориентированными. Каждая SQL/MapReduce-функция реализуется в виде отдельного класса, и при выработке плана выполнения SQL-запроса, содержащего вызовы таких функций, для каждого вызова образуется объект соответствующего класса с обращением к его методу-конструктору

(инициализатору функции). Это обеспечивает настройку функции и получение требуемого описания ее результирующей таблицы.

Более точно, взаимодействие оптимизатора запросов с инициализатором функции производится через специальный объект, называемый *контрактом времени выполнения (Runtime Contract)*. Анализируя вызов функции, оптимизатор выявляет имена и типы данных столбцов входной таблицы, а также имена и значения разделов дополнительных параметров и соответствующим образом заполняет некоторые поля объекта-контракта, который затем передается инициализатору функции. Инициализатор завершает подготовку контракта путем заполнения его дополнительных полей, содержащих, в частности, информацию о схеме результирующей таблицы, и обращается к методу `complete` объекта-контракта. На основе готового контракта продолжается выработка плана выполнения запроса, и этот контракт соблюдается при последующем выполнении SQL/MapReduce-функции всеми ее экземплярами.

Наиболее важными методами интерфейсов классов для функций над строками и разделами являются методы `OperateOnSomeRows` и `OperateOnPartition`. При обращении к этим методам (реальном выполнении соответствующей функции) в качестве аргументов передаются *итератор над строками*, для обработки которых вызывается функция, и объект *emitter*, с помощью вызовов которого возвращаются результирующие строки.

Чтобы можно было начать использовать некоторую SQL/MapReduce-функцию, ее нужно инсталлировать. Для этого используется общий механизм инсталляции файлов, реализованный в *n*Cluster. Этот механизм реплицирует файл во всех рабочих узлах системы. Далее проверяется, что этот файл содержит SQL/MapReduce-функцию, а также выясняются ее статические свойства: является ли она функцией на строками или же над разделами, содержит ли она вызовы комбинатора и т.д.

Таким образом в этих двух системах обеспечивается возможность развитого анализа данных поблизости от самих данных. Разработчики серверных аналитических приложений несколько ограничиваются моделью MapReduce (в большей степени в Greenplum Database, в меньшей – в *n*Cluster), но зато пользовательский процедурный код хорошо распараллеливается по данным в массивно-параллельной среде.

4. Параллельная СУБД на основе MapReduce

Начну этот раздел с того, что в одной из первых серьезных статей, посвященных сравнению эффективности технологий MapReduce и массивно-параллельных СУБД при решении аналитических задач [27], утверждалось, что развитость и зрелость технологии параллельных СУБД категории sharing-nothing позволяет им обходиться стоузловыми кластерами для поддержки

самых крупных сегодняшних аналитических баз данных петабайтного масштаба. Вместе с тем, особые качества масштабируемости и отказоустойчивости технологии MapReduce проявляются при использовании кластеров с тысячами узлов. Из этого делался вывод, что в обозримом будущем эти качества параллельным СУБД не то чтобы не требуются, но, во всяком случае, не являются для них настоятельно необходимыми.

Однако спустя всего несколько месяцев появилась статья [30], в которой звучат уже совсем другие мотивы (и это при том, что авторские коллективы [27] и [30] значительно пересекаются). В [30] говорится, что в связи с ростом объема данных, которые требуется анализировать, возрастает и число приложений, для поддержки которых нужны кластеры с числом узлов, больше ста. В то же время, имеющиеся в настоящее время параллельные СУБД не масштабируются должным образом до сотен узлов. Это объясняется следующими причинами.

- При возрастании числа узлов кластера возрастает вероятность отказов отдельных узлов, а массивно-параллельные СУБД проектировались в расчете на редкие отказы.
- Современные параллельные СУБД расчитаны на однородную аппаратную среду (все узлы кластера обладают одной и той же производительностью), а при значительном масштабировании полной однородности среды добиться почти невозможно.
- До последнего времени имелось очень небольшое число систем аналитических баз данных, для достижения требуемой производительности которых требовались кластеры с более чем несколькими десятками узлов. Поэтому существующие параллельные СУБД просто не тестировались в более масштабной среде, и при их дальнейшем масштабировании могут встретиться непредвиденные технические проблемы.

Требуемые характеристики масштабируемости и отказоустойчивости может обеспечить технология MapReduce, поскольку она с самого начала разрабатывалась с расчетом на масштабирование до тысяч узлов, и ее реализация от Google эффективно используется для поддержки внутренних операций этой компании. Несмотря на то, что изначально технология MapReduce ориентировалась на обработку неструктурированных текстовых данных, известны показательные примеры ее использования и для обработки огромных объемов структурированных данных.

Однако объективно при обработке структурированных данных MapReduce не может конкурировать с параллельными СУБД по производительности, что объясняется отсутствием схемы у обрабатываемых данных, индексов, оптимизации запросов и т.д. В результате при выполнении многих типичных аналитических запросов MapReduce показывает производительность, более чем на порядок уступающую производительности параллельных СУБД [27, 31].

В проекте HadoopDB [48] специалисты из университетов Yale и Brown предпринимают попытку создать гибридную систему управления данными, сочетающую преимущества технологий и MapReduce, и параллельных СУБД. В их подходе MapReduce обеспечивает коммуникационную инфраструктуру, объединяющую произвольное число узлов, в которых выполняются экземпляры традиционной СУБД. Запросы формулируются на языке SQL, транслируются в среду MapReduce, и значительная часть работы передается в экземпляры СУБД. Наличие MapReduce обеспечивается масштабируемость и отказоустойчивость, а использование в узлах кластера СУБД позволяет добиться высокой производительности.

4.1. Общая организация HadoopDB

Архитектура HadoopDB показана на Рис. 4, позаимствованном из [49]. Основой системы является реализация MapReduce, выполненная в проекте Hadoop [33]. К ней добавлены компоненты компиляции поступающих в систему SQL-запросов, загрузки и каталогизирования данных, связи с СУБД и самих СУБД. При реализации всех компонентов системы максимально использовались пригодные для этого программные средства с открытыми исходными текстами.

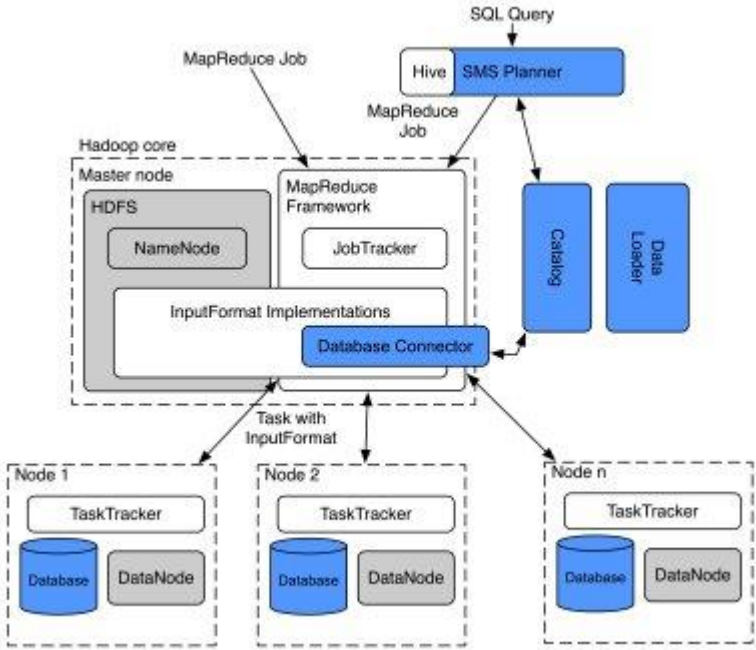


Рис. 4. Архитектура HadoopDB

4.1.1. Немного про Hadoop MapReduce

Как отмечалось в разделе 2, реализация Hadoop MapReduce основана на спецификациях Google, содержащихся в [24]. Однако в этом проекте используется собственная терминология, и для простоты описания особенностей HadoopDB в этом пункте кратко описывается организация Hadoop MapReduce в терминах Hadoop.

Hadoop MapReduce опирается на распределенную файловую систему HDFS (Hadoop Distributed File System) [36]. Файлы HDFS имеют блочную структуру, и блоки одного файла распределяются по узлам данных (DataNode). Файловая система работает под централизованным управлением выделенного узла имен (NameNode), в котором поддерживаются метаданные о файлах (в том числе, об их размерах, о размещении блоков и их реплик и т.д.).

В самой среде Hadoop MapReduce в соответствии с [24] поддерживаются один узел-распорядитель (в Hadoop он называется JobTracker) и много узлов-исполнителей (здесь TaskTracker). В узле JobTracker планируется выполнение MR-заданий, а также отслеживаются данные о загрузке узлов TaskTracker и доступных ресурсах. Каждое задание разбивается на задачи *Map* и *Reduce*, которые назначаются узлом JobTracker узлам TaskTracker с учетом требований локальности данных и балансировки нагрузки.

Требование локальности удовлетворяется за счет того, что JobTracker пытается назначать каждую задачу *Map* тому узлу TaskTracker, для которого данные, обрабатываемые этой задачей, являются локальными. Балансировка нагрузки достигается путем назначения задач всем доступным узлам TaskTracker. Узлы TaskTracker периодически посылают в узел JobTracker контрольные сообщения с информацией о своем состоянии.

Для обеспечения доступа к входным данным MR-задания поддерживается библиотека InputFormat. В Hadoop MapReduce имеется несколько реализаций этой библиотеки, одна из которых позволяет всем задачам одного MR-задания обращаться к JDBC-совместимой базе данных.

4.1.2. Собственные компоненты HadoopDB

Как показывает рис. 4, в архитектуре HadoopDB присутствует ряд компонентов, расширяющих среду Hadoop MapReduce.

4.1.2.1. Коннектор баз данных (Database Connector)

Коннектор баз данных обеспечивает интерфейс между TaskTracker и независимыми СУБД, располагаемыми в узлах кластера. Этот компонент расширяет класс InputFormat и является частью соответствующей библиотеки. От каждого MR-задания в коннектор поступает SQL-запрос, а также параметры подключения к системе баз данных (указание драйвера JDBC, размер структуры выборки данных и т.д.).

Теоретически коннектор обеспечивает подключение к любой JDBC-совместимой СУБД. Однако в других компонентах HadoopDB приходится учитывать специфику конкретных СУБД, поскольку для них требуется по-разному оптимизировать запросы. В экспериментах, описываемых в [30], использовалась реализация коннектора для PostgreSQL, а в [49] уже упоминается некоторая поколоночная система. В любом случае, для среды HadoopDB эта реализация обеспечивает естественное и прозрачное использование баз данных в качестве источника входных данных.

4.1.2.2. Каталог

В каталоге поддерживаются метаданные двух сортов: параметры подключения к базе данных (ее месторасположение, класс JDBC-драйвера, учетные данные) и описание наборов данных, содержащихся в кластере, расположение реплик и т.д. Каталог сохраняется в формате XML в HDFS. К нему обращаются JobTracker и TaskTracker для выборки данных, требуемых для планирования задач и обработки данных.

4.1.2.3. Загрузчик данных (Data Loader)

Обязанностями загрузчика данных являются:

- глобальное разделение данных по заданному ключу при их загрузке из HDFS;
- разбиение данных, хранимых в одном узле, на несколько более мелких разделов (*чанков, chunk*);
- массовая загрузка данных в базу данных каждого узла с использованием чанков.

Загрузчик данных состоит из компонентов GlobalHasher и LocalHasher. GlobalHasher запускает в Hadoop MapReduce специальное задание, в котором читаются файлы данных HDFS и производится их разделение на столько частей, сколько имеется рабочих узлов в кластере. Сортировка данных не производится. Затем LocalHasher в каждом узле копирует соответствующий раздел из HDFS в свою файловую систему, разделяя его на чанки в соответствии с установленным в системе максимальным размером чанка.

В GlobalHasher и LocalHasher используются разные хэш-функции, обеспечивающие примерно одинаковые размеры всех чанков. Эти хэш-функции отличаются от хэш-функции, используемой в Hadoop MapReduce для разделения данных по умолчанию. Это способствует улучшению балансировки нагрузки.

4.1.2.4. Планирование SQL-запросов

Внешний интерфейс HadoopDB позволяет выполнять SQL-запросы. Компиляцию и подготовку планов выполнения SQL-запросов производит планировщик SMS (SMS Planner на рис. 4), являющийся расширением планировщика Hive [50].

Планировщик Hive преобразует запросы, представленные на языке HiveQL (вариант SQL) в задания MapReduce, которые выполняются над таблицами, хранимыми в виде файлов HDFS. Эти задания представляются в виде ориентированных ациклических графов (directed acyclic graph, DAG) реляционных операций фильтрации (ограничения), выборки (проекции), соединения, агрегации, каждая из которых выполняется в конвейере: после обработки каждого очередного кортежа результат каждой операции направляется на вход следующей операции.

Операции соединения, как правило, выполняются в задаче *Reduce* MR-задания, соответствующего SQL-запросу. Это связано с тем, что каждая обрабатываемая таблица сохраняется в отдельном файле HDFS, и невозможно предполагать совместного размещения соединяемых разделов таблиц в одном узле кластера. Для HadoopDB это не всегда так, поскольку соединяемые таблицы могут разделяться по атрибуту соединения, и тогда операцию соединения можно вытолкнуть на уровень СУБД.

Для пояснения того, как работает планировщик Hive, и каким образом его функциональность расширяется в SMS, невозможно обойтись без примера, и для простоты воспользуемся примером из [30]. Пусть задан следующий простой запрос с агрегацией, смысл которого состоит в получении ежегодных суммарных доходов от продаж товаров:

```
SELECT YEAR(saleDate), SUM(revenue)
  FROM sales
  GROUP BY YEAR(saleDate);
```

В Hive этот запрос обрабатывается следующим образом:

- Производится синтаксический разбор запроса, и образуется его абстрактное синтаксическое дерево.
- Далее работает семантический анализатор, который выбирает из внутреннего каталога Hive *MetaStore* информацию о схеме таблицы `sales`, а также инициализирует структуры данных, требуемые для сканирования этой таблицы и выборки нужных полей.
- Затем генератор логических планов запросов производит план запроса – DAG реляционных операций.
- Вслед за этим оптимизатор перестраивает этот план запроса, проталкивая, например, операции фильтрации ближе к операциям сканирования таблиц. Основной функцией оптимизатора является разбиение плана на фазы *Map* и *Reduce*. В частности, перед операциями соединения и группировки добавляется операция переразделения данных (Reduce Sink). Эти операции отделяют фазу *Map* от фазы *Reduce*. Оценочная (cost-based) оптимизация не используется, и поэтому получаемые планы не всегда эффективны.
- Генератор физических планов выполнения запросов преобразует логический план в физический, допускающий выполнение в виде

одного или нескольких MR-заданий. Первая (и каждая аналогичная) операция Reduce Sink помечает переход от фазы Map к фазе Reduce некоторого задания MapReduce, а остальные операции Reduce Sink помечают начало следующего задания MapReduce. Физический план выполнения приведенного выше запроса, сгенерированный планировщиком Hive, показан на рис. 5.

- Полученный DAG сериализуется в формате XML. Задания инициируются драйвером Hive, который руководствуется планом в формате SQL и создает все необходимые объекты, сканирующие данные в таблицах HDFS и покортежно обрабатывающие данные.

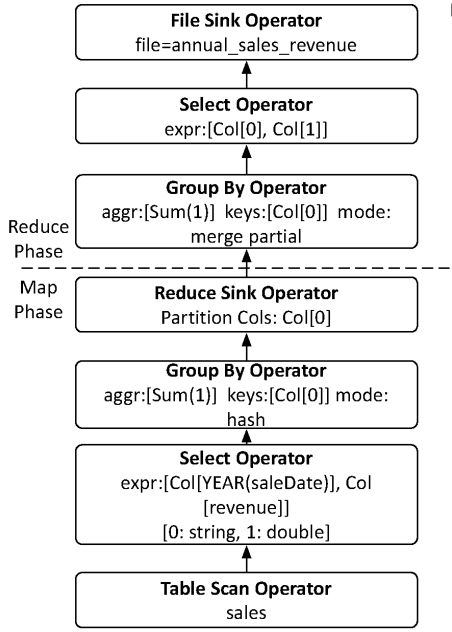


Рис. 5. Задание MapReduce, генерируемое Hive

В планировщике SMS функциональность планировщика Hive расширяется следующим образом. Во-первых, до обработки каждого запроса модифицируется MetaStore, куда помещается информация о таблицах базы данных. Для этого используется каталог HadoopDB (см. выше).

Далее, после генерации физического плана запроса и до выполнения MR-заданий выполняются два прохода по физическому плану. На первом проходе устанавливается, какие столбцы таблиц действительно обрабатываются

запросом, и определяются ключи разделения, используемые в операциях Reduce Sink.

На втором проходе DAG запроса обходится снизу-вверх от операций сканирования таблиц до формирования результата или первой операции Reduce Sink. Все операции этой части DAG преобразуются в один или несколько SQL-запросов, которые проталкиваются на уровень СУБД. Для повторного создания кода SQL используется специальный основанный на правилах генератор.

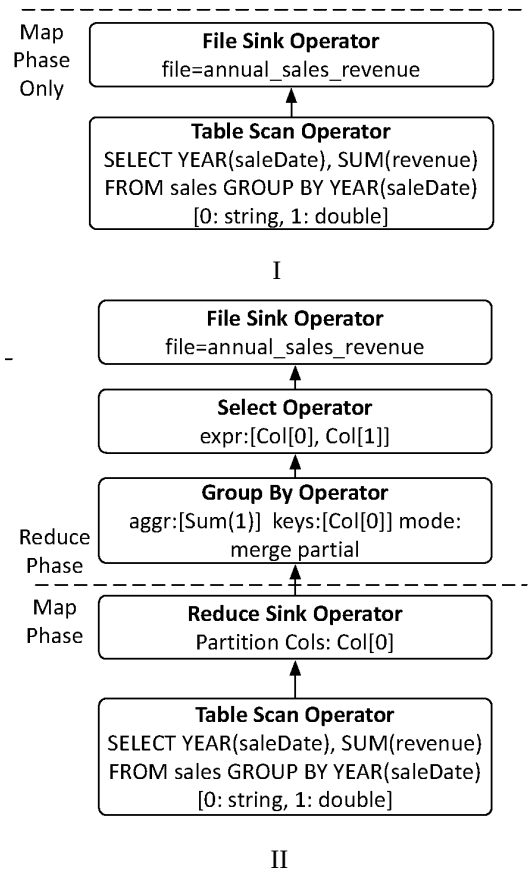


Рис. 6. Варианты MR-заданий, генерируемые SMS

На рис. 6 показаны два плана, которые производит SQL для приведенного выше запроса. План в левой части рисунка производится в том случае, если таблица `sales` является разделенной по `YEAR(saleDate)`. В этом случае

вся логика выполнения запроса выталкивается в СУБД. Задача *Map* всего лишь записывает результаты запроса в файл HDFS.

В противном случае генерируется план, показанный в правой части рис. 6. При выполнении запроса по этому плану на уровне базы данных производится частичная агрегация данных, а для окончательной агрегации требуется выполнение задачи *Reduce*, производящей слияние частичных результатов группировки, которые получены в каждом узле на фазе задачи *Map*.

4.2. Производительность, масштабируемость и устойчивость к отказам и падению производительности узлов кластера

В [30] описан ряд экспериментов, показывающих, что гибридное использование технологий MapReduce и баз данных в реализации HadoopDB позволяет добиться от этой системы производительности, соизмеримой с производительностью параллельных СУБД, и устойчивости к отказам и падению производительности узлов, свойственной MapReduce. Я не буду в этой статье описывать детали этих экспериментов, а лишь кратко отмечу их основные результаты.

4.2.1. Производительность и маштабируемость

В большинстве экспериментов параллельные СУБД существенно превосходят HadoopDB по производительности, а HadoopDB оказывается значительно (иногда на порядок) производительнее связки Hive и Hadoop MapReduce. В экспериментах использовались поколоночная параллельная СУБД Vertica и некоторая коммерческая параллельная СУБД-X с хранением таблиц по строкам. Наибольшую производительность, естественно, демонстрировала Vertica, но в ряде случаев HadoopDB уступала ей значительно меньше, чем на десятичный порядок.

В [30] значительное отставание HadoopDB от параллельных СУБД объясняется тем, что в качестве базовой СУБД в HadoopDB использовалась PostgreSQL, в которой отсутствует возможность хранения таблиц по столбцам (как уже отмечалось, в [49] в HadoopDB уже используется поколоночная СУБД). Кроме того, в экспериментах с HadoopDB не использовалось сжатие данных. Наконец, в HadoopDB возникали значительные накладные расходы на взаимодействие Hadoop MapReduce и PostgreSQL, которые потенциально можно снизить. Так что в целом производительность HadoopDB не должна критически отставать от производительности параллельных СУБД.

Время загрузки данных в HadoopDB в десять раз больше соответствующего времени для Hadoop MapReduce. Однако это окупается десятикратным выигрышем в производительности при выполнении некоторых запросов.

Как и следовало ожидать, при возрастании числа узлов в кластере при одновременном увеличении объема данных HadoopDB (как и Hadoop)

масштабируется почти линейно. Но следует заметить, что в этом диапазоне не хуже масштабируется и Vertica (с СУБД-X дела обстоят несколько хуже), а эксперименты на кластерах большего размера не производились. Так что объективных данных в этом отношении пока нет.

4.2.2. . Устойчивость к отказам и неоднородности среды

В экспериментах с отказоустойчивостью и падением производительности некоторого узла сравнивались HadoopDB, Hadoop MapReduce с Hive и Vertica. В первом случае работа одного из узлов кластера искусственным образом прекращалась после выполнения 50% обработки запроса. Во втором случае работа одного узла замедлялась за счет выполнения фонового задания с большим объемом ввода-вывода с тем же диском, на котором сохранялись файлы соответствующей системы.

При продолжении работы после отказа узла СУБД Vertica приходилось выполнять запрос заново с использованием реплик данных, и время выполнения запроса возрастало почти вдвое. В HadoopDB и Hadoop MapReduce с Hive время выполнения увеличивалось примерно на 15-20% за счет того, что задачи, выполнявшиеся на отказавшем узле, перераспределялись между оставшимися узлами. При этом относительная производительность HadoopDB оказывается несколько выше, чем у Hadoop MapReduce с Hive, поскольку в первом случае обработка запроса проталкивалась на узлы, содержащие реплики баз данных, а во втором приходилось копировать данные, не являющиеся локальными для обрабатывающего узла.

При замедлении работы одного из узлов производительность Vertica определялась скоростью этого узла, и в экспериментах время выполнения запроса увеличивалось на 170%. При использовании HadoopDB и Hadoop MapReduce с Hive время выполнения запроса увеличивалось всего на 30% за счет образования резервных избыточных задач в недозагруженных узлах.

Проект HadoopDB представляется мне очень интересным и перспективным. В отличие от других систем, рассматриваемых в этой статье, HadoopDB – это проект с открытыми исходными текстами, так что потенциально участие в этой работе доступно для всех желающих. Помимо прочего, продукт HadoopDB открывает путь к созданию высокопроизводительных, масштабируемых и отказоустойчивых параллельных СУБД на основе имеющихся программных средств с открытыми кодами.

5. Использование MapReduce для подготовки данных параллельных СУБД

Я не могу считать себя специалистом в области различных средств ETL (Extract-Transform-Load), которых существует множество, и которые применяются во многих компаниях, использующих хранилища данных (я не

буду даже пытаться как-то их классифицировать и/или сравнивать). Но, в любом случае, важность этих средств трудно переоценить, поскольку в хранилище данных по определению поступают данные из самых разнообразных источников: транзакционных баз данных, сообщений, участвующих в организации бизнес-процессов, электронной почты, журналов Web-серверов и т.д. Все эти данные нужно очистить, привести к единому формату, согласовать и загрузить в хранилище данных для последующего анализа.

Наверное, можно согласиться с идеологами MAD-аналитики из Greenplum (см. п. 3.1.1), что при использовании ортодоксального подхода к организации хранилищ данных подключение нового источника к хранилищу данных может занять недопустимо много времени во многом как раз из-за потребности в создании соответствующей процедуры ETL. Наверное, можно согласиться и с тем, что для аналитиков гораздо важнее получить новые данные, чем быть вынужденными ждать неопределенное время их в согласованной форме. Но совершенно очевидно, что если данные в хранилище данных не очищать никогда, то со временем в них не разберется никакой, даже самый передовой аналитик.

Итак, что мы имеем. Число источников данных, пригодных для анализа в составе хранилища данных, все время растет. Их разнородность тоже все время возрастает. Все меньший процент составляют структурированные базы данных, данные поступают из частично структурированных файлов и совсем неструктурированных текстовых документов. Для каждой разновидности источников данных нужна своя разновидность процедуры ETL, и по причине роста объемов исходных данных для обеспечения умеренного времени их загрузки в хранилище данных эти процедуры должны выполняться в массивно-параллельной среде. И в этом может помочь технология MapReduce.

5.1. MapReduce и ETL

Как отмечается в [31], для канонического способа использования технологии MapReduce характерно применение следующих операций:

- чтение журнальных данных из нескольких разных файлов-журналов;
- разбор и очистка журнальных данных;
- преобразования этих данных, в том числе их частичная агрегация;
- принятие решения о схеме результирующих данных;
- загрузка данных в хранилище данных или другую систему хранения.

В точности такие же шаги выполняются в системах ETL при извлечении, преобразовании и загрузке данных. По сути дела, MapReduce производит из исходных "сырых" данных некоторую полезную информацию, которую потребляет другая система хранения. В некотором смысле можно считать любую реализацию MapReduce параллельной инфраструктурой выполнения процедур ETL.

Имелись попытки реализации процедур ETL внутри сервера баз данных средствами языка SQL. Разработчики параллельных СУБД с поддержкой MapReduce Greenplum Database и *n*Cluster компании Aster Data тоже намекают, что их встроенный MapReduce можно использовать и для поддержки ETL. Но исторически системы ETL промышленного уровня существуют отдельно от СУБД. Обычно СУБД не пытаются выполнять ETL, а системы ETL не поддерживают функции СУБД.

5.2. Hadoop и Vertica

Эти рассуждения наводят на мысль, что если иметь в виду поддержку именно ETL, то наиболее грамотное (и самое простое) решение по интеграции технологий MapReduce и параллельных баз данных применяется в Vertica (см. рис. 7, позаимстованный из [51]).

В Vertica реализован свой вариант интерфейса DBInputFormat компании Cloudera [51] для Hadoop MapReduce, позволяющий разработчикам MapReduce выбирать данные из баз данных Vertica и направлять результирующие данные в эти базы данных. При этом подходе технологии MapReduce и параллельных баз данных тесно не интегрируются, но каждая из них может использовать возможности другой технологии.

Скорее всего, мы еще многое услышим о системах ETL, основанных на использовании технологии MapReduce, и, скорее всего, предводителем этого направления будет Vertica.

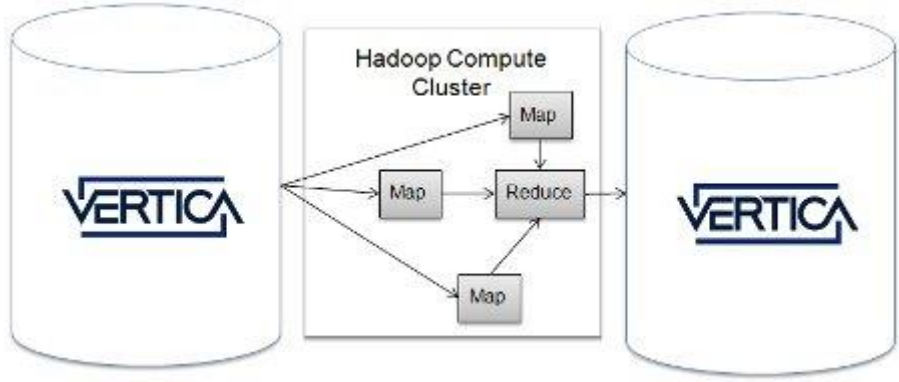


Рис. 7. MapReduce

6. Заключение

Еще пару лет назад было непонятно, каким образом можно с пользой применять возникающие "облачные" среды для высокоуровневого управления

данными. Многие люди считали, что в "облаках" системы управления базами данных будут просто вытеснены технологий MapReduce. Это вызывало естественное недовольство сообщества баз данных, авторитетные представители которого старались доказать, что пытаться заменить СУБД какой-либо реализацией MapReduce если не безнравственно, то, по крайней мере, неэффективно.

Однако вскоре стало понятно, что технология MapReduce может быть полезна для самих параллельных СУБД. Во многом становлению и реализации этой идеи способствовали компании-стартапы, выводящие на рынок новые аналитические массивно-параллельные СУБД и добивающиеся конкурентных преимуществ. Свою лепту вносили и продолжают вносить и университетские исследовательские коллективы, находящиеся в тесном сотрудничестве с этими начинающими компаниями.

На сегодняшний день уже понятно, что технология MapReduce может эффективно применяться внутри параллельной аналитической СУБД, служить инфраструктурой отказоустойчивой параллельной СУБД, а также сохранять свою автономность в симбиотическом союзе с параллельной СУБД. Все это не только мешает развитию технологии параллельных СУБД, а наоборот, способствует ее совершенствованию и распространению.

Интересные работы ведутся и в направлении использования "облачных" сред для создания нового поколения транзакционных средств управления данными. Но это уже, как говорили братья Стругацкие, совсем другая история.

Литература

[1] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, Gerhard Weikum. The Claremont Report on Database Research, http://db.cs.berkeley.edu/claremont/claremontreport08.pdf, 2008 г.
Перевод на русский язык: Ракеш Агравал, Анастасия Айламаки, Филипп Бернштейн, Эрик Брювер, Майкл Кери, Сураджит Чаудхари, Анхай Доан, Даниэла Флореску, Майкл Франклин, Гектор Гарсиа Молина, Йоханнес Герке, Ле Грюнвальд, Лаура Хаас, Элон Хэлеви, Джозеф Хелерстейн, Яннис Иоаннидис, Хэнк Корт, Дональд Коссман, Сэмюэль Мэдден, Роджер Магулас, Бенг Чин Ой, Тим О'Рейли, Раджу Рамакришнан, Суннита Сарагави, Майкл Стоунбрейкер, Александер Залай, Герхард Вейкум. Клермонтский отчет об исследованиях в области баз данных, http://citforum.ru/database/articles/claremont_report/, 2008 г.

[2] Curt Monash. Data warehouse appliances – fact and fiction, http://www.dbms2.com/2007/12/03/data-warehouse-appliances-%e2%80%93-fact-and-fiction/, December 3, 2007

[3] Википедия. Britton Lee, Inc., http://en.wikipedia.org/wiki/Britton_Lee,_Inc., 2010

[4] Teradata Home Page, http://www.teradata.com/t/, 2010

[5] C.H.C. Lemg and K.S. Wong. File Processing Efficiency on the Content Addressable File Store, http://www.vldb.org/conf/1985/P282.PDF, Proceedings of the VLDB Conference, Stockholm, 1985, 282-291

[6] Netezza Home Page, http://www.netezza.com/, 2010

[7] Vertica Systems Home Page, http://www.vertica.com/, 2010

[8] DATAllegro Home Page, http://www.datallegro.com/, 2010

[9] Greenplum Home Page, http://www.greenplum.com/, 2010

[10] Aster Data Systems Home Page, http://www.asterdata.com/, 2010

[11] Kognitio Home Page, http://www.kognitio.com/, 2010

[12] EXASOL AG Home Page, http://www.exasol.com/, 2010

[13] Calpont Corporation Home Page, http://www.calpont.com/, 2010

[14] Dataupia Corporation Home Page, http://www.dataupia.com/, 2010

[15] Infobright Home Page, http://www.infobright.com/, 2010

[16] Kickfire Home Page, http://www.kickfire.com/, 2010

[17] SQL Server 2008 R2 Parallel Data Warehouse, http://www.microsoft.com/sqlserver/2008/en/us/parallel-data-warehouse.aspx, 2010

[18] Ingres Corporation Home Page, http://www.ingres.com/, 2010

[19] PostgreSQL Home Page, http://www.postgresql.org/, 2010

[20] MySQL Home Page, http://www.mysql.com/, 2010

[21] Oracle Exadata, http://www.oracle.com/us/products/database/exadata/index.html, 2010

[22] Richard Hackathorn, Colin White. Data Warehouse Appliances: Evolution or Revolution?, http://www.beyeresearch.com/study/4639, June 26, 2007

[23] M. Stonebraker and U. Cetintemel. One Size Fits All: An Idea whose Time has Come and Gone, http://www.cs.brown.edu/%7Eugur/fits_all.pdf // Proc. ICDE, 2005, 2-11. Перевод на русский язык: Майкл Стоунбрейкер, Угур Кетинтемел. Один размер пригоден для всех: идея, время которой пришло и ушло, http://citforum.ru/database/articles/one_size_fits_all/, 2007

[24] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, http://labs.google.com/papers/mapreduce.html // Proceedings of the Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004, 137–150

[25] Michael Stonebraker, David J. DeWitt. MapReduce: A major step backwards, http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/, January 17, 2008

[26] Michael Stonebraker, David J. DeWitt. MapReduce II, http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/, January 25, 2008

[27] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis, http://cs-www.cs.yale.edu/homes/dna/papers/benchmarks-sigmod09.pdf // Proceedings of the 35th SIGMOD International Conference on Management of Data, 2009, Providence, Rhode Island, USA, 165-178.

Перевод на русский язык: Эндрю Павло, Эрик Паулсон, Александр Разин, Дэниэль Абади, Дэвид Девитт, Сэмюэль Мэдден, Майкл Стоунбрейкер. Сравнение подходов к крупномасштабному анализу данных, http://citforum.ru/database/articles/mr_vs_dbms/, 2009

[28] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, Caleb Welton. MAD Skills: New Analysis Practices for Big Data, http://db.cs.berkeley.edu/jmh/papers/madskills-032009.pdf // Proceedings of the VLDB'09 Conference, Lyon, France, August 24-28, 2009, 1481-1492.
Перевод на русский язык: Джеффри Коэн, Брайен Долэн, Марк Данлэп, Джозеф Хеллерстейн, Кейлэб Велтон. МОГучие способности: новые приемы анализа больших данных, http://citforum.ru/database/articles/mad_skills/, 2009

[29] Eric Friedman, Peter Pawlowski, John Cieslewicz. SQL/MapReduce: A practical approach to self-describing, polymorphic, and parallelizable userdefined functions, http://www.asterdata.com/resources/downloads/whitepapers/sqlmr.pdf // Proceedings of the 35th VLDB Conference, August 24-28, 2009, Lyon, France, 1402-1413.
Перевод на русский язык: Эрик Фридман, Питер Павловски и Джон Кислевич. SQL/MapReduce: практический подход к поддержке самоописываемых, полиморфных и параллелизуемых функций, определяемых пользователями, http://citforum.ru/database/articles/asterdata_sql_mr/, 2010

[30] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, Alexander Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads, http://www.vldb.org/pvldb/2/vldb09-861.pdf // Proceedings of the 35th VLDB Conference, August 24-28, 2009, Lyon, France, 922-933.
Перевод на русский язык: Азза Абузейд, Камил Байда-Павликовски, Дэниэль Абади, Ави Зильбершац, Александр Разин. HadoopDB: архитектурный гибрид технологий MapReduce и СУБД для аналитических рабочих нагрузок, http://citforum.ru/database/articles/hadoopdb/, 2010

[31] Michael Stonebraker, Daniel Abadi, David J. Dawitt, Sam Madden, Erik Paulson, Andrew Pavlo and Alexander Rasin. MapReduce and Parallel DBMSs: Friends or Foes?, http://database.cs.brown.edu/papers/stonebraker-cacm2010.pdf. // Communications of the ACM, vol. 53, no. 1, January 2010, 64-71.
Перевод на русский язык: Майкл Стоунбрейкер, Дэниэль Абади, Дэвит Девитт, Сэм Мэдден, Эрик Паулсон, Эндрю Павло и Александр Разин. MapReduce и параллельные СУБД: друзья или враги?, http://citforum.ru/database/articles/mr_vs_dbms-2/, 2010

[32] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/ru/papers/gfs-sosp2003.pdf // Proceedings of the ACM Symposium on Operating Systems Principles, Bolton Landing, New York, USA, October 19–22, 2003, 29 - 43

[33] Hadoop MapReduce Home Page, http://hadoop.apache.org/mapreduce/, 2010

[34] Apache Hadoop Home Page, http://hadoop.apache.org/, 2010

[35] The Apache Software Foundation Home Pagem, http://www.apache.org/, 2010

[36] Hadoop Distributed File System Home Page, http://hadoop.apache.org/hdfs/, 2010

[37] Map/Reduce Tutorial, http://hadoop.apache.org/common/docs/current/mapred_tutorial.htm, 2010

[38] Lawrence A. Rowe, Michael R. Stonebraker. The POSTGRES Data Model, http://www.vldb.org/conf/1987/P083.PDF // Proceedings of the 13th VLDB Conference, Brighton, 1987, 83-96

[39] Michael Stonebraker, Erlka Anderson, Eric Hanson and Brad Ruben. Quel as a Data Type. ACM SIGMOD Record, Volume 14 , Issue 2 (June 1984), 208-214

[40] Michael Stonebraker, Jeff Anton, Eric N. Hanson. Extending a Database System with Procedures, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.4497&rep=rep1&type=pdf // ACM Trans. Database Syst. 12(3), 1987, 350-376

[41] MPI: The Message Passing Interface, http://parallel.ru/tech/tech_dev/mpi.html, 2010

[42] Michael Stonebraker, JeffAnton, and Michael Hirohama. Extensibility in Postgres, http://sites.computer.org/debull/87JUN-CD.pdf // IEEE Database Engineering Bulletin 10(2), June 1987, 16-24

[43] PostgreSQL 8.4.3 Documentation. V. Server Programming, http://www.postgresql.org/docs/8.4/static/server-programming.html, 2010

[44] A Unified Engine for RDBMS and MapReduce, http://www.greenplum.com/download.php?alias=register-map-reduce&file=Greenplum-MapReduce-Whitepaper.pdf, Greenplum Whitepaper, 2009

[45] Python Package Index (PyPi) Home Page, http://pypi.python.org/pypi. 2010

[46] Comprehensive Perl Archive Network (CPAN) Home Page, http://www.cpan.org/, 2010

[47] Ajeet Singh. Aster Data's SQL-MapReduce: Deriving Deep Insights from Large Datasets, http://www.asterdata.com/resources/assets/wp_Aster_Data_4.0_MapReduce_Technical_Whitepaper.pdf, Aster Data Whitepaper, 2009

[48] HadoopDB Home Page, http://db.cs.yale.edu/hadoopdb/hadoopdb.html, 2010

[49] Azza Abouzied, Kamil Bajda-Pawlikowski, Jiewen Huang, Daniel J. Abadi, Avi Silberschatz. HadoopDB in Action: Building Real World Applications, http://cs-www.cs.yale.edu/homes/dna/papers/hadoopdb-demo.pdf // Proceedings of the 36th SIGMOD International Conference on Management of Data, 2010, Indianapolis, Indiana, USA.

[50] Hive Home Page, http://hadoop.apache.org/hive/, 2010

[51] Aaron Kimball. Database Access with Hadoop, http://www.cloudera.com/blog/2009/03/database-access-with-hadoop/, March 06, 2009

Обмен данными в распределенной системе поддержки решений[1]

Карпов Л. Е., Юдин В. Н.

Аннотация. Обмен данными – расширение возможностей локальной системы поддержки решений с целью привлечения опыта, накопленного другими пользователями в подобных системах. Рассматриваются два варианта обмена в сети, где присутствуют несколько подобных систем: виртуальная интеграция (аналог консилиума) и консолидация (импорт знаний).

1. Введение

Система поддержки принятия решений разрабатываемая в Институте системного программирования РАН, предназначена для помощи пользователю-эксперту в аккумуляции его опыта путем накопления и интерпретации его знаний в виде *прецедентов* (случаев) и информационной поддержки принятия решений в различных областях интеллектуальной деятельности на основе современных информационных технологий: теории принятия решений, вывода по прецедентам и распознавания образов.

Изначально наполняемая смоделированными типовыми схемами принятия решений, а в дальнейшем пополняемая случаями из реальной практики, накопленная совокупность прецедентов не является простым набором слабо связанных между собой случаев. Напротив, эта совокупность структурируется и классифицируется функционирующей системой, образуя так называемую *базу прецедентов*. Система, построенная по такому принципу, является самообучаемой: чем больше прецедентов содержится в базе, тем характернее спектр возможных значений их параметров, тем выше вероятность найти *наиболее подходящий* прецедент, тем, следовательно, выше качество принимаемого решения.

Вывод на основе прецедентов представляет собой метод принятия решений, моделирующий человеческие рассуждения. Метод использует знания о предыдущих ситуациях или случаях (прецедентах), которыми могут быть

[1] Поддержка работы осуществляется Российским Фондом Фундаментальных Исследований по проектам № 09-01-00351-а и № 09-07-00191-а.

встречавшиеся ранее проблемы или типичные случаи, а также принятые в связи с ними решения.

Традиционно прецедент рассматривается, как набор, состоящий из описания некоторого случая или проблемы, описания решения проблемы (действий, предпринимаемых в этом случае) и результата применения решения (исход). При рассмотрении новой проблемы (текущего случая) в системе делается попытка найти похожий прецедент, который в дальнейшем надлежит использовать в качестве аналога. Решение проблемы, выбранное из аналога, частью которого оно является, либо используется прямо и непосредственно, либо адаптируется к текущему случаю. После того, как текущий случай будет решен (то есть для него будет сформировано решение и изучен его результат), этот случай вносится в базу прецедентов вместе со своим решением и исходом для его возможного последующего использования.

В основе подходов к отбору прецедентов лежит оценка схожести прецедента и текущего случая. Обычно для определения этой схожести в пространстве признаков вводится специальная метрика, для вычисления которой в этом пространстве определяются точки, соответствующие текущему случаю и его исследуемым аналогам. На основе вычисленных значений метрики находится ближайшая к текущему случаю точка, представляющая прецедент.

В реальной жизни, когда объекты описываются разнородными признаками, в том числе и логическими, ввести метрику, которая обладает строго определенными свойствами, не всегда удается. В этих случаях вместо метрики используется так называемая *мера близости*. Один из способов применения меры близости использует разбиение базы прецедентов на классы, то есть группы "похожих" случаев (случаи, принадлежащие одному классу, по такому определению, являются схожими). Разбиение базы прецедентов на отдельные классы может проводиться по-разному. Например, такое разбиение можно проводить с помощью экспертного знания, когда признаки заболеваний и границы допустимых значений этих признаков задаются экспертом-врачом на основе его теоретических знаний и имеющегося у него опыта врачебной практики, можно также использовать специально разработанные обучающие выборки случаев.

База прецедентов системы поддержки принятия решений включает в себя:

- описание объемлющего признакового пространства для случаев, хранящихся в базе, в частности, типы и границы признаков объектов,
- описания случаев, рассматриваемых как прецеденты (признаки случая, решение, исход),
- описания классов случаев (перечень признаков класса, границы признаков).

Основные функции системы поддержки – оценка (распознавание) текущего случая, представленного совокупностью признаков, и поиск аналогов случая.

Текущий случай сравнивается с описаниями классов в базе прецедентов, что позволяет отнести его к тому или иному классу (задачу отнесения отдельного объекта к одному из классов называют распознаванием объекта). В разрабатываемой программной системе реализуется метод отбора наиболее подходящих прецедентов, базирующийся на оценке близости, смысл которой описан далее в этой работе.

Каждый класс в базе прецедентов представлен в пространстве признаков многомерным параллелепипедом, минимально объемлющим прецеденты этого класса. При поиске подходящего класса значения признаков текущего случая сравниваются с проекциями границ классов на пространство этих признаков. В общем случае при таком сравнении выделяется сразу целый ряд классов, в которые попадает исследуемый случай, для работы выбираются те прецеденты, которые находятся в любом из них.

В описании случая может присутствовать большое количество признаков, на практике их может быть до нескольких десятков, поэтому четко разграничить классы, в которые попадает случай, удается не всегда. Одной из наиболее частых причин этого является недостаток информации в его описании. В подобных случаях среди признаков выделяют *существенные*. Может также возникнуть ситуация, в которой текущий случай имеет набор признаков, не пересекающийся с наборами признаков, ранее введенными в систему. Иногда такая ситуация возникает вследствие того, что некоторые несущественные для данного случая признаки были сознательно отброшены и не включены в рассмотрение. Однако некоторые другие признаки, являющиеся существенными по отношению к некоторым из имеющихся классов, у текущего случая по разным причинам могут также отсутствовать. Вследствие этого, не полностью описанный случай может попасть в пересечение классов (то есть сразу в несколько классов одновременно), другими словами, оцениваться неоднозначно только потому, что у него не хватает признака, который дифференцировал бы его от других классов. В зависимости от сложности пересечения классов, в которое попал исследуемый случай, прецеденты делятся на группы. Те прецеденты, что находятся во внутренней области пересечения, естественно считать наиболее близкими к текущему случаю, их в проводящемся анализе ситуации необходимо рассматривать в первую очередь, обращаясь к другим (временно отложенным прецедентам) только в случаях крайней необходимости.

Важным свойством создаваемой системы поддержки принятия решений является возможность использования опыта нескольких экспертов одновременно. С технической стороны это приводит к возможности создания распределенных конфигураций системы, работающих в локальных и/или глобальных общедоступных или специализированных информационных сетях. В то же время сохраняется возможность создавать изолированные установки, работающие в локальном окружении, без каких-либо контактов с другими аналогичными установками системы. В системе, предназначенной

для персонального использования на локальном компьютере, каждый пользователь хранит описания встретившихся именно ему прецедентов, формирует собственную базу прецедентов, не имея доступа к опыту других пользователей. Однако естественным расширением возможностей системы было бы привлечение опыта, накопленного другими пользователями в подобных системах. Этот опыт материализован в данных, хранящихся в других установках системы. Обмен данными между аналогичными системами, по сути, представляет собой обмен накопленными знаниями.

Под распределенной системой поддержки решений будем понимать совокупность локальных систем, между которыми имеется возможность обмена данными. Метод пересылки данных при нашем рассмотрении архитектуры распределенной системы значения не имеет. Для передачи параметров запроса могут быть использованы разные технические методы: от обмена файлами на физическом носителе – до возможностей информационных сетей.

Обмен может осуществляться двумя способами. Первый вариант – запрос из базовой системы в удаленный компьютер. Параметры запроса – показатели текущего случая (Рис. 1): имя системы и описание случая, то есть значения его признаков.

Рис. 1. Оценка текущего случая с помощью запроса к удаленному компьютеру.

В ответ на запрос система с удаленного компьютера сообщает базовой системе схожие прецеденты вместе с описаниями их классов:

- Имя системы (необязательный параметр сообщения)
- Группы описаний классов, каждая из которых содержит
  - имя класса
  - границы класса по признакам
- Группы описания прецедентов, каждая из которых содержит
  - имя случая
  - признаки случая

Пользователь не обязан знать, на каком компьютере данные хранятся физически. В запросе, адресованном конкретной системе в сети, где присутствуют несколько подобных систем, имеется возможность указывать имя системы, но если имя не указано, это означает, что запрос выдается ко всей сети в целом. Системы, где выявлены описания схожих случаев, выдают ответ на запрос в указанном формате. Чтобы базовая система не ждала ответа от сети бесконечно долго, остальные выдают пустой ответ.

В базовой системе формируется картина из прецедентов и классов, полученная в результате объединения всей имеющейся и полученной в ответ на запрос информации (Рис. 2), так, как если бы все данные хранились в одной системе.

Такой способ обмена будем называть *виртуальной интеграцией*, так как реального пополнения базы прецедентов базовой системы при рассмотрении конкретного случая не происходит. Виртуальная интеграция позволяет проводить на практике оценку текущего случая, принимая во внимание опыт, накопленный в каждой из локальных систем. В классической медицине, например, аналогом такой интеграции является консилиум врачей, когда мнение другого врача может помочь в оценке картины заболевания. Объединение данных из нескольких систем при рассмотрении сложного случая позволяет существенно повысить качество принимаемого решения так же, как и в случае консилиума.

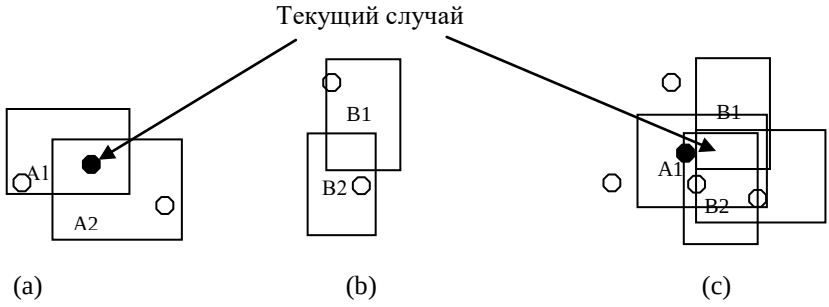


Рис. 2. Оценка текущего случая методом виртуальной интеграции

a) в базовой системе (A) до интеграции, два прецедента из классов A1 и A2

b) в удаленном компьютере (B), два прецедента из классов B1 и B2

c) в базовой системе (A) после интеграции.

В частном случае, в базовой системе вообще могут отсутствовать прецеденты и классы, необходимые для оценки текущего случая. Тогда виртуальная картина будет собираться из того, что получено по запросу из удаленных компьютеров.

Реально картина на экране компьютера не выглядит так просто, как на *рис. 2*. Многомерный случай размерностью больше двух невозможно достаточно наглядно представить двумерной проекцией. Наиболее информативный вариант – представление в виде так называемой реляционной модели, в виде нескольких взаимосвязанных таблиц. Покажем это на примере системы поддержки врачебных решений в диагностике и выборе лечения «Спутник Врача», разрабатываемой в Институте системного программирования РАН в рамках текущего проекта.

На Рис. 3 показаны прецеденты, найденные системой для смоделированного случая «симптомы острого живота» (левый верхний угол). Источником двух первых прецедентов является базовая система (которую заполнял абдоминальный хирург, специализирующийся на лечении органов брюшной полости), третьего – удаленный компьютер (заполнял пульмонолог – специалист, занимающийся лечением заболеваний лёгких и дыхательных путей).

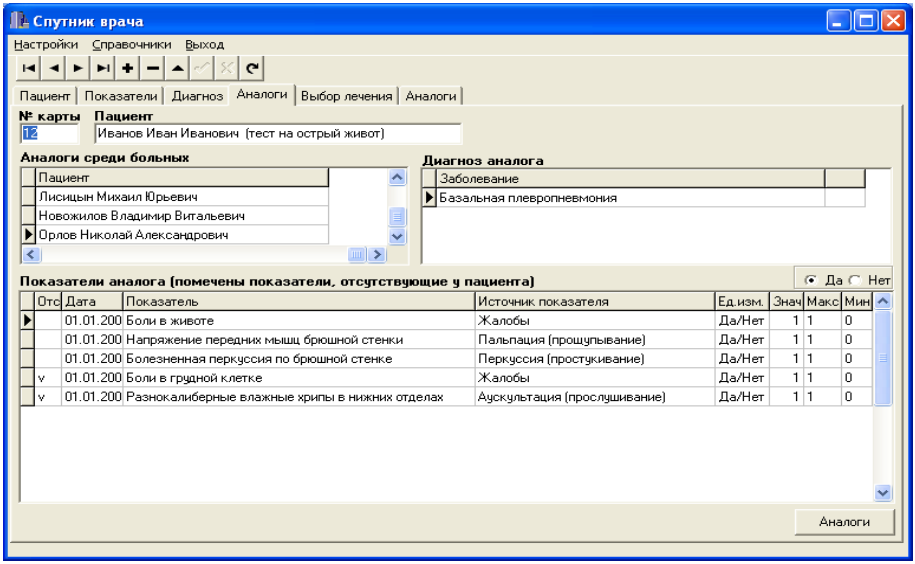


Рис. 3. «Спутник врача». Прецеденты с симптомами «острого живота».

Виртуальная интеграция имеет смысл, когда информация из удаленной системы может оказать дополнительную помощь в диагностике, но не относится прямо к области деятельности пользователя, ведущего работу с базовой системой. При рассмотрении последующих случаев полученная информация более участвовать не будет (виртуальная интеграция всегда должна проводиться заново при рассмотрении каждого нового случая).

Если же зафиксированный случай оказался более интересным, и пользователь базовой системы предполагает, что в дальнейшем он будет сталкиваться с подобными случаями повторно, ему будет полезнее импортировать всю или определенную часть прецедентов удаленной системы в свою базу прецедентов. Тогда он решает позаимствовать из нее часть прецедентов вместе с описаниями их классов, либо некоторые классы целиком. Возникает второй вариант обмена – *консолидация знаний*. В этом случае осуществляется доступ не к сервису, а к данным удаленного компьютера (*рис. 4*).

Консолидация – это не просто импорт части случаев из одной базы прецедентов в другую, но и сопутствующая ему реорганизация базы прецедентов. Импорт случаев из базы может выполняться как в автоматизированном, так и в ручном режиме. Прецеденты и классы из удаленной системы копируются в базовую систему, что приводит к необходимости переопределения границ классов с учетом возможных коллизий. Реорганизация может привести к выявлению новых классов, ранее отсутствовавших в базовой системе.

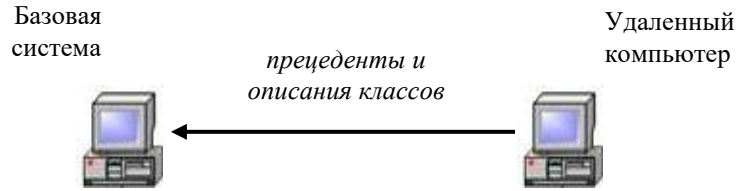


Рис. 4. Импорт прецедентов из удаленного компьютера.

Пользователь базовой системы может контролировать процесс импорта. Сначала на экране системы высвечивается картина, отражающая содержимое, полученное по запросу от удаленного компьютера (Рис. 2b). Оператор помечает прецеденты, которые нужно импортировать, затем переключается в интегрированный режим для просмотра ожидаемого результата (Рис. 2c). При необходимости цикл повторяется многократно.

Оба рассмотренных варианта имеют очень важную модификацию, существенным образом влияющую на процесс взаимодействия систем между собой. Эта модификация процесса распространения информации между системами состоит в том, что в качестве одного из источников дополнительных данных может выбираться крупный информационный центр коллективного доступа, который содержит выверенные данные, поступающие из локальных систем (Рис. 5).

Заполнение коллективной базы прецедентов не может осуществляться путем простого импорта данных. В данном случае импорт должен в обязательном порядке сопровождаться *валидацией* описаний случаев, то есть

синхронизацией терминологии, уточнением численных и логических показателей, отсеиванием сомнительных и откровенно ошибочных данных.

Чтобы обеспечить возможность удаленного доступа пользователей к системе и облегчить его технически, как в локальных, так и в глобальных сетях (в том числе в сети Интернет), при разработке программного обеспечения предлагается использовать совокупность стандартов, на которых базируются современные сетевые службы. Изучение тенденций развития сетевых служб показывает, что их разработчики все чаще стремятся использовать единые подходы, независимо от того, предполагается ли работать только в локальных сетях, или возможен выход и в глобальные сети (корпоративные или общедоступные).

На примерах взаимодействия нескольких установок разрабатываемой системы поддержки принятия решений можно проследить аналогию с организацией распределенных баз данных и связанным с ними понятием *репликации*. Общим для этих понятий является то, что система поддержки принятия решений должна позволять управлять распределенными данными таким образом, чтобы эта распределенность была прозрачна для пользователей. Прозрачность доступа означает, что пользователи осуществляют доступ к распределенным данным точно так же, как если бы они хранились централизованно.

Рис. 5. Совместное использование локальных систем и информационных центров коллективного доступа.

Отличия рассматриваемых понятий – значительные. Распределенные системы поддержки принятия решений – самодостаточные системы, созданные для использования независимыми пользователями. В этих системах отсутствует разделение приложений и базы данных по узлам приема запросов и узлам данных. Словари данных устроены однотипно, но создаются они независимо:

классы, одинаковые по сущности, в разных системах могут иметь разное обозначение и несовпадающие границы.

Если репликация – это процесс приведения данных электронных таблиц двух баз данных в идентичное состояние, то обмен данными в системах поддержки принятия решений лучше назвать *управляемым копированием*. Если репликация в распределенных базах данных может быть однонаправленной или мультинаправленной, то здесь обмен всегда однонаправленный. По старшинству локальные системы никак не выделяются: все взаимодействующие системы считаются равноправными, обмен может происходить между любой парой из них.

Для распределенных баз данных существует проблема возобновления работы процесса репликации при потере связи и последующем ее восстановлении. Здесь же данные обладают свойством идемпотентности: обмен данными происходит одним файлом, при обрыве связи передача просто повторяется.

Репликация в базах данных может быть синхронной или асинхронной. Процесс обмена в системах принятия решений всегда синхронный.

Проблема справочников и генерации идентификаторов не столь важна в системах принятия решений. Конфликты тоже возникают, но в основном – в описаниях классов и случаях отнесения к тому или иному классу. Различия в обозначении и границах классов не имеют большого значения. Наоборот, они позволяют пользователю сопоставить свой и чужой опыт при оценке текущего случая. Эти конфликты могут разрешаться либо на уровне алгоритма обмена, использующего правила административного старшинства (например, базовая система считаются приоритетной), либо, руководствуясь наиболее поздним по времени событием становления класса, либо на уровне вмешательства ответственного пользователя - администратора базовой системы. Но даже при избытке первых двух факторов, все равно невозможно разрешить все конфликты без участия администратора.

Литература

[1] Карпов Л. Е., Юдин В. Н. Методы добычи данных при построении локальной метрики в системах вывода по прецедентам // Институт Системного Программирования РАН, Препринт, 2006.

[2] Карпов Л. Е., Юдин В. Н. Адаптивное управление по прецедентам, основанное на классификации состояний управляемых объектов // Труды Института Системного Программирования РАН, т.13, ч.2. М., 2007, стр. 37-57.

[3] Карпов Л. Е., Юдин В. Н. Интеграция методов добычи данных и вывода по прецедентам в медицинской диагностике и выборе лечения // Сб. докладов 13-й Всероссийской конференции «Математические методы распознавания образов (ММРО-13)», М., 2007, стр. 589-591.

[4] Карпов Л. Е., Юдин В. Н. Система поддержки принятия решений для практикующих врачей // Ежегодная техническая конференция «Корпоративные базы данных-2008», Электронный ресурс. Режим доступа: http://www.citforum.ru/seminars/cbd2008

[5] Юдин В. Н., Карпов Л. Е., Ватазин А. В. Методы интеллектуального анализа данных и вывода по прецедентам в программной системе поддержки врачебных решений // Альманах клинической медицины, т.17, часть 1. М., 2008, стр. 266-269.
[6] Ватазин А. В., Карпов Л. Е., Юдин В. Н. Виртуальная интеграция и консолидация знаний в распределенной системе поддержки врачебных решений // Альманах клинической медицины, т.20, М., 2009, стр. 83-86.
[7] Watson I., Marir F. Case-Based Reasoning: A Review, The Knowled. Engineer. A Review, 1994. V. 9. No.4.

Многопараметрическое управление на основе прецедентов

Карпов Л. Е., Юдин В. Н.

Аннотация. Ведущаяся разработка связана с решением задачи многопараметрического управления объектом со сложным взаимным влиянием воздействий в ситуациях, когда влияние внешних факторов может оказаться взаимозависимым и даже противоречивым, когда трудно или невозможно получить точную модель поведения объекта. Предлагается подход к математической формализации понятия управления на теории принятия решений, методов добычи данных и вывода по прецедентам.

1. Введение

Построение системы управления, основанной на учете сразу многих видов внешних воздействий, не сводится к простому построению нескольких систем, совместно решающих общую задачу. Во-первых, влияние внешних факторов может оказаться взаимозависимым, и даже противоречивым, а во-вторых, достаточно часто могут встречаться ситуации, когда среди внешних факторов имеются как зависящие друг от друга воздействия, так и независимые. Другим аспектом, усложняющим построение многопараметрических систем управления, является отсутствие возможности связать с каждым элементарным внешним воздействием ровно один параметр внутреннего состояния системы, на которой это воздействие осуществляется. Остается и даже увеличивается та часть проблем построения систем управления, которая связана со сложностью формализации внешних воздействий и состояний управляемого объекта.

Одна из наиболее простых стратегий управления – реагировать на события по мере их появления – носит название «замкнутого управления» (Рис. 1), или управления с обратной связью, при котором предполагается возможность изменять управление в зависимости от его воздействия на конечный результат.

«Адаптивное управление» отличается от замкнутого наличием модели управляемого объекта (Рис. 2), в которой анализируются возможные последствия управления (прогноз). Правильная реакция возможна лишь при наличии формализованного описания, которое называют «математической моделью» объекта, адекватно отображающей среду функционирования и сам объект управления.

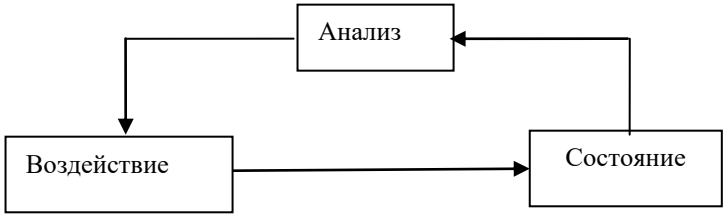


Рис. 1. Структура замкнутого управления.

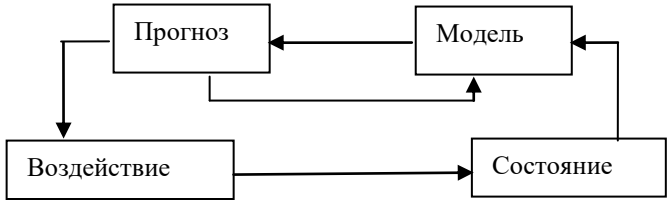


Рис. 2. Структура адаптивного управления.

При недостаточности наших знаний об объекте и среде, в которой он функционирует, не представляется возможным получить точную модель поведения объекта. Вместо этого мы владеем только априорной информацией о состояниях объекта, управляющих воздействиях на него и результатах воздействий. В таких случаях предлагается иная структура адаптивного управления на основе прецедентов (Рис. 3).

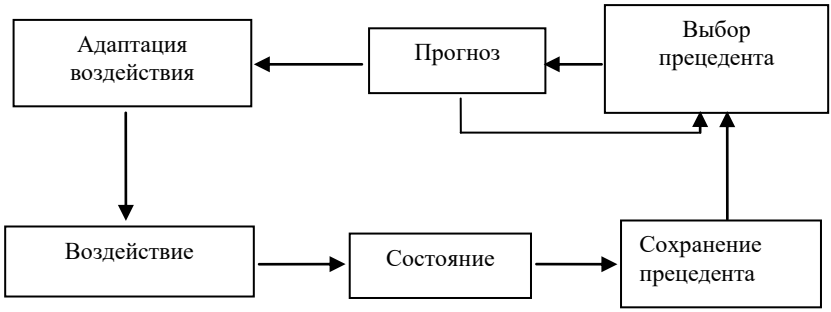


Рис. 3. Схема адаптивного управления по прецедентам.

Ведущаяся в ИСП РАН разработка связана с решением задачи многопараметрического управления процессом со сложным взаимным влиянием воздействий, для которого трудно или невозможно получить точную модель поведения. Разработкой предусматривается построение концептуальной системы управления, работающей в условиях многочисленных, в том числе не зависящих друг от друга внешних воздействиях, влияющих на разные параметры состояния объекта управления. Задача новой системы состоит в том, чтобы дать возможность адекватно учитывать многопараметрические внешние воздействия на объект управления со сложным, непараметризуемым поведением.

Группой разработчиков уже выработаны методика многопараметрического управления плохо формализуемым объектом (процессом) на основе прецедентов, методика разбиения состояний объектов на классы, проведен выбор методики оценки схожести с прецедентом на основе учета входных и целевых факторов, а также разработаны принципы построения программной системы управления на основе разработанной методики.

Управление многомерным процессом подразумевает целенаправленное изменение множества параметров состояния и структуры объекта на основе выработанной стратегии, методов и алгоритмов управления. При попытках управления объектами, для которых не удаётся построить адекватную математическую модель, возникает ситуация, принципиально отличающаяся от «классической». В таких ситуациях вместо точного вида математической модели объекта доступна только априорная информация о состояниях объекта управления, управляющих воздействиях на него и результатах воздействий (прецеденты). Одним из примеров таких объектов является организм человека, законы функционирования которого до конца ещё не известны.

В осуществляемой разработке используется подход, в котором целенаправленность управляющих воздействий базируется на прецедентах. В терминах вывода по прецедентам, информация о состоянии объекта – это описание проблемы, а выдача управляющего воздействия есть решение проблемы, тогда результат воздействия – это результат применения решения.

2. Понятие прецедента в управлении

Схематически любой шаг управления объектом можно представить как совокупность трех элементов (Рис. 4) – состояние объекта до воздействия (c_i), управляющее воздействие (e_i) и состояние объекта после воздействия (c_{i+1}). Эту тройку составляющих обычно называют «прецедентом», или «случаем».

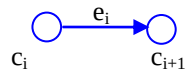


Рис. 4. Элементарный шаг процесса управления.

Состояние объекта описывается набором признаков, причем допускаются как числовые, так и логические признаки. Управляющие воздействия, в зависимости от конкретного приложения, могут иметь разный формат, вплоть до описаний в виде текстового комментария. В технике внешние воздействия могут проявляться, например, в изменениях параметров внешней среды – атмосферного давления, влажности воздуха, его прозрачности и так далее. В медицине воздействие может быть элементом проведенного лечения, например, в качестве воздействия могут восприниматься прием лекарственных средств, выполненные процедуры, а также оперативное вмешательство в работу организма человека.

Случаи, отражающие всю хронологию воздействий на отдельный объект, связываются в так называемую «цепь управляющих воздействий». Эту последовательность состояний и воздействий можно представить в виде цепи, вершины которой – состояния объекта, а дуги – управляющие воздействия (Рис. 5).

Рис. 5. Цепь элементарных шагов процесса управления.

Такие последовательности состояний и воздействий могут возникать также и по воле управляющего объектом. Например, в медицине в тяжелых случаях принято воздействовать на пациента не сразу, а последовательно приближаясь к норме, то есть, задавая не одно следующее воздействие, а назначая сразу серию последовательных воздействий, каждое из которых лишь немного приближает состояние объекта к желаемому.

Состояние «после воздействия» одновременно является состоянием «до следующего воздействия», поэтому цепь представляет собой совокупность пар «состояние-воздействие». Совокупность состояний объектов и воздействий на них образует так называемую «базу прецедентов».

Значительная часть подходов к управлению, в частности, управление физическими объектами, строится на предположении о жёсткой взаимосвязи отдельных входных воздействий на управляемые объекты с отдельными параметрами состояний этих объектов.

Особенностью разрабатываемой в ИСП РАН системы будет способность адекватно учитывать многопараметрические внешние воздействия на объект управления со сложным взаимным влиянием воздействий. Система позволит управлять объектом в условиях совмещения ряда, возможно, противоречащих, воздействий. В отсутствие математической модели объекта выбор воздействий осуществляется по прецедентам. Чем больше прецедентов в базе, тем больше спектр их возможных значений, тем выше вероятность найти «наиболее подходящий» прецедент и выше качество принимаемого решения.

Обычно состояние объекта представляется набором признаков. Позиционирование состояния объекта до воздействия по отношению к описанным классам производится в его признаковом пространстве. Соответствующая объекту точка сравнивается с расположением классов в проекции на пространство его признаков. Недостаточно полно описанные объекты могут неоднозначно позиционироваться, то есть попадать в область пересечения классов, один из которых – исходный.

На следующем шаге процесса управления выбираются воздействия, которые уже когда-то применялись к сходным состояниям, и сравниваются состояния «после воздействия» в том же пространстве признаков. Прецеденты, в которых достигается нужный класс, считаются наиболее предпочтительными (Рис. 6).

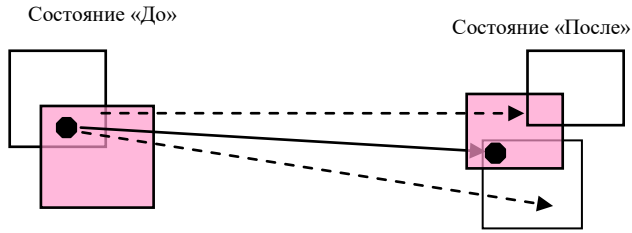


Рис. 6. Отбор прецедентов.

При отборе прецедентов могут встретиться случаи, наборы признаков которых в начальном состоянии совпадают, но для которых одно и то же воздействие приводит к разным исходам, то есть к разным конечным состояниям (Рис. 7). Наличие таких случаев в одной базе прецедентов связано с тем, что иногда состояние объекта описывается недостаточно полно, то есть в этих случаях присутствует неучтенный разделяющий признак. Если бы этот признак был учтен, совпавшие исходные состояния трактовались бы как разные еще при занесении их в базу прецедентов. По этой же причине для не полностью описанных случаев невозможно гарантировать, что воздействие, позаимствованное у прецедента, приведет в нужное состояние.

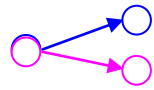


Рис. 7. Слипание состояний из-за неполноты описания случая.

3. Многопараметрические воздействия

В реальности объект управления рассматривается как одиночный. При этом воздействие на этот одиночный объект может быть представлено в виде ряда составляющих по разным признакам или группам признаков (параметров состояния), по которым осуществляется контроль и управление. Значительная часть подходов к управлению, в частности, управление физическими объектами, строится на предположении о независимости воздействий по каждому такому признаку. Это позволяет рассматривать управление по каждому признаку отдельно. В такой трактовке воздействия, влияющие на какой-либо один параметр состояния, никак не влияют на другие параметры.

В условиях независимости воздействий по каждому параметру управление объектом удобней рассматривать в виде нескольких непересекающихся цепей (Рис. 8).

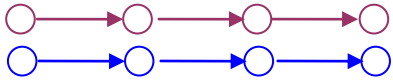


Рис. 8. Цепи управления по двум независимым параметрам.

Наличие нескольких цепей управления есть лишь определенная абстракция, в действительности же существует одна многопараметрическая цепь состояний и воздействий, а рассмотрению на самом деле подлежит один сложный объект.

Такое разделение удобно и оправданно, например, в медицине исторически сложилось, что врач определенной специализации контролирует определенный набор признаков, наиболее характерных для своей (хорошо им изученной) группы заболеваний. В технике механики и электрики также контролируют совершенно разные параметры состояний объектов управления.

Но условие независимости воздействий не всегда выполнимо. Для человеческого организма с его сложными внутренними связями и большими компенсаторными возможностями воздействие по одному признаку может проявляться через другие признаки. И тогда, чтобы компенсировать эффект первого воздействия, приходится применять дополнительные воздействия по другим группам признаков. Иногда влияние подобных воздействий носит взаимозависимый и даже взаимоисключающий характер.

Приведем пример из медицины. При интенсивной терапии при серьезных патологиях в условиях нарушенного водно-электролитного баланса у больного возникает опасность избытка жидкости в организме. Контролируется набор показателей: (повышенное) артериальное давление, застойное полнокровие в легких, наличие периферических и полостных отеков и другие осложнения, связанные с избыточным объемом жидкости.

Для выравнивания состояния больного в качестве управляющих воздействий применяются различные методы дегидратации, что приводит к снижению артериального давления, снижению перфузии почек, появлению отеков, снижению уровня белка в крови. Такие эффекты, в свою очередь, можно нивелировать исключением натрия и калия, введением белка (альбумина) в организм. Как видно, воздействие по одной группе параметров сказывается на другой. Удержать ситуацию в некоем коридоре – задание трудновыполнимое.

Можно представить себе различные виды взаимного влияния воздействий, такое влияние чаще всего непараметризуемо.

Рассмотрим схематически состояние объекта по двум группам признакам, по которым может осуществляться отдельный контроль. Обозначим через c_{11} начальное состояние объекта по первой группе, c_{21} – по второй. Предположим, воздействие e_1 переводит объект в состояние c_{12}, оно же приводит к изменению состояния объекта в c_{22}. На схеме это можно обозначить как дополнительное воздействие от c_{21} к c_{22} (Рис. 9(a)), либо как разветвленное воздействие от c_{11} к c_{22} (Рис. 9(b)). И, наконец, возможны взаимные влияния по обеим цепям (Рис. 9(c)).

Наличие точки разветвления говорит о том, что управляющее воздействие привело к необходимости оценивать состояние объекта по двум цепям. Возможны и более сложные переплетения взаимных влияний воздействий по разным цепям (Рис. 9(d)).

В самом общем случае, прецедентом будет одномоментная совокупность состояний объекта до воздействия, вся совокупность воздействий плюс совокупность состояний после множественного воздействия по всем группам признаков.

При наличии нескольких цепей управления задача выбора воздействия предполагает одновременное решение нескольких подзадач, аналогичных той, что приведена на рис. 6. Хороший прогноз воздействия, особенно многофакторного, возможен только при наличии в базе достаточного числа прецедентов, покрывающих весь спектр возможных взаимовлияний. В ней может содержаться или отсутствовать информация о влиянии на какие-либо группы признаков, о взаимном влиянии воздействий. Отсутствие в базе сведений о влиянии каких-либо воздействий на состояние объекта управления не обязательно означает отсутствие влияния этих воздействий на объект. Оно всего лишь означает, что в базе прецедентов не накоплена соответствующая информация. В подобных случаях в соответствии с часто используемым в науке подходом приходится рассматривать воздействия, сведений о которых недостаточно, как независящие друг от друга.

С другой стороны, иногда можно явно указать на отсутствие влияния какой-либо группы признаков на другую. Приведем наглядный пример из медицинской практики. Средство для лечения остеопороза – препарат, регулирующий фосфорно-кальциевый обмен – миакальцик – обладает рядом побочных эффектов, среди которых – реакция гиперчувствительности: кожная

сыпь, повышение артериального давления и периферические отеки. Этот негативный эффект можно предотвратить упреждающими воздействиями, даваемыми независимо по каждой цепи: приемом препаратов для снижения давления, противоаллергическими препаратами, снижением натрия в пище. Обратного влияния на первое воздействие эти воздействия не имеют. Поэтому можно воспользоваться прецедентами из базы, отражающими влияние каждого из этих препаратов в отдельности, вне их связи с первым.

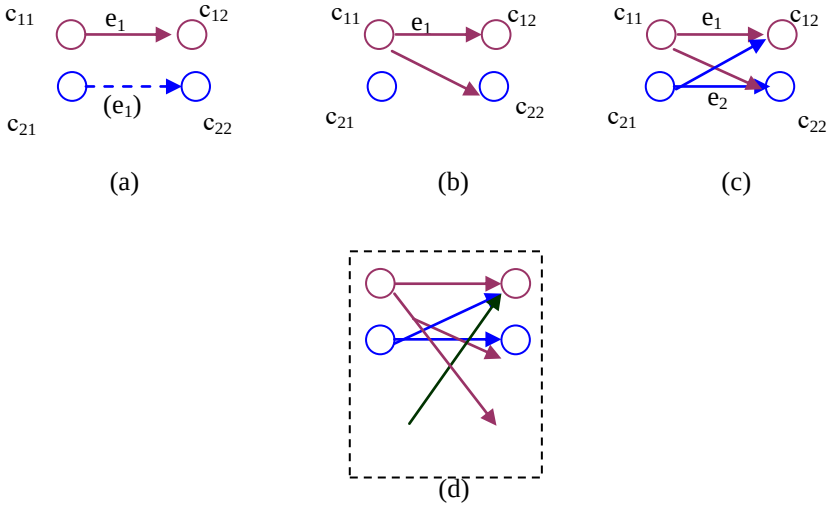


Рис. 9. Взаимное влияние различных воздействий на один объект управления.

Поиск золотой середины, то есть оптимального воздействия при выборе многофакторного воздействия – задача сложная, недостаточно формализованная и сильно зависящая от предметной области, даже внутри такой области, как медицина, поэтому на данном этапе она оставлена человеку, являющемуся экспертом в предметной области.

Уменьшение числа показателей при многофакторном управлении, своеобразная регрессия – упрощает задачу выбора воздействия. В примере с нарушением электролитного баланса можно ввести общий показатель, характеризующий состояние больного – внутритканевое сопротивление, условно характеризующее наполнение тканей жидкостью. Но, к сожалению, такое решение возможно не всегда.

Отсутствие подходящего прецедента приводит к тому, что при выборе воздействия приходится руководствоваться случайными критериями, или просто прекращать управление объектом до тех пор, пока он самопроизвольно не перейдет в состояние, для которого прецедент будет обнаружен. В

результате управление превращается из адаптивного – в замкнутое управление или управление без прогноза (Рис. 1 и 2). Когда управление осуществляется без прогноза, остается простая реакция на изменение ситуации, заставляющая объект совершать маятникообразные движения без приближения к цели.

Если воздействие по первой группе привело к непредвиденному изменению состояния по второй, можно поступить несколькими способами. Вот некоторые из них:

- воздействовать на вторую группу, чтобы нейтрализовать нежелательное изменение (нет уверенности, что это не повлияет на первую группу);
- убавить первое воздействие или применить противоположное ему (полностью или частично) с целью убрать его влияние на вторую группу (нет уверенности, что будет получен изначально желаемый результат по первой группе).

Во всех случаях необходимо искать оптимум («золотую середину») в величине воздействий, чтобы добиться приемлемого результата.

4. Выбор управляющего воздействия

Основная проблема в любом выводе, основанном на прецедентах, – выбор наиболее подходящих прецедентов, который упирается в оценку схожести прецедента и текущего случая. Выбор прецедентов связан с выбором параметров состояния объекта управления, которые могут быть разными в разных системах управления, однако он связан также не только с выбором набора (номенклатуры) воздействий, которые могут иметь объективный характер, но и с определением некоторых их параметров, например, силы воздействия, дозы лекарства и т. д. При отборе прецедентов можно использовать предварительно накопленные знания о предметной области. В ситуации, когда известных параметров объекта и окружающей среды недостаточно для полного и однозначного определения его поведения, принимать решение о нужном на очередном шаге управляющем воздействии на объект, зная только его параметры, нельзя. Знание поведения объекта будет полнее и точнее, когда управление осуществляется не по измеряемым извне параметрам его поведения, а по его состояниям. Если на основе априорной информации удается сформировать обобщенные образы – классы состояний, когда реакция объекта, находящегося в состоянии, принадлежащего любому такому классу, известна, то управляющее воздействие можно рассматривать как отображение объекта управления из одного класса в другой. Именно поэтому состояния объектов разбиваются на классы, состояния в каждом из которых настолько близки друг другу, что рассматриваются с точки зрения управления как эквивалентные.

Цель управления – достижение нужного состояния объекта. Как начальное, так и конечное состояния описываются вхождением в соответствующие

классы. Цель отдельного шага управления – перевод объекта из текущего класса в требуемый класс (в частности, целью может быть удержание состояния в том же классе). Для достижения этой цели необходимо, основываясь на состоянии объекта, выбирать наиболее приемлемое из возможных управляющее воздействие. Для прогноза вместо математической модели объекта используется накопленная ранее база прецедентов.

Схематически такая взаимосвязь может быть проиллюстрирована таким образом:

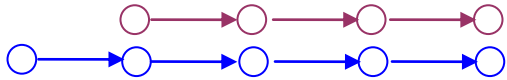


Рис. 10. Цепи управления по независимым параметрам.

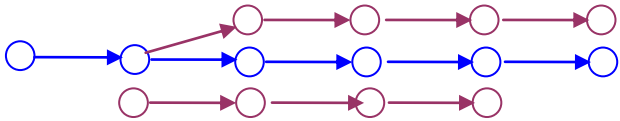


Рис. 11. Взаимосвязь цепей управления по разным параметрам.

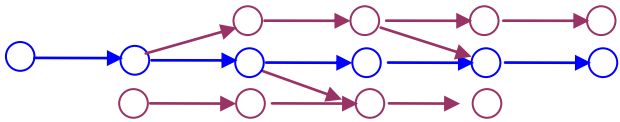


Рис. 12. Сложная взаимосвязь цепей управления по разным параметрам.

Воздействия, влияющие на какой-либо один параметр состояния, могут быть независимыми, то есть никак не влиять на другие параметры (Рис. 10). Однако на практике можно наблюдать взаимосвязь входных воздействий и параметров состояния (Рис. 11).

Некоторые входные воздействия действительно связаны с определенными параметрами состояния объектов, в то время, как другие воздействия могут

влиять на значения сразу нескольких параметров, что серьезно усложняет процесс управления. Возможны и более сложные переплетения взаимных влияний элементарных воздействий и параметров состояний (Рис. 12).

В медицине для такого сложного объекта, каким является человеческий организм, со сложными внутренними связями и большими компенсаторными возможностями, воздействие, соответствующее одному параметру неизменно проявляется в виде изменений, за которые считаются ответственными воздействия, соответствующие другим параметрам. Чтобы компенсировать нежелательный эффект одного воздействия, часто приходится применять другие воздействия. Иногда влияние подобных воздействий носит взаимоисключающий характер. Так, ряд лекарств, предназначенных для снижения артериального давления (например, лозап), одновременно снижает гемоглобин крови. В свою очередь лекарства, повышающие уровень гемоглобина (например, эритропоэтин), повышают артериальное давление. Другой пример – пересадка органов в условиях присоединившегося туберкулеза: взаимоисключающее влияние иммуносупрессорной терапии и противотуберкулёзных средств. Среди возможных решений проблемы – выбор уровня воздействия (в зависимости от состояния объекта, воздействие может лежать в допустимых, равновесных, или кризисных границах) и/или расширение признакового пространства (иногда дополнительное исследование позволяет понять причину реакции на то или иное воздействие).

5. Диаграмма «Сущности-Связи» для базы прецедентов

Основная часть базы прецедентов – совокупность объектов, их состояний и воздействий на них. Кроме этого, в базе хранятся дополнительные структуры, создаваемые над этой совокупностью, в виде описаний классов.

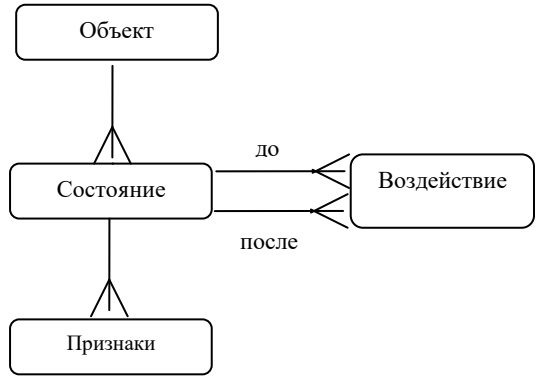
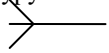


Рис. 13. Модель Сущности-Связи базы прецедентов.

Представим структуру базы прецедентов в виде диаграммы Сущность-Связь (Рис. 13), где связь  представляет отношение многие-к-одному.

Объект представлен совокупностью состояний. В свою очередь каждое состояние описывается совокупностью признаков. Воздействие имеет ссылки на состояние «до» и состояние «после» воздействия. С помощью пары этих ссылок можно проследить всю цепь управляющих воздействий на объект.

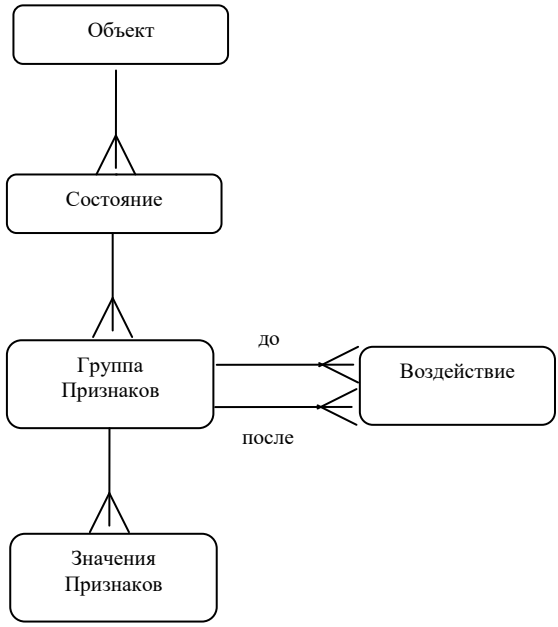


Рис. 14. Модель Сущности-Связи базы прецедентов в условиях нескольких цепей управления.

6. Организация базы прецедентов многопараметрического управления

Модель Сущности-Связи базы прецедентов, показанная на Рис. 13, в условиях наличия нескольких цепей управления (или многопараметрического управления) изменяется (Рис. 14).

Объект теперь представляется множеством состояний (в разные моменты времени), которым соответствуют сразу несколько групп признаков. Воздействие применяется не к сущности Состояние, а к некоторому виртуальному состоянию, описываемому группой признаков.

Литература

[1] V.N. Yudin. Applying Cluster Analysis for Searching for Analogs in Diagnostics and Choice of Treatment // Pattern Recognition and Image Analysis, Vol. 13, No. 4, 2003, pp 706-713.

[2] Карпов Л. Е., Юдин В. Н. Адаптивное управление по прецедентам, основанное на классификации состояний управляемых объектов // Труды Института Системного Программирования РАН, т. 13, ч.2. М., 2007, стр. 37-57.

[3] Карпов Л. Е., Юдин В. Н. Интеграция методов добычи данных и вывода по прецедентам в медицинской диагностике и выборе лечения // Сб. докладов 13-й Всероссийской конференции "Математические методы распознавания образов (ММРО-13)". М., 2007, стр. 589-591.

[4] Карпов Л. Е., Юдин В. Н. Система поддержки принятия решений для практикующих врачей // Ежегодная техническая конференция «Корпоративные базы данных-2008», Электронный ресурс. Режим доступа: http://www.citforum.ru/seminars/cbd2008

[5] Юдин В. Н., Карпов Л. Е., Ватазин А. В. Процесс лечения как адаптивное управление человеческим организмом в программной системе «Спутник Врача» // Альманах клинической медицины, т.17, часть 1. М., 2008, стр. 262-265.

[6] Юдин В. Н., Карпов Л. Е., Ватазин А. В. Методы интеллектуального анализа данных и вывода по прецедентам в программной системе поддержки врачебных решений // Альманах клинической медицины, т.17, часть 1. М., 2008, стр. 266-269.

[7] Ватазин А. В., Карпов Л. Е., Юдин В. Н. Виртуальная интеграция и консолидация знаний в распределенной системе поддержки врачебных решений // Альманах клинической медицины, т.20, М., 2009, стр. 83-86.

[8] Klaus-Dieter Althof, Eric Auriol, Ralph Barlette, and Michel Manago. // A Review of Industrial Case-Based Reasoning Tools, AI Intelligence, 1995.

Объектно-ориентированное программирование в ограничениях: новый подход на основе декларативных языков моделирования данных

Семенов В.А., Ильин Д.В., Морозов С.В., Сидяка О.В.

Аннотация. Объектно-ориентированное программирование в ограничениях (ООСР) сочетает две ортогональные, но комплементарные парадигмы программирования, а именно: объектно-ориентированное программирование (ООР) и логическое программирование в ограничениях (CLP). Несмотря на привлекательность идеи синтеза парадигм и известные попытки реализации, до сих пор не существует единого понимания, какие конструктивные очертания она может приобрести при дальнейшей проработке и развитии. Ключевыми вопросами при этом остаются выразительность описания прикладной задачи в ограничениях и ее алгоритмическая разрешимость. В настоящей работе предлагается и обсуждается новый системный подход к реализации ООСР на основе использования декларативных языков моделирования данных.

1. Введение

В последние годы наблюдается ренессанс идей логического программирования в ограничениях (CLP) в таких актуальных научных областях и дисциплинах, как интеллектуальные системы принятия решений (логические нейронные сети), компьютерная графика (декларативные сценарные модели и графические интерфейсы), экономические модели (недоопределенные вычисления), системы автоматизированного проектирования (параметрическое моделирование), информационные системы (активные базы данных и управление целостностью), семантический поиск данных (дескриптивные логики), верификация и тестирование программного обеспечения (временные логики), системы коллективной инженерии (полисиллогистический вывод) [1, 2]. Многие крупные индустриальные компании проявляют интерес к этой теме, а ACM признала ее одним из стратегических направлений исследований.

Вместе с тем, следует констатировать, что как технология общего назначения CLP не получило широкое распространение, а в перечисленных выше областях разрабатываются и используются специализированные языки и программные системы. Их главным недостатком и принципиальным

ограничением является невозможность специфицировать и решать задачи относительно переменных, являющихся сложно структурированными, семантически согласованными данными — в частности, объектно-ориентированными данными, определяемыми актуальными информационными моделями и международными стандартами ISO STEP [3], OMG MDA [4], W3C Semantic Web [5].

Эти факторы обуславливают выработку нового системного подхода, который бы, с одной стороны, обеспечил решение широкого класса задач в ограничениях, а с другой — приблизил способы их постановки к популярным универсальным технологиям объектно-ориентированного моделирования. Применение для этих целей универсальных языков моделирования данных EXPRESS [6], ODL/OQL [7], UML/OCL [8, 9], OWL [10], получивших признание и распространение в широких научных и индустриальных сообществах, приобретает особую привлекательность. В самом деле, прикладную задачу в ограничениях можно описать путем определения пользовательских типов данных и задания на них разнообразных семантических правил. Развитые средства алгебраической спецификации правил вместе с предопределенными конструкциями для задания областей значений переменных, типов и размеров коллекций, кардинальности объектных типов, обязательности атрибутов, свойств уникальности атрибутов, условий наличия или отсутствия ассоциативных циклов, позволяют сделать это относительно простым и наглядным образом.

В ряде случаев целесообразным представляется использование стандартных информационных моделей [11–14], разработанных индустриальными консорциумами для таких областей как программная инженерия, информационные технологии, машиностроение, атомная энергетика, авиационная и космическая промышленность, судостроение, архитектура и строительство, нефтегазовый комплекс, фармацевтика. Постановка и решение задач программирования в ограничениях с их применением может приводить к возникновению новых, индустриально значимых приложений. К числу таких приложений можно отнести, например, задачу тестирования интероперабельности приложений, осуществляющих обмен данными и функционирующих в составе интегрированных программных комплексов, задачу управления целостностью прикладных данных в развитых вычислительных и информационных системах, использующих оптимистические модели транзакций, или задачу верификации программной модели в соответствии с подготовленными контрактными спецификациями. Во всех упомянутых случаях речь идет о формировании (или согласованной модификации) коллекций структурированных данных по заданной спецификации объектно-ориентированной модели. Полученные данные должны при этом соответствовать заданной модели и удовлетворять ее семантическим правилам.

Естественно, что универсальность и декларативность перечисленных выше языков моделирования порождает проблему алгоритмической разрешимости систем ограничений, специфицированных на них. Идентификация математической задачи и выбор релевантного алгоритма решения при возможном переопределенном или недоопределенном характере системы ограничений являются общими проблемами для большинства подходов. Дополнительным фактором сложности, привносимым языками моделирования, является сам класс задач логического программирования в ограничениях на множествах объектно-ориентированных данных. Данный класс, обозначаемый в дальнейшем CLP(O), предполагает дополнительную структуризацию переменных по сравнению с традиционными постановками в булевых, рациональных, вещественных числах CLP(B), CLP(Q), CLP(R) и на конечных доменах CLP(FD) соответственно. Кроме того, возможность алгебраической спецификации произвольных семантических правил приводит к необходимости совместного анализа и разрешения неоднородных ограничений, что крайне затруднительно при использовании традиционных методов, ориентированных на частные математические постановки. Наконец, задание правил на типах данных, а не только на отдельных переменных, порождает проблему формирования множества неизвестных переменных, в данном случае — коллекций объектов и элементов данных, относительно которых задача в ограничениях должна решаться.

Отметим, что как самостоятельная научная дисциплина программирование в ограничениях охватывает три направления, а именно: разрешение ограничений CSP (Constraint Satisfaction Problem) [15], логическое программирование в ограничениях CLP (Constraint Logic Programming) [1] и конкурентное программирование в ограничениях CCP (Concurrent Constraint Programming) [16]. Несмотря на особенности в постановках и методах решения задач, все три направления тесно связаны между собой. В рамках обсуждаемого подхода к ООСР мы не видим причин отказываться ни от одного из них. Применяя обозначение CLP(O), мы лишь подчеркиваем роль методов логического программирования как конструктивной основы для выстраивания перспективных вычислительных стратегий разрешения систем неоднородных ограничений, формально специфицированных на декларативных языках объектно-ориентированного моделирования данных.

В разделе 2 приводится несколько примеров постановки и решения классической математической задачи о ферзях с использованием парадигм логического, функционального и объектно-ориентированного программирования. В разделе 3 предлагаемый декларативный подход сравнивается с известными технологиями программирования в ограничениях и близкими им технологиями генерации тестовых данных с учетом контекстных ограничений. Общая вычислительная стратегия решения задач в ограничениях строится и обсуждается в разделе 4. В заключении указываются основные направления исследований для детальной алгоритмической проработки предлагаемого подхода и его практической реализации.

2. Некоторые примеры

Напомним, что задачи в ограничениях обычно описываются путем определения множества неизвестных переменных и зависимостей между ними. При этом процесс решения заключается в локализации областей значений переменных или в поиске значений, удовлетворяющих заданным зависимостям. Перечислим основные особенности их постановки.

Спецификации ограничений в рассматриваемых задачах носят декларативный характер, исключающий явное описание способов решения задачи (в качестве исключения здесь следует указать императивные языки ограничений, на которых пользователь описывает и некоторые методики решения). Спецификации обладают свойством нейтральности по отношению к неизвестным переменным и возможным альтернативным способам выражения и пересчета их друг через друга. Спецификации обладают также свойством аддитивности, предполагающим, что порядок задания ограничений не существенен для постановки исходной задачи, логические условия которой всегда могут быть представлены в эквивалентной конъюктивной форме. Исключение составляют так называемые иерархические системы, в которых разрешение ограничений осуществляется поэтапно в соответствии с предварительно назначенными приоритетами. Наконец, спецификации ограничений носят неопределенный характер с точки зрения их алгоритмической разрешимости. Поскольку система ограничений может быть недоопределена или переопределена, постановка задачи в ограничениях априори не предполагает ни существование, ни единственность решения, и его поиск должен вестись с учетом всех этих обстоятельств. Обсудим перечисленные свойства в контексте постановки и решения задач в классе CLP(O).

С этой целью рассмотрим классическую математическую задачу о ферзях и приведем возможные способы ее описания и решения на языках логического, функционального и объектно-ориентированного программирования. Задача формулируется следующим образом: необходимо расставить на шахматной доске восемь ферзей так, чтобы ни один из них не оказался под боем остальных фигур. Один из вариантов решения приведен на Рис. 1.

Положение каждой фигуры на шахматной доске характеризуется парой координат *x*/*y*, каждая из которых принимает целые значения от 1 до 8 (здесь мы отступим от традиционной буквенно-цифровой нотации ходов в шахматной партии, подобной “e2-e4”, и используем оператор «/» для объединения координат в один элемент списка). Тогда шахматная позиция представляется списком вида [*x*l/*y*l, *x*2/*y*2, *x*3/*y*3, *x*4/*y*4, *x*5/*y*5, *x*6/*y*6, *x*7/*y*7, *x*8/*y*8]. В этом представлении решение на Рис. 1 выглядит как [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1].

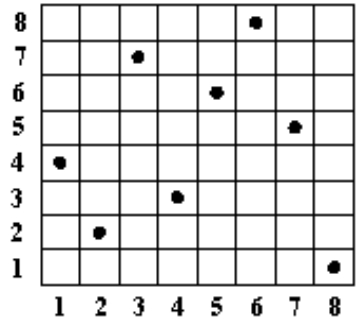


Рис. 1. Один из вариантов решения задачи о восьми ферзях

Принимая во внимание необходимость расположения ферзей на разных вертикалях (или на разных горизонталях), представление списка может быть сразу уточнено как [l/*y*l, 2/*y*2, 3/*y*3, 4/*y*4, 5/*y*5, 6/*y*6, 7/*y*7, 8/*y*8]. Таким образом, задача сводится к определению горизонтальных (или вертикальных) координат, исключающих расположение нескольких фигур на одних и тех же линиях шахматной доски.

Обсудим, каким образом задача может быть описана и решена на языке логического программирования Prolog. Уточненный список может использоваться в качестве шаблона решения, к которому последовательно применяются правила вывода в рамках общей схемы рекурсивного программирования. Будем считать список координат согласованным, если он определяет позицию, в которой ни один из ферзей не находится под боем. Возможны два случая:

Случай 1. Список пуст. Очевидно, пустой список является согласованным при отсутствии фигур и возможных атак.

Случай 2. Список не пуст. Тогда он представим в виде [*x*/*y* | *Остальные*], где *x*/*y* задает положение первого ферзя, а список "*Остальные*" — положение остальных. Определим необходимые условия, при которых непустой список является согласованным. Во-первых, список "*Остальные*" должен быть согласованным. Во-вторых, значения *x* и *y* должны принадлежать множеству целых чисел от 1 до 8 включительно. В-третьих, значения *x* и *y*, а также их разности (*x*-*y*) и суммы (*x*+*y*) не должны совпадать с соответствующими значениями из списка "*Остальные*".

Данные условия могут быть описаны на языке Prolog [17] в виде следующей программы (см. Рис. 2).

```
?-  шаблон( S), решение( S).
решение( [ ] ).
решение( [x/y | Остальные ] ) :-
       % Первый ферзь на поле x/y,
       % остальные ферзи на полях из списка
         Остальные
     решение( Остальные ),
     принадлежит( y, [1, 2, 3, 4, 5, 6, 7, 8] ),
     не_бьет( x/y | Остальные ).  % Первый ферзь
     не бьет остальных
не_бьет( x/y, [ ]).              % Некого бить
не_бьет( x/y, [x1/y1 | Остальные] ) :-
            y =\= y1,            % Разные
            y-координаты
            y1 - x1 =\= y - x,   % Разные
            диагонали
            y1 + x1 =\= y + x,
            не_бьет( x/y, Остальные).
принадлежит( x, [x | L] ).
принадлежит( x, [y | L] ) :-
            принадлежит( x, L).
% Шаблон решения
   шаблон([1/y1,2/y2,3/y3,4/y4,5/y5,6/y6,7/y7,8/y8]).
```

Рис. 2. Программа решения задачи о ферзях на языке Prolog

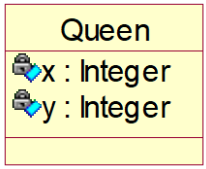
Обсудим возможность описания этой же задачи на языке ConstraintLisp [18], представляющим собой расширение стандарта ANSI Common Lisp [19] для программирования в ограничениях. Нововведением в ConstraintLisp является набор функций, который позволяет описывать арифметические ограничения в рамках функциональной и объектно-ориентированной парадигм, поддерживаемых языком ANSI Common Lisp. В программе на рисунке 3 определяется класс Queen с двумя атрибутами x и y для задания положения фигуры на шахматной доске. Для представления шахматной позиции используется массив Chessboard, в котором хранится восемь экземпляров класса Queen. Массив конструируется и инициализируется в результате вызова соответствующей функции make-queens. Отметим, что содержательные ограничения задачи определяются в виде вспомогательной функции constraints с формальными параметрами, соответствующими координатам анализируемых фигур. При этом фактическое назначение ограничений объектам массива Chessboard осуществляется в теле вложенного цикла, где описанный вид ограничений применяется ко всем парам итерируемых объектов. Решение генерируется и выводится на печать при вызове специальных функций generateObjs и printArrayObjs.

```
;;; определение класса Queen
(defclass Queen ( )
    (( x :initarg :x :accessor x ) ;;; координата по
     горизонтали
    ( y :initarg :y :accessor y))) ;;; координата по
     вертикали
;;; основная процедура поиска решения
(define Queens ( )
(let ((Chessboard ( make-queens)) ) ;;; создаем массив
из восьми ферзей

    (cldotimes ( I 7 ) ;;; начальное значение переменной
    I 0

        (cldotimes ( J (- 7 I )) ;;; начальное значение
        переменной J 0

            (constraints ( x (aref Chessboard I ))

                ( x (aref Chessboard (+ J I 1 ))) (1+ I)
                (+ J I 2 ))))

      ( generateObjs Chessboard ) ;;; генерация решения
      ( printArrayObjs Chessboard ))) ;;;
      вывод результатов
;;; ограничения отсутствия вертикальных и диагональных а
так
(define constraints (x1 x2 y1 y2)
    (constr (/= x1 x2)) ;;; по вертикали
    (constr (/= (+ x1 y1 ) (+ x2 y2 )))  ;;; по главной
    диагонали
    (constr (/= (- x1 y1 ) (- x2 y2 )))) ;;; по
     второй диагонали
;;; создаем массив ферзей
(defun make-queens ( )
    (let ( (queen-array (make-array 8 )))
        (dotimes (I 8 queen-array )
            (setf (aref queen-array I )
                (make-instance 'queen :x (make-cvar-in
                '((1 ,8 )))
                                      :y (1+ I) )))))
```

Рис. 3. Программа решения задачи о ферзях на языке ConstraintLisp

Наконец, опишем задачу о ферзях, используя декларативный язык объектно-ориентированного моделирования UML/OCL. Для этого определим

необходимые объектные типы данных и инвариантные условия на них. На диаграмме классов языка UML (см. рисунок 4) представлен объектный тип *Queen* с соответствующими целочисленными атрибутами *x* и *y*, задающими положение фигуры на шахматной доске. В виде инвариантов контекста *Queen* на языке OCL описаны исходные условия задачи о ферзях. Инвариант *A* задает необходимое число фигур на шахматной доске. Инвариант *B* определяет допустимую область значений для координат фигур, ограниченных полем 8x8 клеток. Условие расположения ферзей на разных вертикалях и горизонталях выражается инвариантом *C*, а условие расположения ферзей на разных диагоналях — инвариантом *D*.




```
context Queen

-- (A)

inv: self.AllInstances()->size() = 8

-- (B)

inv: self.x >= 1 and self.x <= 8 and self.y >= 1 and self.y <= 8

-- (C)
inv: self.AllInstances()->forAll(q1, q2 | q1 <> q2 implies
(q1.x <> q2.x and q1.y <> q2.y))
-- (D)

inv: self.allInstances()->forAll(q1, q2 | q1 <> q2 implies

((q1.x - q1.y) <> (q2.x - q2.y) and (q1.x + q1.y) <> (q2.x +
q2.y)))
```

Рис. 4. Описание задачи о ферзях в виде диаграммы классов и спецификации ограничений на языке UML/OCL

Несмотря на декларативность используемых в примерах языков, спецификации на языке UML/OCL обладают большей выразительностью для моделирования данных и постановки соответствующих задач в ограничениях. Для описания задачи о ферзях потребовалось лишь структурировать данные и определить ограничения, которым они должны удовлетворять. Подобный способ исключает задание каких-либо вспомогательных методик или схем решения, присущих языкам логического и функционального программирования. В самом деле, вычислительная модель Prolog

подразумевает редукцию исходной постановки к рекурсивной или итерационной схеме поиска решений на основе определяемых пользователем правил логического вывода и целей. Заметим, что порядок определения правил и целей может существенно влиять на ход решения. В рассмотренном примере на ConstraintLisp помимо определения объектной модели данных потребовалось явно сконструировать неизвестные переменные и для них задать схему применения ограничений. Отметим громоздкость подобного функционального способа постановки задач в ограничениях по сравнению с декларациями на UML/OCL.

3. Сравнительный анализ подходов к CLP

3.1. Краткий обзор технологий программирования в ограничениях

Программирование в ограничениях как самостоятельное научное направление сложилось в конце 60-ых – начале 70-ых годов. Примечательно, что первыми приложениями были задачи обработки изображений и параметрического моделирования пространственно-двумерных сцен. С тех пор направление существенно эволюционировало, охватывая новые классы задач и выдвигая новые подходы к их решению. Тем не менее, четыре фундаментальных принципа, а именно: декларативное моделирование ограничений, локальное распространение результатов, эффективный глобальный поиск решений и специализация языковых и математических средств, по-видимому, не претерпели серьезных изменений. Начиная со Sketchpad [20], ALICE [21], ThingLab [22], эти принципы успешно эксплуатируются и в современных реализациях.

Принцип декларативности [23] нашел конструктивное воплощение во многих языках функционального и логического программирования, включая Lisp и Prolog, допускающих описание и решение некоторых задач в ограничениях. Со временем понимание, что функциональная рекурсия и логический вывод являются лишь отдельными элементами более общей стратегии разрешения ограничений для сложно структурированных неизвестных переменных, привело к появлению CostraintLisp, Prolog III [24] и CHIP [25]. Язык реляционных баз данных SQL [26], предусматривающий развитые декларативные конструкции для описания запросов и ограничений целостности, также несет в себе черты языка программирования в ограничениях. Связь с технологиями управления базами данных становится еще более очевидной при анализе относительно новой концепции Constraint Database (CDB) [27], которая, расширяя реляционную, дедуктивную и объектно-ориентированную модели, устанавливает непосредственную связь между типами данных и ограничениями, которые их предопределяют.

Принцип локального распространения (в англоязычной литературе соответствующий терминам constraint propagation или consistency inference)

[28] имеет долгую историю и многочисленные приложения, в частности, в области интеллектуализации графических интерфейсов пользователя. Спекулятивные попытки предвосхитить результаты поиска путем локальных и циклических уточнений частных решений оказались довольно плодотворными при решении больших систем уравнений и неравенств. Многочисленные "радужные" воплощения принципа привели к созданию целой палитры методов: Red (красный), Orange (оранжевый), Yellow (желтый), Green (зеленый), Blue (синий), DeltaBlue (инкрементально-синий), SkyBlue (небесно-синий), UltraViolet (ультрафиолет), Purple (фиолетовый), DeepPurple (пурпурный), Indigo (темно-синий) [29–31]. Некоторые из них обеспечивают как полный, так и инкрементальный поиск решений применительно к простым и иерархическим системам ограничений.

Принцип эффективного глобального поиска решений довольно естественен для прикладных задач в ограничениях, большая часть из которых является NP-полными. В отличие от методов распространения, направленных на пропозициональный вывод и локальное улучшение частных решений, методы глобального поиска обеспечивают систематический или стохастический поиск общих решений, удовлетворяющих всем заданным ограничениям. Заметим, что поиск с возвратом (backtracking) в сочетании с локальным распространением нашел применение практически во всех реализациях систем программирования в ограничениях, включая упомянутые выше диалекты языков Lisp и Prolog. В случаях редукции исходной задачи в ограничениях к типовой математической постановке появляется возможность применения известных и апробированных методов решения. Например, симплекс-метод используется в качестве стандартного математического средства в системе 2LP [32], а метод комбинаторного поиска на конечных доменах — в успешном коммерческом продукте ILOG Solver [33].

В некоторых случаях для усиления выразительности языковых средств, а также для упрощения идентификации математических постановок выделяются специальные типы ограничений и для них предусматриваются особые математические методики. В частности, в CHIP и ILOG Schedule поддерживаются некоторые типовые ограничения, возникающие в задачах планирования и составления расписаний. Специализация языковых и математических средств является общей чертой для всех известных реализаций систем программирования в ограничениях, каждая из которых охватывает лишь некоторые типы ограничений и порождаемые ими классы задач. Например, Prolog III обеспечивает решение систем линейных ограничений, условий на логических переменных и списках, CHIP — линейных ограничений, условий на логических переменных и конечных доменах, CHARME [34] — условий на конечных доменах, 2LP — только линейных ограничений, ILOG Solver — линейных ограничений и условий на конечных доменах и интервалах, HELIOS [35] — условий на интервалах, NEWTON [36] — нелинейных уравнений. Схожая ситуация сложилась и в области систем конкурентного программирования в ограничениях CCP, к

ярким представителям которых следует отнести AKL [37], Oz [38], CIAO [39], TAO [40]. Для краткости мы не рассматриваем CCP технологии в настоящей работе, адресуя заинтересованного читателя к обзорам [16, 41].

3.2. О задачах генерации тестовых данных

Обсуждаемый класс задач программирования в ограничениях довольно широк и охватывает самые разнообразные приложения. Безусловно, к ним могут быть отнесены и задачи генерации данных, возникающие при тестировании разного рода программного обеспечения, начиная с протоколов обмена данными и заканчивая трансляторами языков программирования. Несмотря на отмеченную общность, технологии генерации тестовых данных развиваются как самостоятельное направление программной инженерии с конца 70-ых – начала 80-ых годов. Как правило, в постановках задач генерации первостепенное внимание уделяется синтаксически адекватному представлению данных, порождаемому заданной формальной грамматикой некоторого целевого языка. Позитивные и негативные тесты строятся как наборы синтаксически корректных и некорректных программ, полнота которых обеспечивается покрытием продукционных правил грамматики. Сформированные таким образом наборы данных могут использоваться, например, для тестирования синтаксических анализаторов.

Понимание важности учета семантики языка при генерации тестов привело к целому ряду работ, основные идеи которых тесно перекликаются с рассмотренными выше принципами программирования в ограничениях. В частности, декларативное описание семантических правил, последовательная локализация решений, применение мутаций при формировании семантически согласованных и рассогласованных наборов данных, стохастический поиск решений могут быть отнесены к рассмотренным выше принципам и методам программирования в ограничениях. Эти идеи нашли применение в рамках концепции тестирования в ограничениях CBT (Constraint-Based Testing) [42].

Отметим роль аппарата атрибутных грамматик Кнута [43] при генерации тестов. Данный аппарат удобен как для описания синтаксиса целевого языка, так и для задания семантических правил. На его основе успешно решаются задачи комплексного тестирования программного обеспечения с учетом синтаксиса и статической семантики целевого языка. В классических работах программы, сгенерированные на основе заданной контекстно-свободной грамматики, подвергаются дополнительным испытаниям для отбора семантически корректных тестов. В ряде работ предпринимаются попытки внедрить элементы императивного подхода, в частности, явно задавать последовательность и процедуры пересчета атрибутов в дереве синтаксиса для удовлетворения семантических правил языка. Эти элементы применяются, в частности, в технологии тестирования программного обеспечения на основе формальных спецификаций и моделей, развиваемой в ИСП РАН [44–47].

3.3. Основные достоинства предлагаемого подхода

Главной особенностью предлагаемого подхода к объектно-ориентированному программированию в ограничениях является использование универсальных языков моделирования данных EXPRESS, ODL/OQL, UML/OCL, OWL. Язык UML/OCL рассматривается здесь лишь в контексте моделирования данных, хотя его функциональное назначение существенно шире. Язык описания онтологий OWL отнесен к группе объектно-ориентированных языков в силу известного соответствия между их конструкциями и возможностью непротиворечивой трансформации моделей, описанных на них, друг в друга.

Обсуждаемый системный подход к ООСР обладает рядом достоинств по сравнению с упомянутыми выше технологиями, а именно:

— *декларативность* описания задачи в ограничениях, исключающая необходимость предварительной проработки редукции исходной постановки к схемам поиска решения на основе функциональной рекурсии и логического вывода. Отметим, что данный этап неизбежно присутствует при использовании языков функционального и логического программирования и, как правило, вызывает наибольшие трудности;

— *выразительность* описания задачи в ограничениях, обусловленная непосредственной интерпретацией исходной постановки в терминах информационного моделирования. Вместо определения отдельных переменных и зависимостей между ними пользователем описывается единая согласованная модель данных и ограничений, которая служит входной информацией для системы программирования в ограничениях. Результатом работы системы являются наборы данных, структурированные в соответствии с заданной моделью и удовлетворяющие ее семантическим правилам;

— *универсальность* описания задачи в ограничениях, определяемая возможностью его использования при решении разнообразных задач программной инженерии. Спецификации, подготовленные на актуальных языках моделирования, могут применяться для автоматизированной разработки программ с использованием CASE инструментов. К числу подобных приложений могут быть отнесены задачи генерации схем баз данных, программных интерфейсов доступа к данным, протоколов обмена данными, графических пользовательских интерфейсов на соответствующих реализационных языках. На основе подобных спецификаций может решаться и задача генерации тестовых данных в рамках концепции СВТ.

Естественно, общность предлагаемого подхода при описании задач в ограничениях на декларативных языках объектно-ориентированного

моделирования порождает проблемы алгоритмического характера, связанные с необходимостью разрешения больших систем разнородных алгебраических ограничений. Выработка единой вычислительной стратегии для удовлетворения подобных ограничений и решения задач в обсуждаемом классе CLP(O) представляется наиболее критичной.

4. Вычислительная стратегия программирования в ограничениях

В основе предлагаемой вычислительной стратегии разрешения ограничений лежит несколько конструктивных принципов и идей:

— использование единого метамодельного представления для работы с данными и моделями, специфицированными на языках EXPRESS, ODL/OQL, UML/OCL, OWL. Для взаимного преобразования моделей и определяемых ими данных могут быть применены известные трансформации;

— статический анализ спецификаций модели данных и идентификация математических постановок, к которым может быть сведена исходная задача в ограничениях. Для простых ограничений формируются альтернативные правила пересчета неизвестных переменных друг через друга, которые могут использоваться в методах распространения. Для сложных систем ограничений (и их подсистем) осуществляется анализ зависимостей по данным и определяется тип математической задачи, а также возможный метод решения. Например, идентификация множества ограничений как системы линейных алгебраических уравнений позволяет использовать известные методы матричной факторизации;

— последовательная редукция исходной задачи в ограничениях к задачам формирования множества неизвестных переменных (расчет необходимого числа объектов каждого типа), установления взаимосвязей (отношений ассоциации, агрегации и композиции между объектами) и определения числовых параметров (значений атрибутов объектов);

— применение методов линейного и квадратичного целочисленного программирования [48–50] для определения кардинальности множеств типизированных объектов. Выбор вида функционала позволяет эмулировать содержательные ситуации, когда, например, требуется сформировать минимальные, максимальные или репрезентативные наборы данных;

— применение методов логического вывода [51, 52] (прежде всего, метода резольвент и метода полисиллогистического вывода) для задания взаимосвязей объектов и определения неизвестных логического типа;

— применение методов интервальной арифметики, локализации и распространения для нахождения неизвестных вещественного типа при наличии простых зависимостей по данным;

— применение известных методов символьных преобразований [53, 54] и вычислительной математики для совместного решения систем (линейных, полиномиальных, нелинейных) алгебраических уравнений и неравенств;

— применение метода семантической реконсиляции [55–58] для инкрементального поиска общих решений с учетом всей системы наложенных разнородных ограничений.

Таким образом, в качестве основной стратегии предлагается использовать вышеописанную редукционную схему, состоящую в последовательном конструировании необходимого числа типизированных объектов, задании для них взаимосвязей и определении значений их атрибутов. Заметим, что большая часть семантических правил в объектно-ориентированных моделях специфицируется независимым друг от друга образом и не приводит к сложным зависимостям по данным. Это дает основания полагать о конструктивности применения подобной стратегии.

В более сложных ситуациях, когда основная стратегия не обеспечивает нахождение общего решения, предлагается использовать оригинальный метод семантической реконсиляции для коррекции уже найденных частных решений. Ключевые элементы этого метода были успешно апробированы в приложениях коллективной инженерии, к которым предъявляются близкие требования целостности и корректности результирующего представления данных. Применительно к обсуждаемому классу задач CLP(O) метод семантической реконсиляции предполагает следующие этапы:

— валидация семантических правил модели для исходных объектно-ориентированных данных и идентификация ограничений, не разрешенных в результате применения редукционного подхода;

— определение альтернативных способов коррекции для каждого нарушенного правила (определение наборов элементарных операций над множествами объектов, исправляющих нарушенные правила);

— совместный анализ выявленных способов коррекции с учетом уже выполненных правил и установление логических отношений между элементарными операциями;

— формирование и решение разреженной системы логических уравнений и определение допустимых комбинаций элементарных операций, удовлетворяющих всем ограничениям модели;

— коррекция исходных данных путем применения вычисленных наборов операций.

Ожидается, что применение редукционного подхода в сочетании с методом семантической реконсиляции обеспечит надежное решение задач в классе CLP(O).

Проиллюстрируем применение описанной вычислительной стратегии на примере задачи о правильном многограннике. Задача формулируется следующим образом: в трехмерном пространстве требуется построить правильный многогранник с заданным числом граней. Отметим, что в подобной постановке решение может и не существовать (известным фактом, например, является невозможность построения правильного пятигранника), поэтому желателен анализ существования решения.

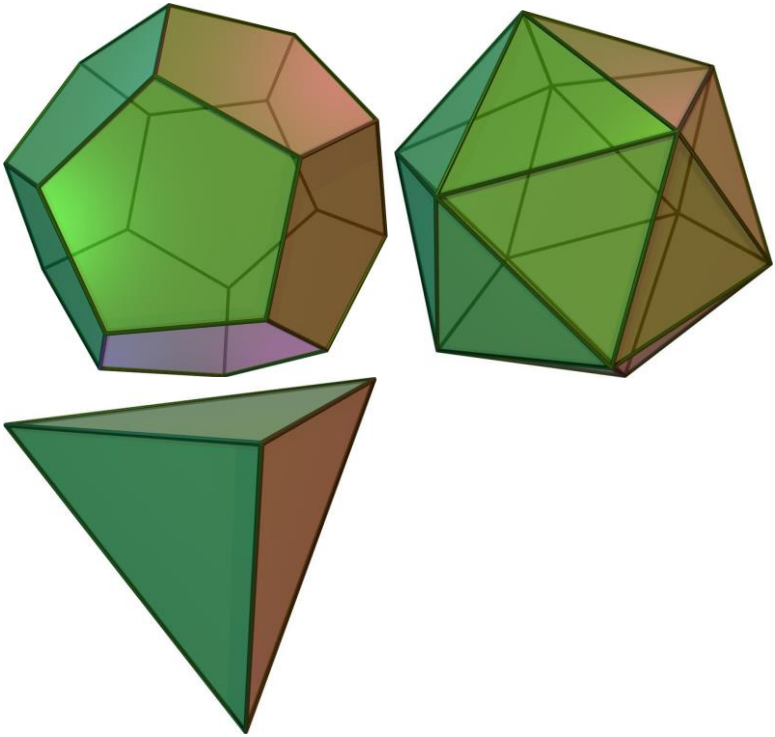


Рис. 5. Додекаэдр, икосаэдр, тетраэдр

Опишем задачу в ограничениях в виде спецификаций модели данных. Определим абстрактный класс *polyhedron* для представления произвольного правильного многогранника. Многогранник состоит из граней, ребер и вершин, моделируемых классами *face*, *edge* и *vertex* соответственно. Конкретизации класса *polyhedron* (см. Рис. 5), соответствующие геометрическим моделям *тетраэдра* (имеет 4 треугольных грани, 4 вершины, 6 рёбер, в каждой сходятся 3 ребра), *октаэдра* (имеет 8 треугольных граней, 8 ребер, 6 вершин, в каждой сходятся 4 ребра), *гексаэдра* (имеет 6 четырехугольных граней, 12 ребер, 8 вершин, в каждой сходятся 4 ребра), *додекаэдра* (имеет 12 пятиугольных граней, 30 рёбер и 20 вершин, в каждой сходятся 3 ребра) и *икосаэдра* (имеет 20 треугольных граней, 30 рёбер, 12 вершин, в каждой сходятся 5 ребер), моделируются классами *tetrahedron*, *octahedron*, *hexahedron*, *dodecahedron* и *icosahedron* соответственно. В рассматриваемой постановке абсолютный масштаб многогранника не существенен, поэтому для определенности положим длину ребер равной единице.

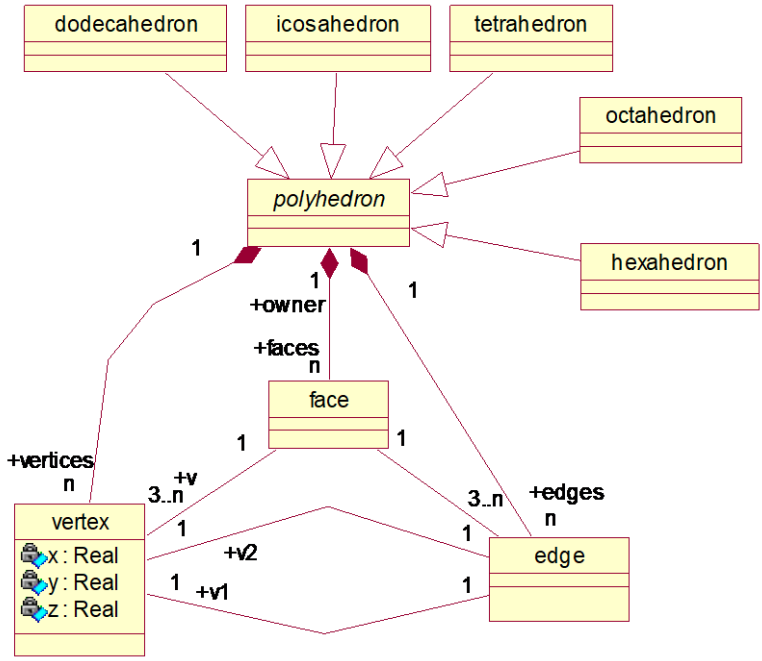


Рис. 6. UML модель для представления многогранников

На Рис. 6 приведена диаграмма классов UML, определяющая основные типы данных для представления многогранников. Условия исходной задачи

редуцированы в ограничения кратности для отношений ассоциаций и агрегаций между классами модели данных. Однако для исчерпывающей постановки исходной задачи требуется также задать:

— условие согласованности и связности топологических элементов многогранника, определяемое, в частности, формулой Эйлера для общего числа вершин, ребер и граней;

— условие выпуклости многогранника, например, в виде правил пространственной локализации его вершин в полупространствах, задаваемых плоскостями граней;

— условие правильности многогранника как равенства длин всех его ребер выбранному единичному значению.

Для описания этих ограничений воспользуемся языком OCL. Формула Эйлера для многогранника представляется в виде следующего инварианта, распространяемого на экземпляры класса *polyhedron*:

```
context polyhedron
inv: self.vertices->size() - self.edges->size() +
self.faces->size() = 2
```

Для задания ограничения выпуклости воспользуемся уравнением плоскости $Ax + By + Cz + D = 0$, проходящей через первые три вершины каждой грани и задающей полупространство для локализации остальных вершин многогранника. На языке OCL условие выпуклости многогранника может быть выражено соответствующим инвариантом класса face, в котором используются объявления вспомогательных локальных переменных:

```
context face
  let secVertices : Sequnce(Vertex) = collect(self.v) in
  let first : Vertex = secVertices(1) in
  let second : Vertex = secVertices(2) in
  let third : Vertex = secVertices(3) in

    let A : Real = first.y * (second.z - third.z ) +
                   second.y * (third.z - first.z ) +
                   third.y * (first.z - second.z ) in
    let B : Real = first.z * (second.x - third.x ) +
                   second.z * (third.x - first.x ) +
                   third.z * (first.x - second.x ) in

    let C : Real = first.x * (second.y - third.y ) +
                   second.x * (third.y - first.y ) +
```

```
                    third.x * (first.y - second.y ) in

    let D : Real = first.x * (third.y * second.z -
    second.y * third.z) +
                    second.x * (first.y * third.z -
                    third.y * first.z) +
                    third.x * (second.y * first.z -
                    first.y * second.z) in

self.owner.vertices->collect( v1 : vertex | not
self.vertices->includes( v1 ) )
    ->forAll( v2 : vertex | A * v2.x + B * v2.y + C *
    v2.z + D > 0 )
```

Наконец, условие равенства длин ребер прозрачным образом специфицируется средствами языка OCL как инвариант класса *edge*:

```
Context edge
inv: (self.v2.x – self.v1.x)*(self.v2.x – self.v1.x) +
     (self.v2.y – self.v1.y)*(self.v2.y – self.v1.y) +
     (self.v2.z – self.v1.z)*(self.v2.z – self.v1.z) = 1
```

В рамках описанной выше общей вычислительной стратегии программирования в ограничениях правильный многогранник может быть построен путем последовательной редукции исходной задачи к типовым математическим постановкам и их согласованному решению. Следуя данной стратегии, вначале проводится идентификации и разрешение ограничений, связанных с количеством топологических элементов многогранника. Как результат, определяется требуемое количество объектов соответствующих типов UML модели, а из их атрибутов формируется вектор неизвестных переменных. Затем на основе анализа спецификаций ограничений формируется система нелинейных алгебраических уравнений и неравенств относительно переменных, соответствующих координатам вершин многогранника. Решение системы может быть осуществлено численным образом, например, с использованием квази-ньютоновских методов, хорошо зарекомендовавших себя при решении широких классов нелинейных алгебраических задач. Рассмотренный пример иллюстрирует возможность применения единой стратегии для решения довольно сложных вычислительных задач, в постановке которых участвуют разнородные ограничения.

5. Заключение

Таким образом, предложен новый системный подход к реализации парадигмы объектно-ориентированного программирования в ограничениях на основе использования декларативных языков моделирования данных. Обсуждены и проиллюстрированы преимущества данного подхода по сравнению с известными технологиями функционального и логического программирования, а также отмечена его общность с методами генерации тестовых данных с учетом контекстных ограничений. Определена единая вычислительная стратегия для решения задач в выделенном классе CLP(O), сочетающая в себе ряд конструктивных идей и принципов. В дальнейшем предполагается детальная алгоритмическая проработка предложенного подхода и полнофункциональная реализация системы объектно-ориентированного программирования в ограничениях на основе декларативных языков моделирования данных.

Литература

[1] Dechter R. Constraint processing. Morgan Kaufmann, San Francisco, 2003.
[2] M. G. Buscemia, U. Montanarib. A survey of constraint-based programming paradigms. // Computer Science Review, Vol. 2, Issue 3, Dec. 2008, pp. 137–141.
[3] ISO 10303: 1994, Industrial automation systems and integration — Product data representation and exchange.
[4] OMG Model Driven Architecture: How systems will be built, http://www.omg.org/mda.
[5] W3C Semantic Web Activity, http://www.w3.org/2001/sw.
[6] ISO 10303-11: 2004, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual. Edition 2.
[7] Cattel R.G., Barry D.K., Berler M., et.al. The Object Data Management Standard: ODMG 3.0. Morgan Kaufmann, San Francisco, 2000.
[8] Unified Modeling Language, OMG Available Specification, Version 2.2, http://www.omg.org/spec/UML/2.2.
[9] Object Constraint Language Specification, Version 2.0, http://www.omg.org/technology/documents/formal/ocl.htm.
[10] OWL Web Ontology Language Guide, http://www.w3.org/TR/2004/REC-owl-guide-20040210.
[11] List of STEP Application Protocols, http://www.steptools.com/library/standard/step_2.html.
[12] Energetics, The Energy Standards Resource Centre, http://www.energistics.org/posc/Default.asp.
[13] BuildingSMART International Alliance for Interoperability, http://www.buildingsmart.com.
[14] Catalog of UML Profile Specifications, http://www.omg.org/technology/documents/profile_catalog.htm.
[15] E. Tsang. Foundations of constraint satisfaction. Academic Press Limited, London & San-Diego, 1993.

[16] V. Saraswat, P. Lincoln. Higher-Order Linear Concurrent Constraint Programming. Technical Report, Xerox PARC, Department of Computer Science, Stanford University, 1992.
[17] ISO/IEC 13211-1:1995. Information Technology — Programming Languages — Prolog — Part 1: General Core.
[18] Bing Liu, Yuen-Wah Ku. ConstraintLisp: An Object-Oriented Constraint Programming Language. // ACM SIGPLAN Notices, Vol. 27, No. 11, 1992, pp. 17–26.
[19] ANSI INCITS 226-1994 (R2004). Information Technology — Programming Language — Common Lisp.
[20] I.E. Sutherland. Sketchpad: A man-machine graphical communication system. Technical Report No. 296, Lincoln Laboratory, Massachusetts Institute of Technology, 1963.
[21] Alice, http://www.ps.uni-saarland.de/alice/index.html.
[22] A. Borning. The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. // ACM Transactions on Programming Languages and Systems, Vol. 3, No. 4, 1981, pp. 353–387.
[23] T. Mancini. Declarative constraint modelling and specification-level reasoning. PhD Thesis, Università degli Studi di Roma “La Sapienza”, Roma, Italy, March 2005.
[24] A. Colmerauer. An introduction to Prolog III. // Communications of the ACM, Vol. 33, No. 7, 1990, pp. 69–90.
[25] CHIP V5, http://www.cosytec.com.
[26] A. Beaulieu. Learning SQL, 2nd Edition. O’Reilly Media Inc., 2009.
[27] G. Kuper, L. Libkin. Constraint Databases. Springer-Verlag, Berlin, 2000.
[28] G. Tack. Constraint Propagation — Models, Techniques, Implementation. Doctoral Thesis, Saarland University, Germany, 2009.
[29] Freeman-Benson B.N., Maloney J., Borning A. An Incremental Constraint Solver. // Communication of the ACM, Vol. 33, No. 1, 1990, pp. 54–63.
[30] M. Sannella. SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction. // Proceedings of the 1994 ACM Symposium on User Interface Software and Technology, 1994, pp. 137–146.
[31] A. Borning, K. Marriott, P. Stuckey, Y. Xiao. Solving Linear Arithmetic Constraints for User Interface Applications: Algorithm Details. Technical Report 97-06-01, Department of Computer Science and Engineering University of Washington, 1997.
[32] K. McAloon, C. Tretkoff. 2LP: Linear Programming and Logic Programming. // Proceedings of PPCP’93, Newport, Rhode Island, 1993, pp. 178–189.
[33] IBM ILOG CP — Features and Benefits, http://www-01.ibm.com/software/integration/optimization/cp1/about/?S_CMP=rnav.
[34] Charme Reference Manual. AI Development Centre, Bull, France, 1990.
[35] L. Michel, P. Van Hentenryck. Helios: A modeling language for global optimization and its implementation in Newton. // Theoretical Computer Science, Vol. 173, No. 1, 1997, pp. 3–48.
[36] P. Van Hentenryck, L. Michel, F. Benhamou. Newton — constraint programming over nonlinear constraints. // Science of Computer Programming, Vol. 30, No. 1–2, 1998, pp. 83–118.
[37] P. Brand. Enhancing the AKL compiler using global analysis. Technical Report, Deliverable D.WP2.1.3.M2 in the ESPRIT project ParForce, 6707, 1994.
[38] The Mozart Programming System, http://www.mozart-oz.org.
[39] M. Hermenegildo. Some Challenges for Constraint Programming // Constraints, Vol. 2, No. 1, 1997, pp. 63–69.

[40] I. Shvetsov, T. Nesterenko, S. Starovit. Technology of Active Objects. AAAI Technical Report WS-97-05, Russian Research Institute of Artificial Intelligence & Institute of Informatics Systems, 1997.
[41] V. A. Saraswat. Concurrent Constraint Programming. The MIT Press, 1993.
[42] A. Gotlieb, T. Denmat, B. Botella. Constraint-based test data generation in the presence of stack-directed pointers. // Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005, pp. 313–316.
[43] М.В. Архипова. Автоматическая генерация тестов для семантических анализаторов трансляторов. Автореферат диссертации на соискание ученой степени кандидата физико-математических наук. Москва, ИСП РАН, 2006.
[44] А. Петренко, Е. Бритвина, С. Грошев, А. Монахов, О. Петренко. Тестирование на основе моделей. // Открытые системы, № 9, 2003, http://www.osp.ru/os/2003/09/183388.
[45] А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Генерация тестовых данных сложной структуры с учетом контекстных ограничений. // Труды Института системного программирования: т. 9. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2006, с. 83–96.
[46] A.K. Petrenko. Specification Based Testing: Towards Practice. // LNCS-2244, 2001, p.p. 287–300.
[47] С.В. Зеленов, С.А. Зеленова. Автоматическая генерация позитивных и негативных тестов для тестирования фазы синтаксического анализа. // Труды Института системного программирования: т. 8, ч. 1. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2004, с. 41–58.
[48] Васильев Ф.П., Иваницкий А.Ю. Линейное программирование. — М.: Факториал Пресс, 2003.
[49] О.В. Хамисов. Численное решение специальных задач невыпуклого квадратичного программирования. // Дискретный анализ и исследование операций, сер. 1, т. 12, № 4, 2005, с. 81–91.
[50] У. И. Зангвилл. Нелинейное программирование. Единый Подход. — М.: Советское радио, 1973.
[51] И. Братко. Алгоритмы искусственного интеллекта на языке PROLOG. — М.: Вильямс, 2004.
[52] Л. Стерлинг, Э. Шапиро. Искусство программирования на языке Пролог. — М.: Мир, 1990.
[53] M. Chetty, K.P. Dabke. Symbolic computations: an overview and application to controllerdesign. // Proceedings of IEEE sponsored international conference on Information, Decision and Control, Adelaide, 8th-10th February, 1999, pp.451–456.
[54] Марчук Г.И. Методы вычислительной математики. — М.: Наука, 1977.
[55] Semenov V.A., Karaulov A.A. Semantic-Based Decomposition of Long-Lived Transactions in Advanced Collaborative Environments. // Proceedings of 6 European Conference on Product and Process Modeling, ECPPM 2006, Spain, Valencia, September 11–15, 2006, pp. 223–232.
[56] Семенов В.А., Ерошкин С.Г., Караулов А.А., Энкович И.В. Семантическая реконсиляция прикладных данных на основе моделей. // Труды Института системного программирования: т. 13, ч. 2. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2007, с. 141–164.
[57] Semenov V.A. Collaborative Software Engineering Using Metamodel-Driven Approach. // Proceedings 16th IEEE International Workshops on Enabling Technologies:

Infrastructure for Collaborative Enterprises, WET ICE 2007, IEEE Computer Society Conference Publishing Services, 2007, pp. 178–179.
[58] Semenov V.A. Semantics-Based Reconciliation of Divergent Replicas in Advanced Concurrent Engineering Environments. // Complex Systems Concurrent Engineering: Collaboration, Technology Innovation and Sustainability, Springer-Verlag, London, 2007, pp. 557–564.

Теоретические и экспериментальные оценки сложности методов локального распространения в задачах программирования в ограничениях

Семенов В.А., Сидяка О.В.

Аннотация. Обсуждаются вопросы универсальности и эффективности методов локального распространения значений и степеней свободы применительно к задачам программирования в ограничениях. На основе сравнительного анализа обозначаются границы применимости методов и даются теоретические оценки их сложности. Отмечается важность построения и использования комбинированных алгоритмов, обеспечивающих надежное решение широких классов задач за полиномиальное время. Для предложенного комбинированного алгоритма проводятся серии вычислительных экспериментов, моделирующих системы ограничений переменной размерности с разным характером зависимостей по данным. Обсуждаются полученные экспериментальные оценки сложности алгоритма и отмечаются его конкурентные преимущества над традиционными методами локального распространения.

1. Введение

В последние годы логическое программирование в ограничениях CLP (Constraint Logic Programming) [1] и лежащие в его основе подходы к разрешению систем ограничений CSP (Constraint Satisfaction Problem) [2] получили заметное развитие. Технологии и методы программирования в ограничениях успешно применяются:

— при организации “интеллектуальных” графических интерфейсов пользователя, предусматривающих специальные правила для эргономичного расположения оконных элементов на экране [3, 4];

— в активных базах данных для определения полных (удовлетворяющих все условия целостности) и корректных (исключающих зацикливание) политик каскадного обновления семантически связанных элементов данных [5];

— в системах CAD/CAM/CAE для решения разнообразных инженерных задач, требующих автоматического и согласованного

пересчета параметров модели в зависимости от решений пользователя, принимаемых в ходе рабочей сессии [6, 7].

Парадигма программирования в ограничениях, основанная на непосредственной интерпретации прикладной задачи в терминах переменных, множеств допустимых значений и систем ограничений, оказалась привлекательной и конструктивной для построения сложных вычислительных приложений, в процессе выполнения которых возникает необходимость динамической идентификации и решения разнообразных математических задач алгебраического типа. Данная парадигма предполагает, что программист описывает прикладную задачу путем декларирования переменных и отношений между ними и полностью освобождается от написания императивной части программного приложения, ответственной за решение задачи с использованием тех или иных алгоритмов. Поскольку описываемые формальным образом ограничения могут иметь вид уравнений, неравенств или логических выражений самого общего вида, анализ зависимостей между переменными, идентификация типа математической задачи, а также ее непосредственное решение требуют целого арсенала математических методов и программных средств.

Для разрешения ограничений обычно применяются альтернативные подходы.

Численный подход [8] использует предварительную редукцию исходной системы ограничений к системе нелинейных алгебраических уравнений или к задаче условной нелинейной оптимизации. В зависимости от характера зависимостей задачи могут решаться разнообразными прямыми и итерационными методами, в частности, методами квазиньютоновского типа, методами сопряженных градиентов, методами матричной факторизации, симплекс-методом. Невысокая эффективность подхода объясняется проблемами локальной и глобальной сходимости итерационных методов для нелинейных систем ограничений, а также высокой сложностью прямых методов при решении линейных задач. Вместе с тем, при наличии сложных зависимостей между переменными данный подход часто оказывается единственно возможным.

Символьный [9] подход предполагает те же редукционные схемы, однако математические задачи решаются методами, основанными на символьных преобразованиях. В случае нелинейных ограничений общего вида время символьного решения растет экспоненциально с ростом числа ограничений. При этом шансы на успешное разрешение всех ограничений обычно невысоки. Для систем специального вида символьные методы могут дать заметный выигрыш.

Кластеризация [10] подразумевает декомпозицию исходной системы ограничений на подсистемы таким образом, чтобы обеспечить разрешимость каждой из них относительно внутренних переменных. Тогда последовательным обходом подсистем можно попытаться удовлетворить все ограничения исходной системы. При наличии изолированных групп

переменных кластеризация существенно упрощает и ускоряет процесс решения. В ряде случаев ее целесообразно применять в сочетании с методами анализа и идентификации типов математических задач, связанных с выделенными подсистемами ограничений.

Методы *локального распространения* [11–13] составляют наиболее распространенный и перспективный подход к решению больших систем ограничений, для которых известны или могут быть выведены альтернативные правила явного разрешения одних переменных относительно других. Как правило, современные реализации методов предусматривают предварительный этап планирования решения, на котором определяется порядок и способ разрешения ограничений, и этап непосредственного решения, на котором план, представленный в виде последовательности правил, применяется для расчета неизвестных переменных. Единожды построенный план может многократно использоваться для решения подобных задач в инкрементальной постановке, допускающей частичное изменение системы ограничений и их параметров.

Различают методы локального распространения значений и степеней свободы.

В методе локального распространения значений часть переменных принимается в качестве параметров задачи, через которые могут быть последовательно выражены остальные неизвестные. Пусть, например, задана система ограничений $x_1 + x_2 = x_3$ и $x_1 + x_2 + x_3 = x_4$, и переменные x_1 и x_3 выбраны в качестве параметров задачи. Тогда остальные неизвестные могут быть найдены путем последовательного применения правил $x_2 = x_3 - x_1$ и $x_4 = x_1 + x_2 + x_3$.

В методе локального распространения степеней свободы [11] происходит последовательный поиск и исключение групп свободных переменных и ассоциированных с ними ограничений. Свободные переменные — переменные, входящие лишь в одно ограничение исходной системы и участвующие в качестве выходов одного из его решающих правил. Найденное правило добавляется в начало плана решения, а процесс продолжается до тех пор, пока не исчерпаны все ограничения исходной системы и остаются свободные переменные.

В разделе 2 мы рассмотрим формальную постановку задачи в ограничениях, разрешимую для методов локального распространения, и обоснуем ее оценки сложности. В разделе 3 приведем алгоритмические версии методов локального распространения значений и степеней свободы и построим комбинированный алгоритм, сочетающий элементы двух методов. Раздел 4 посвящен представлению и анализу результатов вычислительных экспериментов, убедительно доказывающих преимущества комбинированного алгоритма над традиционными методами локального распространения.

2. Задачи в ограничениях

Задачи в ограничениях обычно описываются путем определения множества неизвестных переменных и алгебраических зависимостей между ними. При этом процесс решения заключается в локализации областей значений переменных или в поиске значений, удовлетворяющих заданным зависимостям. Будем рассматривать только *ограничения потока данных*. Подобные ограничения могут разрешаться на основе предварительно выведенных или заданных правил. Каждое такое правило предоставляет метод разрешения ограничения с сигнатурой, определяющей, какие переменные задачи являются его входными параметрами, а какие — выходными. Каждое правило при этом имеет хотя бы один выходной параметр, подлежащий модификации в результате применения правила и разрешения соответствующего ограничения.

2.1. Формальная постановка

Итак, пусть задано множество переменных $X = \{x_i \mid i = 1..n\}$ и система ограничений на нем $C = \{C_j(X_j) \mid j = 1..m,\ X_j \subseteq X\}$. Будем считать, что для каждого ограничения системы $C_j \in C$ известен соответствующий набор правил $R_j = \{R_{jk}(X_{jk}^{IN}, X_{jk}^{OUT}) \mid k = 1..k_j, X_{jk}^{IN} \subset X_j, X_{jk}^{OUT} \subseteq X_j, X_{jk}^{IN} \cap X_{jk}^{OUT} = \varnothing\}$ разрешения относительно переменных, участвующих в них в качестве входных и выходных параметров.

Задача стоит в нахождении плана решения в виде последовательности правил $S : \{1..\mathrm{m}\} \to \bigcup_{j=1}^{m}\left(\bigcup_{k=1}^{k_j} R_{jk}\right)$, удовлетворяющей следующим условиям:

1. последовательность правил полна для разрешения всех ограничений исходной системы $\forall i \in \{1..m\} \exists j \in \{1..m\} : S(j) \in R_i$;
2. последовательность правил неконфликтна и исключает случаи повторной модификации установленных переменных и нарушения разрешенных ограничений системы. Формально данное условие выражается как

$$\forall i', i'' \in \{1..m\}, i' < i'' : \begin{cases} X_{j'k'}^{OUT} \cap X_{j''k''}^{OUT} = \varnothing \\ X_{j'k'}^{IN} \cap X_{j''k''}^{OUT} = \varnothing \end{cases} \quad \text{ãääå} \quad \mathrm{S}(i') = \mathrm{R}_{\mathrm{j'k'}}, \quad \mathrm{S}(i'') = \mathrm{R}_{\mathrm{j''k''}}$$

Задача предыдущего раздела формально представляется как система ограничений относительно множества неизвестных переменных $X = \{x_1, x_2, x_3, x_4\}$:

$$\begin{cases} C_1 : x_1 + x_2 = x_3 \\ C_2 : x_1 + x_2 + x_3 = x_4 \end{cases}$$

.Необходимые решающие правила могут быть заданы, например, следующим образом:

$$\begin{cases} R_{11}: x_1 = x_3 - x_2;\ R_{12}: x_2 = x_3 - x_1;\ R_{13}: x_3 = x_1 + x_2 \\ R_{21}: x_1 = x_4 - x_2 - x_3;\ R_{22}: x_2 = x_4 - x_1 - x_3;\ R_{23}: x_3 = x_4 - x_1 - x_2;\ R_{24}: x_4 = x_1 + x_2 + x_3 \end{cases}$$

Планом решения данной задачи является $S = \{R_{13}, R_{24}\}$. Можно заметить, что решение не единственно. Например, $S = \{R_{11}, R_{24}\}$ также является планом ее решения.

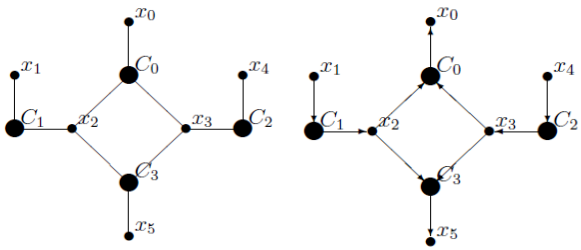
2.2. Визуальное представление задачи и плана решения

Естественным визуальным представлением задачи в ограничениях является двудольный ненаправленный граф $G(X,C,R)$ [2-5], где X – множество вершин, ассоциированных с переменными задачи, C – множество вершин, ассоциированных с ограничениями задачи (на рисунках ниже отображены большими кругами), и R – множество ненаправленных ребер графа, соединяющих вершины ограничений с соответствующими вершинами переменных. Переменная задачи, участвующая в некотором ограничении, отображается ребром, соединяющим соответствующие вершины двудольного графа.

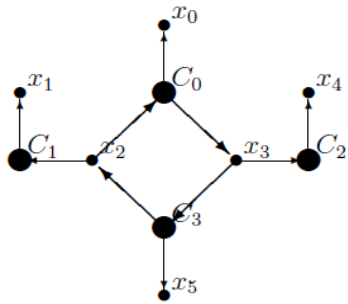
Результатом решения задачи в ограничениях является план в виде последовательности правил, каждое из которых приводит к удовлетворению одного из ограничений исходной системы. План решения визуально отображается на графе с помощью направленных ребер. Комбинация ориентаций ребер, инцидентных вершине некоторого ограничения, определяет для него сигнатуру решающего правила. Входные параметры правила отображаются на графе ребрами, входящими в вершину ограничения, а выходные — ребрами, выходящими из нее. Ребра переменных, участвующих в ограничении, но не задействованных в выбранном правиле его разрешения, исключаются из представления графа. Двудольный ориентированный граф плана решения исходной задачи в ограничениях, представимой в виде $G(X,C,R)$, обозначим как $\overline{G}(X,C,R)$.

Принцип разрешимости задач в ограничениях также имеет прозрачную визуальную интерпретацию. Задача разрешима, если план ее решения, полученный в виде $\overline{G}(X,C,R)$, не содержит циклов и каждая его вершина, ассоциированная с переменной задачи, имеет не более одного входящего ребра. Циклом в данном случае называется упорядоченный набор пар вершин $x_{n_1}, C_{n_2}, x_{n_3}, C_{n_4}, ..., x_{n_{k-1}}, C_{n_k} \in \overline{G}$ такой, что вершины-переменные $x_{n_1}, x_{n_3}, ..., x_{n_{k-1}} \in X$ и вершины-ограничения $C_{n_2}, C_{n_4}, ..., C_{n_k} \in C$ соединены последовательно направленными ребрами от вершины x_{n_i} к вершине $C_{n_{i+1}}$ для

всех $i = 1..k-1$, от вершины C_{n_i} к вершине $x_{n_{i+1}}$ для всех $i = 2..k-2$, и от вершины C_{n_k} к вершине x_{n_1}.



(а) (б)



(в)

Данные условия гарантируют, что существует конечная неконфликтная последовательность правил, применяя которую, можно разрешить все ограничения исходной системы. Назовем задачу потенциально разрешимой, если она разрешима, однако существует, по крайней мере, один циклический план решения $\overline{G}(X,C,R)$. Для задачи, неразрешимой в обсуждаемой постановке, не может быть построен ациклический план решения $\overline{G}(X,C,R)$.

2.3. Вычислительная сложность задачи в ограничениях

Обсудим вопрос вычислительной сложности задач в ограничениях потока данных [14, 15]. Как правило, подобные задачи недоопределены, что приводит к существованию большого числа решений, а наличие альтернативных правил для каждого ограничения исходной системы обуславливает и

многовариантность способов их нахождения. Покажем, что задача в обсуждаемой общей постановке является NP-полной в силу того, что частные постановки также не разрешимы за полиномиальное время. С этой целью рассмотрим классическую задачу раскраски графа в k цветов и переформулируем ее в терминах эквивалентной задачи в ограничениях. Каждой вершине графа поставим в соответствие некоторую переменную задачи, означающую ее цвет, а каждому ребру — ограничение в виде условия несовпадения цветов, связанных с инцидентными вершинами. Каждое такое ограничение можно удовлетворить одним из $k(k-1)$ правил, каждое из которых перекрашивает две инцидентные вершины в разные цвета. Сформулированная задача в ограничениях имеет решение тогда и только тогда, когда исходный граф имеет раскраску в k цветов, что доказывает их эквивалентность и NP-полноту обсуждаемой задачи.

Известно, что в общем случае задача в ограничениях не решается эффективно, однако при переходе к частным упрощенным постановкам возникают альтернативные возможности, обеспечивающие поиск решения за полиномиальное время. В данной работе мы строим комбинированный алгоритм, который, с одной стороны, гарантирует решение в самой общей постановке, а с другой стороны — осуществляет поиск решения за полиномиальное время в частных, но распространенных на практике случаях.

3. Основные алгоритмы локального распространения

Прежде всего, приведем базовые алгоритмы для методов локального распространения значений и степеней свободы, на которых будет основываться предложенный комбинированный алгоритм.

3.1. Алгоритм распространения степеней свободы

Алгоритм распространения степеней свободы [11] гарантированно находит решение за полиномиальное время, если оно существует, и информирует об его отсутствии в противном случае. Однако постановка задачи имеет одно принципиальное уточнение — переменные каждого ограничения исходной системы должны участвовать во всех правилах его разрешения в качестве входных или выходных параметров. Данное условие выражается формальным образом: для любого ограничения системы $\forall C_j(X_j) \in C$, $j = 1..m$ и для любого его правила $\forall R_{jk}(X_{jk}^{IN}, X_{jk}^{OUT}) \in R_j$, $k = 1..k_j$ имеет место $X_{jk}^{IN} \cup X_{jk}^{OUT} = X_j$.

Ниже приведена поэтапная схема алгоритма.

1. Инициализируем план решения $S = \varnothing$.

2. Формируем подмножество всех свободных переменных задачи $X' \subseteq X$. Напомним, что свободной называется переменная, входящая лишь в одно ограничение исходной системы и участвующая в качестве выхода одного из его решающих правил. Формальное определение свободной переменной выглядит следующим образом:

$$x' \in X' \Leftrightarrow \begin{cases} \exists\, j' \in (1..m) : x' \in X_{j'}; \forall\ j \in (1..m), j \neq j' : x' \notin X_j \\ \exists k \in (1..k_{j'}) : R_{j'k} \in R_{j'}, \{x'\} = X_{j'k}^{OUT}) \end{cases}$$

3. Выбираем произвольную свободную переменную $x' \in X'$ и устанавливаем ассоциированное с ней ограничение $C_{j'}$, $\{x'\} = X_{j'}$ (в силу условий задачи оно всегда единственно).

4. Выбираем решающее правило $R_{j'k'}$ для ограничения $C_{j'}$ с фактическим выходным параметром-переменной $\{x'\} = X_{j'k'}^{OUT}$. Добавляем правило в начало плана решения S.

5. Корректируем представление задачи, удаляя переменную x' и связанное с ней ограничение $C_{j'}$. Обновляем подмножество свободных переменных $X' \subseteq X$ с учетом проведенных изменений.

6. Если $X' \neq \varnothing$, то переходим к пункту 3 и выбираем следующую свободную переменную.

7. Если множество ограничений в текущем представлении задачи исчерпано ($C = \varnothing$), то **план решения найден**. Если множество ограничений не исчерпано ($C \neq \varnothing$), то **плана решения не существует**.

Итак, имеет место следующее утверждение о корректности приведенного алгоритма.

Теорема. Алгоритм распространения степеней свободы завершает работу либо когда план решения найден, либо когда план не существует.

Доказательство.

□ Доказательство корректности алгоритма сводится к доказательству двух вспомогательных утверждений:

a. Если алгоритм завершил работу при исчерпании всех ограничений, то построенный план решения корректен.

b. Если алгоритм завершил работу при оставшихся ограничениях, то задача не разрешима.

Для доказательства первого утверждения перенумеруем все ограничения и переменные исходной системы сквозным образом в порядке, обратном исключению свободных переменных и ассоциированных с ними ограничений так, чтобы ограничения непосредственно предшествовали исключаемым переменным. Иными словами, перенумеруем их в соответствии с порядком применения правил в построенном плане решения. Тогда переменные, не являющиеся выходными параметрами ни одного из правил плана решения, окажутся в самом начале списка. При подобном расположении все ребра в графе плана имеют направление от элементов (переменных и ограничений), располагающихся выше по списку, к элементам (ограничениям и переменным соответственно) ниже по списку, что исключает циклы в графическом представлении плана. В самом деле, данному направлению соответствуют все ребра графа, а именно:

— ребра, соединяющие ограничения и выходные переменные правил, примененных для их разрешения;

— ребра, инцидентные переменным, которые не использовались в качестве выходов ни одного из решающих правил плана и, поэтому, оказались в самом начале списка;

— ребра, инцидентные переменным, которые использовались в качестве выходов одного из решающих правил плана, но были удалены в ходе его построения и поэтому не могли служить входными параметрами для правил разрешения ограничений, находящихся выше по списку.

Доказательство второго утверждения основывается на следующих рассуждениях относительно наличия циклов в графическом представлении плана. Во-первых, построенный без циклов план должен иметь хотя бы одно правило с выходными переменными, не используемыми в качестве входных параметров других правил. Если такого правила не существует, то путь в графе плана, проведенный от некоторого ограничения к выходным переменным и затем от них (которые по предположению существуют) к входным параметрам следующих ограничений, неизбежно замкнется. Во-вторых, в построенном графе без циклов не может быть вершины-переменной с несколькими входящими ребрами от разных ограничений. Ограничение, имеющее в качестве выхода переменную без выходных ребер, в качестве выходов имеет только свободные переменные. Поэтому, если в результирующем графе нет ограничения со свободными выходными переменными, то план решения не может быть представлен графом без циклов. ■

3.2. Алгоритм распространения значений, основанный на оценке перспективности выбираемых правил

Обсудим метод локального распространения значений и построим его алгоритмическую версию, использующую дополнительную спекулятивную оценку перспективности выбираемых правил. Подобная оценка позволяет выстраивать план решения локальным образом, но с учетом перспективы разрешения ограничений, анализируемых на последующих этапах алгоритма.

В основе оценки лежат следующие общие соображения. Во-первых, любые два правила плана не могут иметь общие выходные параметры. В противном случае ограничение, удовлетворенное первым, может быть нарушено при попытке разрешения второго ограничения и связанной с ней повторной модификации выходной переменной. Во-вторых, правила, использующие некоторые переменные в качестве входных, должны выполняться строго после правил, для которых эти переменные являются выходными параметрами. На каждом шаге обсуждаемого алгоритма проводится оценка зависимостей и выбирается то правило, применение которого гарантирует разрешимость любого из оставшихся ограничений. Тем самым, процесс планирования решения, реализуемый алгоритмом, всегда оказывается перспективным и может быть продолжен, по крайней мере, на последующем шаге разрешения очередного ограничения.

Ниже приводится пошаговая схема алгоритма, предполагающая последовательное разрешение ограничений задачи методом локального распространения значений. В случае успеха алгоритмом обрабатывается очередное ограничение, в противном случае — осуществляется возврат к предыдущему шагу и предпринимается попытка внести изменения в план решения. Алгоритм программно реализуется, используя аппарат рекурсивных функций.

1. Формируем упорядоченные множества разрешенных и неразрешенных ограничений задачи $C' = \varnothing$ и $C'' = C$ соответственно. Устанавливаем номер итерации $i = 0$ (совпадающий с количеством разрешенных ограничений в системе $i = |C'|$).

2. Инициализируем переменную цикла $j = 0$ для перебора неразрешенных ограничений из множества C''.

3. Устанавливаем индекс текущего обрабатываемого ограничения $j = j + 1$.

4. Если $j > |C''|$, то множество исчерпано и на данной итерации не удалось удовлетворить ни одно ограничение. В этом случае, если множество разрешенных ограничений C' пусто, то это означает,

что перебраны все варианты, **план решения не существует** и следует завершить работу. Если $C' \neq \varnothing$, то последнее разрешенное ограничение перемещается из множества C' обратно в C'' и осуществляется возврат к предыдущей итерации с номером $i = i - 1$, к пункту 7.

5. Пытаемся разрешить ограничение C''_j. Для этого формируем множество перспективных правил $R^*_j \subseteq R_j$. Перспективными считаются те правила, при включении которых в план решения, каждое из оставшихся ограничений C'' может быть удовлетворено, по крайней мере, с помощью одного правила. То есть, $R_{jk} \in R^*_j \Leftrightarrow \forall C_i \in C'' \exists l : X^{OUT}_{il} \cap X^{OUT}_{jk} = \varnothing \;\&\; X^{OUT}_{il} \cap X^{IN}_{jk} = \varnothing$

6. Инициализируем переменную цикла $k = 0$ для просмотра всех перспективных правил из множества R^*_j.

7. Устанавливаем индекс текущего правила $k = k + 1$. Если $k > \left|R^*_j\right|$ и множество перспективных правил исчерпано ($R^*_j = \varnothing$) при попытке разрешить текущее ограничение, то переходим к пункту 3 с целью анализа другого ограничения.

8. Выбираем правило $R^*_{jk} \in R^*_j$ и добавляем его в план решения $R^*_{jk} \in S$. Перемещаем разрешенное ограничение C''_j из множества C'' в C'. Переходим к следующей итерации алгоритма $i = i + 1$.

9. Если все ограничения разрешены и $C'' = \varnothing$, то **план решения найден**. В противном случае переходим к пункту 2 для дальнейшего планирования решения.

Теорема. Алгоритм распространения значений заканчивает свою работу, либо когда план решения найден, либо когда он не существует.

Доказательство.

□ Алгоритм строит корректный план решения, выстраивая правила разрешения ограничений в порядке, исключающем конфликты.

Если план решения алгоритмом не обнаружен, то его действительно не существует, поскольку в худшем случае алгоритм перебирает все варианты его построения за исключением заведомо некорректных, приводящих к образованию циклов в графе плана решения. ■

Как было отмечено, задача разрешения ограничений в общей постановке является NP-полной. Приведенная алгоритмическая версия метода распространения значений также имеет экспоненциальную оценку сложности

в худшем случае, однако позволяет уменьшить число анализируемых вариантов планирования решения за счет исключения заведомо неперспективных.

3.3. Комбинированный алгоритм разрешения ограничений

Построим комбинированный алгоритм, который гарантировал бы решение задачи в самой общей постановке с использованием метода распространения значений, однако осуществлял бы это более эффективным образом при наличии в задаче особенностей, допускающих применение метода распространения степеней свободы. С этой целью в предлагаемом алгоритме будем привлекать метод распространения степеней свободы всякий раз, когда есть возможность исключить свободные переменные, а при ее отсутствии — использовать универсальный метод распространения значений со спекулятивной оценкой перспективности выбранных правил.

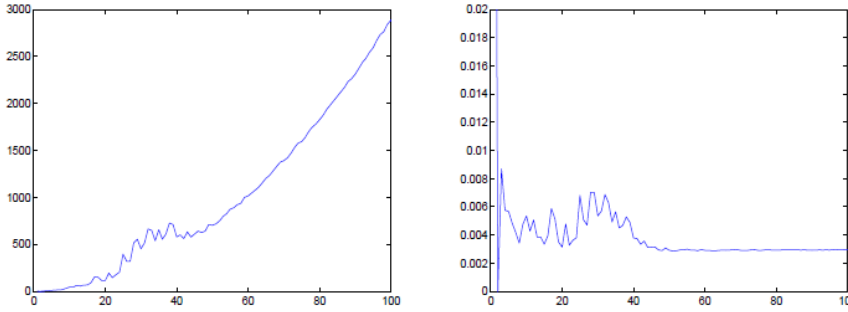
Комбинированный алгоритм работает следующим образом:

1. Определяются переменные, входящие только в одно ограничение исходной системы.
2. Проверяется, есть ли у этих ограничений правиламетоды, которые на выходе имеют только переменные, найденные в п. 1.
3. Если есть, то переменные считаются свободными, и ограничение разрешается по методу, найденному в п. 2.
4. Если свободных переменных нет, то включается поиск решения, основанный на определении неперспективности применения правил. В каждой итерации поиска, если появляется свободная переменная, то соответствующее ограничение разрешается по правилу, выход которого состоит из свободных переменных.

4. Вычислительные эксперименты

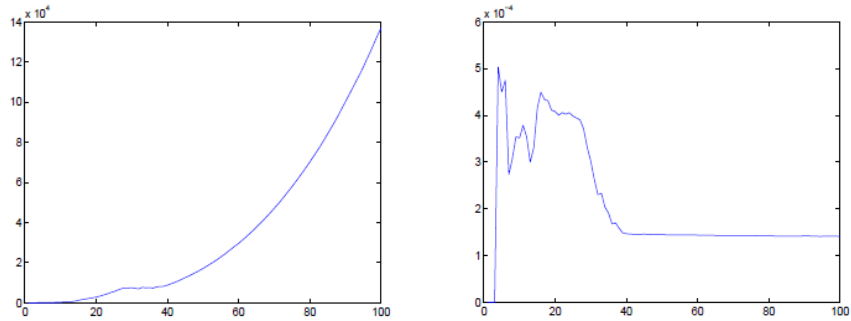
Комбинированный алгоритм, как и базовый метод распространения значений, в худшем случае перебирает все варианты решения за экспоненциальное время. Однако на практике алгоритм демонстрирует полиномиальные оценки сложности в случаях, допускающих эффективную редукцию исходной системы ограничений на основе метода распространения степеней свободы. Ниже приводятся результаты вычислительных экспериментов, демонстрирующие время работы алгоритма в зависимости от числа ограничений и переменных в исходной системе, а также его асимптотическую сложность на наборе тестовых задач.

В качестве первой тестовой задачи была выбрана система ограничений переменной размерности n следующего вида:



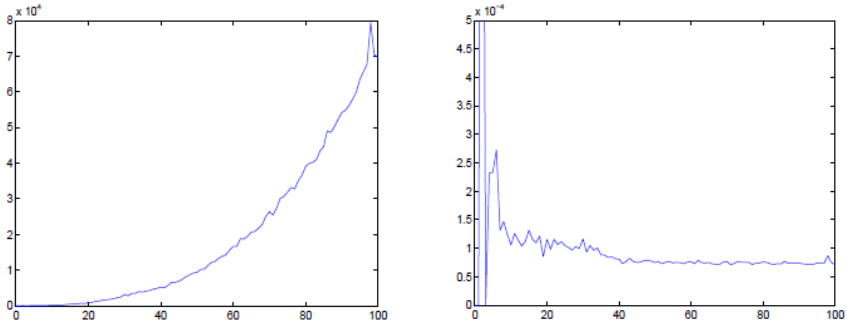
(а) (б)

Рис. 2. Время работы алгоритма в зависимости от количества ограничений (а) и его оценка сложности $O(x^2)$ *(б) при прямом порядке задания ограничений*



(а) (б)

Рис. 3. Время работы алгоритма в зависимости от количества ограничений (а) и его оценка сложности $O(x^3)$ *(б) при обратном порядке задания ограничений*



(а) (б)

Рис. 4. Время работы алгоритма в зависимости от количества ограничений (а) и его оценка сложности $O(x^3)$ (б) при произвольном порядке задания ограничений

$$\begin{cases} x_1 = f_1(x_0) \\ x_2 = f_2(x_0, x_1) \\ \dots \\ x_n = f_n(x_0, x_1, \dots, x_{n-1}) \end{cases} \quad (1)$$

Структура зависимостей между переменными данной задачи подобна портрету нижней треугольной матрицы Якоби для соответствующей системы алгебраических уравнений.

Поскольку время работы алгоритма зависит от порядка, в котором задаются ограничения системы и правила их разрешения, было проведено несколько серий экспериментов. Результаты экспериментов с прямым, обратным и произвольным порядком задания ограничений и соответствующих правил разрешения представлены на Рис. 2, 3 и 4 соответственно. Прямой порядок соответствует способу нумерации ограничений в приведенной выше системе сверху вниз (1), обратный — нумерации в противоположном направлении снизу вверх, а произвольный — нумерации на основе матрицы перестановок, сгенерированных случайным образом. Полученные результаты подтверждают полиномиальную сложность комбинированного алгоритма на рассмотренном классе задач, причем алгоритм демонстрирует квадратичную асимптотическую сложность при прямом порядке задания ограничений и характерную кубическую — при обратном и произвольном порядке.

Эксперименты проводились на персональном компьютере типовой конфигурации с процессором Core 2 Duo E8400 (3.00 GHz) и оперативной памятью 2GB (800 MHz). Время работы алгоритма представлено на графиках в миллисекундах CPU.

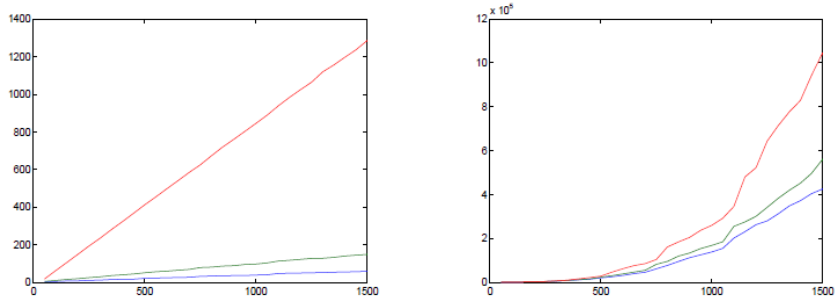
В качестве альтернативной тестовой задачи использовалась система ограничений переменной размерности n с фиксированным максимальным числом параметров k:

$$\begin{cases} x_1 = f_1(x_0) \\ x_2 = f_2(x_0, x_1) \\ \dots \\ x_k = f_k(x_0, x_1, \dots, x_{k-1}) \\ x_{k+1} = f_{k+1}(x_1, x_2, \dots, x_k) \\ \dots \\ x_n = f_n(x_{n-k}, x_{n-k+1}, \dots, x_{n-1}) \end{cases} \quad (2)$$

Выбранная задача эквивалентна системе алгебраических уравнений с диагональной матрицей Якоби шириной k.

Проведенные эксперименты показывают существенное влияние числа параметров в ограничениях на время работы алгоритмов, что объясняется анализом большего числа зависимостей по данным при выборе каждого очередного правила. Не менее важной представляется установленная линейная сложность комбинированного алгоритма для данной задачи в отличие от метода распространения значений. Существенный выигрыш комбинированного алгоритма объясняется эффективной редукцией исходной системы ограничений в результате исключения свободных переменных базовым методом распространения степеней свободы. Как видно из графиков, приведенных на Рис. 5, выигрыш на задачах типовой размерности может составлять несколько порядков и продолжает расти с увеличением размерности задачи.

Наконец, приведем тестовую задачу, которая служит иллюстрацией того, что использование комбинированного алгоритма не налагает каких-либо дополнительных условий на постановку задачи в ограничениях в отличие от алгоритма распространения степеней свободы. Тем самым, более высокая эффективность алгоритма не является следствием сужения границ его применимости или снижения надежности.



(а) (б)

Рис. 5: Время работы комбинированного алгоритма (а) и алгоритма распространения значений (б) в зависимости от количества ограничений при разных значениях параметра k=50, k=10, k=1 (верхние, средние и нижние кривые на графиках соответственно)

Рассмотрим систему ограничений

$$\begin{cases} C_1 : (f_{11}(x_0) - x_1)(f_{12}(x_1) - x_2) = 0 \\ C_2 : (f_{21}(x_1) - x_2)(f_{22}(x_2) - x_3) = 0 \\ \ldots \\ C_n : (f_{n1}(x_{n-1}) - x_n) = 0 \end{cases} \quad (3)$$

со следующим набором правил разрешения:

$$\begin{cases} R_{11} : x_1 = f_{11}(x_0);\ R_{12} : x_2 = f_{12}(x_1); \\ R_{21} : x_2 = f_{21}(x_1);\ R_{22} : x_3 = f_{22}(x_2); \\ \ldots \\ R_{n1} : x_n = f_{n1}(x_{n-1}); \end{cases} \quad (4)$$

В подобной постановке не выполняется условие вхождения всех переменных ограничения в каждое из его правил, которое необходимо для гарантированного поиска решения с помощью метода распространения степеней свободы. При работе данный алгоритм диагностирует невозможность найти план решения, хотя он существует. Комбинированный алгоритм справляется с задачей благодаря привлечению метода распространения значений. Примечательно, что решение находится за линейное время.

5. Заключение

Таким образом, рассмотрены вопросы применения методов локального распространения к задачам программирования в ограничениях. Предложенный комбинированный алгоритм сочетает в себе элементы метода распространения степеней свободы и метода распространения значений, обеспечивая надежное решение широких классов задач за полиномиальное время. Проведенные серии вычислительных экспериментов и полученные экспериментальные оценки сложности комбинированного алгоритма подтверждают его конкурентные преимущества над традиционными методами, что обуславливает его использование в перспективных системах программирования в ограничениях.

Литература

[1] Dechter R. Constraint processing. Morgan Kaufmann, San Francisco, 2003.
[2] E. Tsang. Foundations of constraint satisfaction. Academic Press Limited, London & San-Diego, 1993.
[3] K. Bharat. Supporting the Construction of Distributed, Interoperative, User Interface Applications. Ph.D. Dissertation, Georgia Institute of Technology, 1996.
[4] R.D. Hill. The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications. Proceedings of SIGCHI '92. 1992, pp. 335-342.
[5] Joao Pascoal Faria, Raul Moreira Vidal. 1999. Data-driven Active Rules for the Maintenance of Derived Data and Integrity Constraints in User Interfaces to Databases
[6] C. M. Hoffman, A. Lomonosov. Decomposition plans for geometric constraint systems. // J. Symbolic Computation. Volume 31. 2001. pp. 367-408.
[7] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, R. Paige. A Geometric Constraint Solver. 1995.
[8] A. Borning, K. Marriott, P. Stuckey, Y. Xiao. Solving Linear Arithmetic Constraints for User Interface Applications: Algorithm Details. Technical Report 97-06-01, Department of Computer Science and Engineering University of Washington, 1997
[9] M. Chetty, K.P. Dabke. Symbolic computations: an overview and application to controller design. // Proceedings of IEEE sponsored international conference on Information, Decision and Control, Adelaide, 8th-10th February, 1999, pp.451–456.
[10] I. Fudos. Editable Representations For 2D Geometric Design. 1993
[11] B. V. Zanden, An incremental algorithm for satisfying hierarchies of multiway dataflow constraints. // ACM Transactions on Programming Languages and Systems. Volume 18, Issue 1. 1996. 30-72.
[12] A. Borning, R. Anderson, B. Freeman-Benson. Indigo: A Local Propagation Algorithm for Inequality Constraints. // In Proceedings of the 1996 ACM Symposium on User Interface Software and Technology, 1996, pp. 129-136.
[13] M. Sannella. Skyblue: a multi-way local propagation constraint solver for user interface construction. // Proceedings of the 7th annual ACM symposium on User interface software and technology, 1994, pp. 137-146.
[14] J. H. Maloney. Using Constraints for User Interface Construction. Doctoral Thesis. University of Washington. 1992
[15] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni. Dynamic algorithms for classes of constraint satisfaction problems. Theor. Comput. Sci. 259, 1-2 (May. 2001), 2001. pp. 287-305.

Упаковка прямоугольников в полосу модифицированным методом Нелдера-Мида с использованием генетического алгоритма

С.А. Мартишин М.В. Храпченко

Аннотация. Исследуется задача упаковки прямоугольников в полубесконечную полосу. Известно, что эта задача является NP-трудной. Предложен новый эвристический алгоритм упаковки с использованием модифицированного метода Нелдера-Мида, генетического алгоритма и линейного программирования. Проведенное экспериментальное исследование предложенного алгоритма на известных тестовых примерах демонстрирует его преимущества. Предложены перспективные направления его использования для упаковки произвольных выпуклых многоугольников. Показана принципиальная возможность распараллеливания данного алгоритма.

1. Введение

Задача упаковки прямоугольников в одну полубесконечную полосу (так называемая strip packing problem) возникает в различных сферах, начиная от элементарного раскроя материала и размещения микросхем до размещения вычислительных задач (приложений) на кластере.

Данная задача достаточно хорошо исследована [2,3,17]. Однако, в связи с тем, что рассматриваемая задача является NP-трудной, существует и продолжает появляться большое количество методов ее решения, основанных на различных подходах, позволяющих сократить объем вычислений. Для решения задачи были предложены методы линейного и динамического программирования [9], эвристические методы, наиболее известным из которых является Bottom-Left (BL) алгоритм [3], достаточно полный обзор методов приведен в [19]. Некоторое время назад появились работы, в которых использовались эволюционные методы, в частности, генетические алгоритмы (например, [5,6,12,29,30]). Известны также работы, использующие для решения рассматриваемой задачи алгоритмы имитации отжига [13] и искусственных нейросетей [22]. Все эти алгоритмы направлены на сокращение времени вычисления при сохранении приемлемых характеристик упаковки.

Кроме того, наряду с двумерной задачей в настоящее время вызывают интерес алгоритмы для случая произвольной размерности (см., например, [4]). Еще одно направление исследований связано с упаковкой различных геометрических фигур (например, сфер [8]). В последнее время возобновился интерес к упаковке кругов и квадратов в разнообразные виды областей (квадраты, правильные треугольники, круги). Хотя различные постановки задачи в этом направлении были сформулированы достаточно давно (см., например, [11, 18]), интерес к их решению продолжает оставаться высоким.

Предложенный в данной статье алгоритм основан на выборке при помощи генетического алгоритма подмножества прямоугольников, а затем их упаковке с использованием дополнительных построений (основанных на модифицированном алгоритме Нелдера-Мида [21]), которые позволяют в дальнейшем применить методы динамического и линейного программирования.

Как показывают теоретические исследования, широко известные эвристические алгоритмы полиномиальной сложности, например, Bottom-Left (BL) или Best-Fit Decreasing Heigth (BFDH), шельфовый, в которых осуществляется предварительное упорядочивание прямоугольников, на известных тестовых входных данных [7,15,16,23] не осуществляют оптимальную упаковку (что следует из известных оценок см., например, [19,24,25,28]). В то же время предложенный алгоритм на тех же тестовых данных позволяет получить наилучшую из теоретически возможных упаковок, что подтверждается экспериментальными данными.

Данный алгоритм позволяет при помощи однотипных вычислений производить упаковку в произвольном (n-мерном, n>2) пространстве и при соответствующей доработке (написании соответствующих формул ограничений) может быть использован для упаковки произвольных выпуклых многоугольников и сфер в произвольное число полос, а также производить упаковку произвольных выпуклых многоугольников в области различных видов: круги, квадраты, треугольники и т.д. Кроме того, алгоритм будет работать и в том случае, если некоторая подходящая упаковка для некоторой области уже имеется, и стоит задача на основе данной упаковки и дополнительного множества многоугольников построить упаковку для области иного вида. В этом случае можно изменить границы области упаковки, переписав граничные условия, и дальнейшая работа алгоритма будет производится для вновь построенной области.

2. Постановка задачи

Формально задача плотной упаковки прямоугольников на полубесконечной полосе для двумерного случая может быть сформулирована следующим образом. Дана последовательность прямоугольников $R=\{R_1,\ldots,R_n\}$, для которых заданы w_i, h_i – ширина и высота ***i***-го прямоугольника соответственно, и задана ***W*** – ширина полубесконечной полосы. Требуется найти

ортогональную упаковку без перекрытий данного набора прямоугольников, имеющую минимальную высоту. Под ортогональной упаковкой подразумевается упаковка, в которой прямоугольники расположены параллельно сторонам полосы. Любые вращения прямоугольников запрещены.

В основе алгоритма лежит последовательное выполнение шагов генетического алгоритма и алгоритма упаковки. Сначала производится выборка при помощи генетического алгоритма [1,10,20] подмножества прямоугольников для упаковки, затем выбранные прямоугольники упаковываются с использованием модифицированного алгоритма Нелдера-Мида [21]. Далее производится оценка результата упаковки. Если значение целевой функции генетического алгоритма удовлетворяет критериям, то данное подмножество прямоугольников считается упакованным. Далее последовательно повторяются шаги генетического алгоритма до завершения упаковки всех подмножеств прямоугольников из первоначального множества.

3. Основные понятия и используемые эвристики

3.1. Основные термины и шаги генетического алгоритма

Генетические алгоритмы широко используются в качестве эвристических методов решения NP-трудных задач [1,10,20].

В генетических алгоритмах используются следующие термины и понятия:

Хромосома – вектор (последовательность), содержащий набор значений. Каждая позиция в хромосоме называется геном. Набор хромосом – вариант решения задачи.

Целевая (fitness) функция – функция оценки приспособленности решений.

Основные операторы генетического алгоритма:

Отбор – селекция (reproduction, selection) осуществляет отбор хромосом в соответствии со значениями их функции приспособленности.

Скрещивание (crossover) – операция, при которой две хромосомы обмениваются своими частями.

Мутация – случайное изменение одной или нескольких позиций в хромосоме.

Генетический алгоритм состоит из последовательности шагов:

Шаг 0. Формирование случайной популяции, состоящей из определенного набора хромосом (особей).

Далее итеративно повторяются шаги 1-3.

Шаг 1. Селекция и скрещивание.

Шаг 2. Мутация.

Шаг 3. Отбор наилучших с точки зрения целевой функции хромосом и замена ими исходных хромосом в популяции.

Процесс работы генетического алгоритма продолжается вплоть до выполнения некоторого критерия останова (как правило, количества итераций). В каждом следующем поколении возникают новые решения задачи. Среди них могут присутствовать как плохие, так и хорошие, но благодаря отбору, число приемлемых решений будет возрастать.

3.2. Метод Нелдера-Мида

Метод симплексов (метод Нелдера-Мида) – достаточно часто используемый алгоритм нелинейной оптимизации, представляет собой численный метод для поиска минимума целевой функции в многомерном пространстве. Идея метода заключается в вычислении целевой функции в вершинах симплекса и последовательного перемещения симплекса в направлении оптимальной точки. Этот метод является одним из наиболее быстрых и наиболее надежных не градиентных методов многомерной оптимизации.

Симплекс или N-симплекс определяется как выпуклая оболочка N+1 точек, не лежащих в одной гиперплоскости. Эти точки называются вершинами симплекса.

Основные шаги метода:

Шаг 1. Вычисление значения функции в каждой вершине симплекса.

Шаг 2. Преобразование симплекса. В применении к данному алгоритму – отражение вершины симплекса (выбранная вершина отражается относительно центра противоположной грани). При выборе вершины, в которой целевая функция принимает наилучшее значение (наименьшее отрицательное значение) алгоритм работает быстрее, но вероятность попадания в локальный минимум возрастает. При выборе вершины, в которой целевая функция принимает наихудшее значение (наибольшее отрицательное значение), алгоритм работает медленнее, но надежнее.

Шаг 3. Если критерий завершения удовлетворен (см. ниже, в подробном описании метода), то поиск прекращается.

3.3. Применение задачи линейного программирования

После того как при помощи алгоритма Нелдера-Мида была получена упаковка, ее уплотнение производится с использованием линейного программирования. Высоту упаковки, полученную при помощи алгоритма Нелдера-Мида, примем за максимальную высоту. Минимальная высота полосы вычисляется как суммарная площадь прямоугольников, входящих в данную упаковку, деленная на ширину полосы. Зная ширину, максимальную и минимальную высоту полосы для упаковки, полученной при помощи алгоритма Нелдера-Мида, можно выписать соотношения, определяющие взаимное расположение прямоугольников и их расположение относительно границ полосы.

Сначала запишем соотношения для взаимного расположения прямоугольников (как соотношения между их центрами) в виде линейных неравенств, связанных отношением «ИЛИ» (1).

$$
\begin{aligned}
&-x_i + x_j + \frac{w_i}{2} + \frac{w_j}{2} \le 0 \\
&x_i - x_j + \frac{w_i}{2} + \frac{w_j}{2} \le 0 \\
&-y_i + y_j + \frac{h_i}{2} + \frac{h_j}{2} \le 0 \\
&y_i - y_j + \frac{h_i}{2} + \frac{h_j}{2} \le 0,
\end{aligned}
\qquad (1)
$$

где i и j – номера прямоугольников.

Пусть H- высота упаковки, полученная при помощи алгоритма Нелдера-Мида. Тогда для каждого прямоугольника можно выписать неравенства (2) (как соотношения между границами полосы и центрами прямоугольников), связанные отношением «И»:

$$
\begin{aligned}
&\frac{w_i}{2} - x_i \le 0 \\
&x_i + \frac{w_i}{2} - W \le 0 \\
&\frac{h_i}{2} - y_i \le 0 \\
&y_i + \frac{h_i}{2} - H \le 0,
\end{aligned}
\qquad (2)
$$

где i – номер прямоугольника.

Для определения совместности системы будем применять задачу ЛП.

Алгоритм:

Шаг 1. Для каждой пары i и j выбирается только одно из неравенств (1) и все неравенства (2). Если в (1) выполняется более одного неравенства, выбирается неравенство для y.

Шаг 2. Проверяется совместность системы. Если система совместна, то переходим на Шаг 3, иначе полагаем, что оптимальная упаковка найдена.

Шаг 3. Уменьшаем высоту полосы H и переходим на Шаг 1 с новой высотой H.

Таким образом, уменьшая высоту полосы методом дихотомии между полученной при помощи алгоритма Нелдера-Мида и минимальной, и решая задачу линейного программирования для определения совместности системы неравенств, можно получить минимальную высоту полосы, соответствующую данному порядку расположения прямоугольников.

4. Предлагаемый алгоритм

4.1. Основные шаги генетического алгоритма

Шаг 1. Сортировка.

На данном шаге основной задачей является упаковка тех прямоугольников, у которых ширина сопоставима с шириной полосы. Их упаковка очевидна. Таким образом, выбираются и упаковываются все прямоугольники, разность между шириной которых и шириной полосы меньше некоторого числа δ, которое зависит от ширины полосы (в рассматриваемых примерах δ была на три порядка меньше ширины полосы). Упакованные объекты выводятся из дальнейшего рассмотрения.

Следующие шаги непосредственно описывают работу генетического алгоритма.

Шаг 2. Формирование популяции.

Формируем подмножества прямоугольников (особи популяции). Случайным образом выбираем k прямоугольников. Параметр k выбирается исходя из следующих соображений: быстрота упаковки (время вычисления, зависящее от конкретных вычислительных ресурсов) и их размеры (если размер упаковываемых прямоугольников значительно меньше ширины полосы, то число прямоугольников в подмножестве должно быть по крайней мере таким, чтобы их суммарная ширина была не меньше ширины полосы).

На тестируемых примерах на персональном компьютере (Pentium IV или аналогичном с процессором AMD) величина k находилась в пределах от 10-15 до 50. Аналогичным образом формируем подмножества до тех пор, пока все прямоугольники в совокупности не будут исчерпаны. Последнее подмножество будет, возможно, меньше k, но его в популяцию можно не включать. В итоге имеем популяцию размера $\lfloor n/k \rfloor$.

Шаг 3. Вычисление целевой функции.

Проводим упаковку подмножеств с использованием метода Нелдера-Мида. В качестве целевой функции берем разницу между суммой площадей упакованных прямоугольников и площадью полосы, содержащую данную упаковку. Далее необходимо определить, есть ли подмножества (особи популяции), упаковка которых не требует улучшения. Если есть

подмножество (подмножества) у которой целевая функция не больше некоторого числа Δ, выраженного в процентах, то считается, что данная упаковка выполнена и данное подмножество исключается из дальнейшего рассмотрения (особь исключается из популяции). Число Δ выбирается по возможности минимальным. Очевидно, что полное совпадение является крайне редким случаем, поэтому разумными границами для Δ является $0 \le \Delta \le 10\%(15\%)$. Затем из оставшейся совокупности необходимо выбрать аналогичные наборы прямоугольников (даже если они входят в различные подмножества), поскольку для них упаковка уже найдена. После этого, если необходимо создать новые подмножества из оставшейся совокупности прямоугольников, возвращаемся к Шагу 2, в противном случае переходим к Шагу 4.

Шаг 4. Работа генетического алгоритма.

Работа генетического алгоритма начинается с турнирного отбора. В данном случае применяется самый простой вариант отбора. Процедура **Турнирный отбор** реализована для *m*=2 и состоит из следующих шагов.

a) Случайным образом выбираются два подмножества (две особи популяции) случайным образом и сравниваются по значению фитнесс-функции. Оставляем лучшую.
b) Аналогичным образом выбираем еще две особи, следя за тем, чтобы они не совпали с ранее выбранными. Сравниваем их таким же способом и оставляем лучшую.
c) Полученную пару подвергаем скрещиванию. Случайным образом выбираем количество генов (прямоугольников), которыми особи будут обмениваться (не менее 1 и не более *k-1*).

Случайным образом выбираем точку скрещивания (номер прямоугольника с которого будет произведен обмен прямоугольниками), меняем прямоугольники между подмножествами (особями популяции), считаем целевую функцию для двух вновь получившихся подмножеств. Замену потомками родителей будем производить в любом случае, когда нет совпадений в оставшейся части популяции. В результате получаем две новых особи популяции. Считаем для них целевую функцию (см. Шаг 3) и переходим к Шагу 5.

Шаг 5. Итерации генетического алгоритма. На каждой итерации повторяем Шаг 4 (число итераций в предложенном алгоритме зависит от размера популяции, предполагаем, что размер число итераций равно размеру популяции*10). Предполагается, что при скрещивании различных особей удастся получить упаковку, удовлетворяющую целевой функции. Если удалось получить соответствующую упаковку, то возвращаемся к Шагу 2. Если нет, то применяем мутацию.

Шаг 6. Мутация. Мутация должна выполняться достаточно редко, вероятность ее выполнения в данном алгоритме 1%. За счет нее пробуем улучшить упаковку путем внесения изменения в две особи. Мутацию проводим следующим образом: в случайно выбранной особи случайным образом выбираем прямоугольник и меняем его на произвольный прямоугольник другой произвольно выбранной особи. Вторая особь получает вместо своего прямоугольника прямоугольник первой особи. Считаем целевую функцию, (см. Шаг 3) и переходим к Шагу 5.

Шаг 7. Изменение параметров генетического алгоритма. Если предыдущие шаги не дали приемлемого результата и остались неупакованные прямоугольники, для которых не удалось построить упаковку, отвечающую данной целевой функции, пробуем изменить параметры генетического алгоритма, а именно число прямоугольников в особи (например, $k \pm 1, k \pm 2, \ldots$ и т.п.). Число k не должно быть меньше или больше некоторого разумного числа (в данной конкретной реализации не меньше 5, не больше 55). Возвращаемся к Шагу 2. В конце работы алгоритма проверяем, остались ли неупакованные прямоугольники. Упаковываем все оставшиеся прямоугольники по методу Нелдера-Мида. Заметим, что, если на Шагах 3 и 4 не возникают приемлемые упаковки, то следует изменить параметр k после первых нескольких итераций.

Далее заполняем полосу полученными упаковками, строим ограничения для задачи линейного программирования. Это позволяет осуществить общую упаковку полученных упаковок прямоугольников (об этом будем сказано ниже).

Таким образом на вход подаются значения n – общее число прямоугольников, k – число прямоугольников в упаковке:

Формирование популяции:

```
for i=1 to N /*  N = ⌊n/k⌋  */
      do
            Call RecPack() /*вызов программы упаковки*/
      end
```

Итерации генетического алгоритма

```
for i=1 to i=NumberOfGeneration do
      begin
            for j=1 to j=NumberOfTournamentSelection do
                  begin
                        Call TournamentSelection()
                        Call RecPack()
                  end
            Call Mutation()
            Call RecPack()
      end
```

4.2. Основные шаги модифицированного алгоритма Нелдера-Мида

Алгоритм используется для получения плотной упаковки k прямоугольников на шаге 2 генетического алгоритма. Дано k – число прямоугольников. Для описания взаимного расположения прямоугольников на плоскости будем использовать $2k$-мерное пространство. Координаты центров прямоугольников будем интерпретировать как точку $2k$-мерного пространства (x_1,y_1,x_2, y_2,…, x_k,y_k). Здесь (x_1,y_1) – координаты центра первого прямоугольника, (x_2, y_2) – координаты центра второго прямоугольника и т.д.

Для применения алгоритма Нелдера-Мида важнейшими шагами являются выбор начальной точки и критерий преобразования симплекса. В алгоритме, изложенном ниже, предложен новый способ выбора начальной точки, который позволяет двигаться к решению не выходя за пределы построенной допустимой области в пространстве размерности $2k+2$. Сложность выбора критерия преобразования симплекса связана с необходимостью предотвратить преждевременную остановку алгоритма из-за попадания симплекса в локальный минимум.

Шаг 1. Построение правильного симплекса в пространстве размерности $2k + 2$. При построении симплекса используем свойство, что проекция любой его вершины на противоположную грань совпадает с центром масс этой грани.

Шаг 2. Построение ограничений на расстояния между центрами прямоугольников (каждого с каждым). Ограничения для каждой пары имеют вид (1) (см. выше):

$$-x_i + x_j + \frac{w_i}{2} + \frac{w_j}{2} \le 0$$

$$x_i - x_j + \frac{w_i}{2} + \frac{w_j}{2} \le 0$$

$$-y_i + y_j + \frac{h_i}{2} + \frac{h_j}{2} \le 0$$

$$y_i - y_j + \frac{h_i}{2} + \frac{h_j}{2} \le 0,$$

Естественно, достаточно выполнения хотя бы одного из этих ограничений для пары точек i и j, а также соотношений (2).

В пространстве размерности $2k+1$ выберем точку O_1 с координатами (0,...,0, C_1), C_1>0. Каждое такое ограничение (в пространстве размерности $2k$) изменим так, чтобы в пространстве размерности $2k+1$ оно проходило через точку O_1. В пространстве размерности $2k+2$ выберем точку O_2 с координатами (0,…,0, C_1, C_2), C_2>0. Построим сферу с центром в точке O_2 и радиусом R (R<

C_2). Каждое ограничение (в пространстве размерности *2k+1*) изменим так, чтобы в пространстве размерности *2k+2* оно касалось сферы с центром в точке O_2. Касательные ограничения строятся таким образом, чтобы точка O_2 лежала в отрицательном (то есть допустимом) полупространстве для каждого ограничения. Как показывает опыт вычислений, параметры R и C_2 должны быть связаны следующим соотношением: $0.0005 \leq R/C_2 \leq 0.001$. Параметр C_1 можно выбрать приблизительно в 50-100 раз больше ширины полосы. В этом случае, как показывают результаты экспериментов, полученная упаковка будет наилучшей по сравнению с упаковками, которые были получены за то же время с другими параметрами.

Шаг 3. Построение ограничений для расстояний между центрами прямоугольников и сторонами полосы (для каждого прямоугольника со всеми сторонами). Эти ограничения преобразуем в ограничения пространства *2k+2*, как это было описано на предыдущем шаге.

Шаг 4. Поиск максимальной и минимальной высоты полосы.

Шаг 5. Вычисление текущей высоты полосы как среднего значения между максимальной и минимальной высотой полосы.

Шаг 6. Вписываем правильный симплекс в сферу радиуса $r = \frac{R}{\sqrt{8}}$, центр которой находится в точке с координатами (0,…,0, C_1-r, -r). Увеличиваем размер допустимой области путем сдвига ограничений на R. Замкнутые полупространства, которые имеет вид $x_{2k+1} \leq C_1+R$ и $x_{2k+2} \leq R$, также становятся ограничениями.

Шаг 7. Вычисляем целевую функцию для каждой вершины симплекса. Целевая функция ***C*** – расстояние от вершины до ограничений. По построению точка не нарушает ограничение, если значение целевой функции отрицательно (чем меньше значение целевой функции, тем оно ближе к искомому решению). Уменьшаем размер допустимой области на величину, равную модулю целевой функции, умноженному на коэффициент ***Q*** (***Q*** по порядку равно 10^{-5}).

Шаг 8. Сортировка вершин симплекса в зависимости от значения целевой функции от максимальных к минимальным значениям.

Шаг 9. Отражаем *i*-ую вершины симплекса относительно своей противоположной грани и получаем точку I. Если значение целевой функции для точки I строго отрицательно и разность между целевой функцией и сдвигом меньше ε (чем меньше ε, тем точнее работа алгоритма), то решение найдено. В этом случае проектируем точку I на гиперплоскость $x_{2k+2}=0$ (точка I_1). Затем находим пересечение прямой (I_1, O_1) с гиперплоскостью $x_{2k+1}=0$ (точка I_2). Точка I_2 будет являться решением и алгоритм завершает работу.

Шаг 10. Если значение целевой функции в точке I отрицательно и *i*-ая вершина симплекса не была получена из точки I на предыдущем

преобразовании симлекса и выполнено хотя бы одно из следующих условий 1 или 2, то тогда *i*-ая вершина симплекса заменяется своей противоположной точкой и осуществляется переход на Шаг 8.

Условие 1: значение целевой функции в точке I меньше значения целевой функции в *i*-ой вершине симплекса.

Условие 2: *2k+1* координата точки I меньше *2k+1* координаты *i*-ой вершине симплекса.

В противном случае:

- Если не все вершины симплекса были проверены, то переход на Шаг 9.
- Если ни одна вершина симплекса не была улучшена, то урезание симплекса в два раза относительно вершины с наименьшей (лучшей) целевой функцией.

Шаг 11. Если длина ребра симплекса больше ε_s то переход на Шаг 7. В противном случае решение не может быть найдено и алгоритм завершает работу.

На практике при значении ε от 0.001 до 0.0001 за разумное время были получены упаковки, улучшить которые невозможно (см. таблицу 1). При этом $\varepsilon_s = 10^{-9}$.

Следует заметить, что упаковка большого количества прямоугольников приводит к росту вычислительной сложности алгоритма. Поскольку число ограничений оценивается как *k(k-1)/2+4k*, очевидно, что пропорционально увеличивается и количество арифметических операций. На практике это приводит к значительному увеличению времени с ростом числа упаковываемых прямоугольников. Причем за счет того, что появляются дополнительные ограничения, связанные с возрастанием размерности пространства, длительность выполнения возрастает не квадратично, а еще с некоторым коэффициентом, зависящим от размерности задачи.

4.3. Алгоритм уплотнения упаковки

Поскольку для работы алгоритма Нелдера-Мида необходимо, чтобы симплекс имел ненулевую длину ребра, перед упаковкой прямоугольников требуется произвести их урезание. Впоследствии для того, чтобы вернуться к прежним размерам прямоугольников будет использоваться алгоритм уплотнения упаковки.

4.3.1. Проверка существования допустимой области системы линейных неравенств

Шаг 1. Аналогично Шагу 2 алгоритма пункта 4.2 строим ограничения.

Шаг 2. Выбираем точку касания со сферой T на одном из ограничений. Строим *2k-1* гиперплоскостей, которые вместе с гиперплоскостью, соответствующей *i*-му ограничению находятся в общем положении и проходят через точку T. Обозначим такое множество гиперплоскостей через F, а сами гиперплоскости назовем дополнительными. Кроме того, построим гиперплоскость, проходящую через точку T и параллельную гиперплоскости $x_{2k+2}=0$. Пересечением этих *2k+1* гиперплоскостей будет являться прямая *l*.

Шаг 3. Находим ограничение, точка пересечения которого (точка P) с прямой *l* является ближайшей к точке T. Заменяем произвольную дополнительную гиперплоскость из множества F на гиперплоскость, которая соответствует найденному ближайшему ограничению. Если такая замена для множества F является первой, то при замене сдвигаем найденное ближайшее ограничение так, чтобы оно проходило через середину отрезка [T,P]. Вновь построенная точка P' является серединой этого отрезка, затем переносим точку T в точку P' и получаем новую прямую *l*.

Шаг 4. Если множество F еще содержит дополнительные гиперплоскости, то переходим на Шаг 3.

Шаг 5. В результате предыдущих действий будет получена точка T, через которую проходит *2k+1* ограничений. Эти *2k+1* находящихся в общем положении ограничений дают в своем пересечении прямую. Находим точки пересечения этой прямой с ограничениями. Выбираем то ограничение, расстояние до которого от точки T вдоль прямой минимально и у найденной точки пересечения x_{2k+2} координата меньше, чем у точки T. Найденная точка становится точкой T.

Шаг 6. Через точку T будет проходить *2k+2* ограничений. По очереди сдвинем все ограничения, кроме вновь найденного и сдвинутого на Шаге 3, в его отрицательное полупространство. Так как все эти *2k+2* ограничения линейно независимы, то они в пересечении дадут точку, обозначим ее через P. Построим прямую *l*, проходящую через точки T и P. Пересечем эту прямую со всеми ограничениями, которые будут включать и ограничение, задаваемое гиперплоскостью $x_{2k+2}=0$. Будем рассматривать только те точки пересечения, у которых x_{2k+2} координата меньше, чем у точки T. Точкой P становится точка, ближайшая к точке T. Если перебрав *2k* ограничений точку P найти не удалось, то то допустимая область отсутствует и алгоритм завершает работу. Ограничение, при помощи которого была построена точка T, заменяется вновь найденным ограничением, при помощи которого была построена точка P. Точка P становится новой точкой T. Если точка T не принадлежит $x_{2k+2}=0$, возвращаемся к началу шага 6.

Шаг 7. Находим пересечение луча O_1T с гиперплоскостью $x_{2k+1}=0$. Найденная точка пересечения есть искомое решение.

4.3.2. Алгоритм уплотнения упаковки

В основе алгоритма уплотнения упаковки лежит метод линейного программирования.

Шаг 1. Выбираем упаковку, генерируем ограничения, описывающие взаимное расположение всех прямоугольников относительно друг друга внутри упаковки и каждого прямоугольника относительно сторон полосы. Максимальной высотой будет полученная высота упаковки, а минимальной – теоретически возможная.

Шаг 2. Выбираем текущую высоту как среднее значение между минимальной и максимальной высотой.

Шаг 3. Находим точку внутри допустимой области.

Шаг 4. Если допустимая область существует, то максимальную высоту делаем текущей, иначе минимальную высоту делаем текущей. Если разность между минимальной и максимальной высотой больше некоторого числа ε, переходим на Шаг 2, в противном случае алгоритм завершает свою работу.

Следует также заметить, что для увеличения скорости работы алгоритма можно увеличить допустимую область. Для этого необходимо увеличить ширину полосы (например, на 1-3%), а ширину прямоугольников уменьшить (на 1-2%). Далее алгоритм уплотнения упаковки, используя реальные размеры полосы, прямоугольников и данные о взаимном расположении прямоугольников, проверяет возможность данной упаковки. Если такая упаковка возможна, то решение найдено. Если увеличение ширины полосы и уменьшение ширины прямоугольников приводят к невозможности реализации такой упаковки, то упаковку необходимо проводить с исходными данными.

5. Результаты реализации

Для реализации алгоритма были написаны следующие подпрограммы: упаковка с помощью модифицированного алгоритма Нелдера-Мида, генетический алгоритм и уплотнение упаковки при помощи задачи линейного программирования. Для наглядного представления результатов тестирования основной программы упаковки была написана вспомогательная программа графического отображения расположения прямоугольников на плоскости. Используемый язык программирования – C++. Общий объем программного кода составляет порядка 5000 строк.

Сравнение проводилось со стандартно реализованными алгоритмами BL, BLDW и шельфовым.

Плотность упаковки вычисляется следующим образом: вычисляется общая площадь прямоугольников, затем вычисляется занимаемая данной упаковкой площадь на полосе (ширина полосы умножается на высоту упаковки) и берется их соотношение, выраженное в процентах.

В качестве тестовых данных были использованы известные тесты для двумерной задачи упаковки, подробно рассмотренные в [7,15,16,23], а также специальным образом подобранные примеры.

Для проведения численных экспериментов использовался персональный компьютер с процессором AMD Atlon 2600+ и 512Мб DRAM.

На рисунке 1 показан результат предложенного алгоритма, время работы алгоритма 13,7 секунды. Плотность упаковки 96% и является оптимальной для специальным образом подобранных восьми прямоугольников, размеры которых равны 2.95*3.0, 4.95*4.0, 6.95*10.0, 0.95*7.5, 4.95*2.0, 0.95*7.5, 4.95*2.0, 0.95*7.5 и шириной полосы 10.

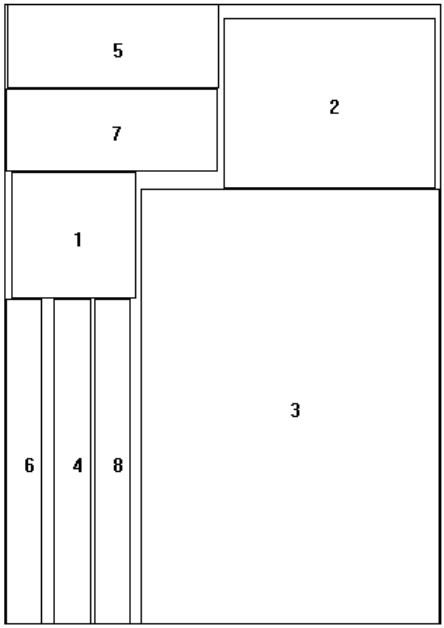


Рис. 1. Упаковка восьми специально подобранных прямоугольников.

Алгоритм BLDW выполнял бы упаковку в следующем порядке: прямоугольник 3 помещается в левом нижнем углу, далее создается новый уровень для прямоугольников 2, 5 и 7, затем прямоугольник 1 помещается на тот же уровень, что и прямоугольник 3, а для прямоугольников 4, 6 и 8 пришлось бы открывать новый уровень. Плотность упаковки – 63%.

Естественно, в случае применения алгоритма BL on-line имеется теоретическая возможность получения результата, аналогичного работе модифицированного алгоритма Нелдера-Мида в том случае, если

прямоугольники предварительно не сортировать по ширине, а подавать их на вход в соответствующем порядке (например, 2,5,7,3,1,4,6,8).

Но вероятность появления прямоугольников в необходимом порядке даже для примера из 8 штук составляет примерно 0,5%. Очевидно, с ростом числа прямоугольников она еще уменьшится.

Очевидно также, что и для алгоритма BFDH легко подобрать пример, когда он позволяет получить плотность упаковки 66,67%, в то время, как предложенный алгоритм дает возможность получить плотность упаковки 100%.

Таким образом в рассмотренных выше случаях предложенный алгоритм дает возможность получить наилучшую упаковку, в то время как остальные рассматриваемые алгоритмы получение наилучшей упаковки не гарантируют.

На рисунке 2 представлен результат упаковки тестового примера (см. [15] C1(P3)) из 16 прямоугольников. Известно, что оптимальная высота упаковки 20.

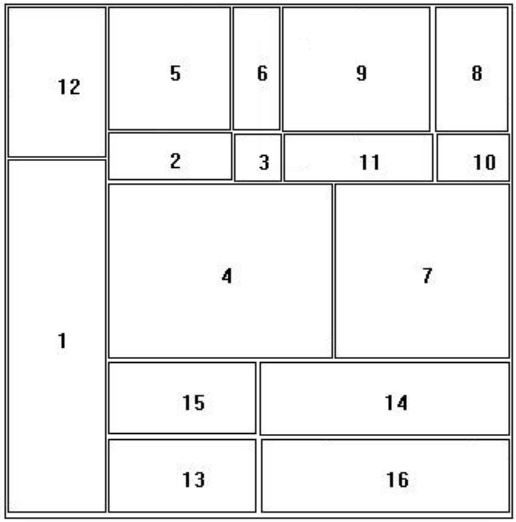


Рис. 2. Оптимальная упаковка 16 прямоугольников тест C1(P3).

В [7] приведены данные о выполнении различных тестов, предложенных в [15,16,23], в том числе и данного, которые показывают, что оптимальной плотности упаковки рассматриваемые алгоритмы не достигают (см таблицу 1), в то время как предложенный алгоритм позволяет ее получить.

Таблица 1. *Фрагмент таблицы, приведенной в [7]. Сравнение в % наилучшей упаковки с упаковкой bottom-left.*

	C1			C2			C3		
Тест	P1	P2	P3	P1	P2	P3	P1	P2	P3
Число прямоугольников	16	17	16	25	25	25	28	29	28
BL	45	40	35	53	80	67	40	43	40
BL-DW	30	20	20	13	27	27	10	20	17
BL-DH	15	10	5	13	73	13	10	10	13

Разумеется, с увеличением роста числа прямоугольников характеристики упаковки, полученной при помощи таких алгоритмов, как BL, BLDW, BLF, шельфового и др. улучшаются. Однако для тестов от 5 до 50 прямоугольников наилучшая из возможных упаковка была получена при помощи предложенного алгоритма.

Для получения окончательного вида упаковки, представленной на рисунках 1 и 2, была использована программа уплотнения упаковки, реализованная по соответствующему алгоритму (см. п. 4).

Ниже на рисунке 3 показан результат упаковки специальным образом подобранных 30 прямоугольников различной формы до и после применения уплотнения с использованием задачи линейного программирования. Высота левой упаковки до применения задачи линейного программирования (84%), правой – после применения задачи линейного программирования (95%), таким образом разница составляет 11%, что является существенной величиной.

Далее приведены данные о времени выполнения предложенного алгоритма и его сравнение по времени выполнения и плотности упаковки с алгоритмами: BL, модификацией BL алгоритма с предварительным упорядочиванием прямоугольников по невозрастанию ширины BLDW и шельфовым. Тестовые примеры были взяты из [15,16,23], время выполнения для предложенного алгоритма соответствовало случаю достижения наилучшей (теоретически возможной) упаковки.

Напомним, что для проведения численных экспериментов использовался персональный компьютер с процессором AMD Atlon 2600+ и 512Мб DRAM, на котором были получены приведенные ниже временные оценки. Очевидно, что при использовании ПК с большей частотой и оперативной памятью время вычисления снизится).

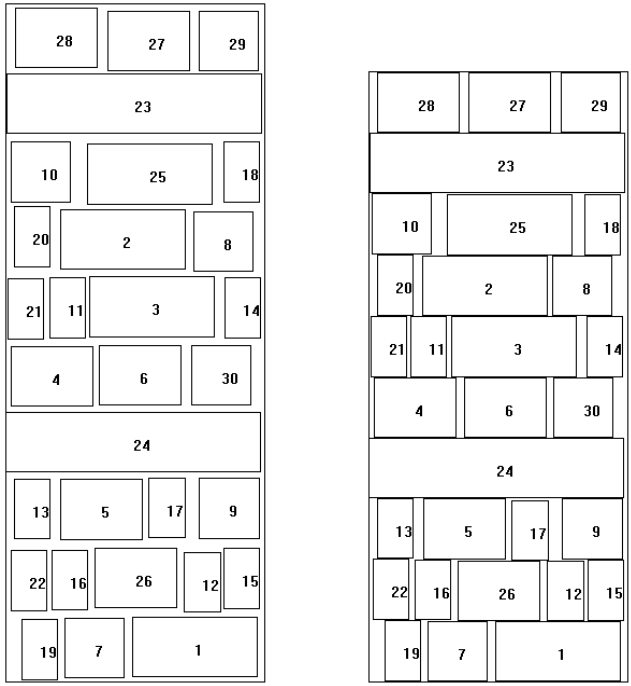


Рис. 3. Упаковка до и после уплотнения с использованием задачи ЛП.

В таблице 2 представлено среднее время работы алгоритма, поскольку в некоторых случаях обработка локальных минимумов может привести к увеличению времени работы. С другой стороны, некоторые упаковки могут быть подсчитаны за значительно меньшее время. Статистические данные также были получены на тестовых примерах [15, 16, 23] для различного числа прямоугольников.

***Таблица 2**. Среднее время работы модифицированного алгоритма Нелдера-Мида тестовые примеры из [15, 16, 23].*

Число прямоугольников	Среднее время упаковки
15 –20 штук	30 – 40 с
25 – 30 штук	1 – 5 мин
45 – 50 штук	10 – 40 мин
60 – 70 штук	3 час

Дальнейшее увеличение числа прямоугольников ведет к значительному увеличению времени работы модифицированного алгоритма Нелдера-Мида, поэтому в предлагаемом алгоритме используется генетический алгоритм, позволяющий осуществлять упаковку блоками по 20-30 штук и затем пытаться ее улучшить.

Поскольку время работы генетического алгоритма и задачи линейного программирования на порядок (или, в некоторых случаях, на два порядка) меньше, чем время работы модифицированного метода Нелдера-Мида, то при окончательной оценке времени выполнения предложенного алгоритма их оценкой времени можно пренебречь.

В следующей таблице 3 представлено среднее время работы алгоритмов и плотность упаковки с 16-17 прямоугольниками для предложенного алгоритма и алгоритмов BL и BLDW.

Таблица 3. *Сравнение времени выполнения и плотности упаковки для различных алгоритмов по тестам [15, 16, 23].*

Алгоритм	Время работы	Средняя плотность упаковки
BL	1с	70%
BLDW	1с	80%
Предложенный алгоритм	15с	97%

На небольшом примере видно, что предложенный алгоритм дает наилучшее решение, хотя и за значительно большее время. В таблице 4 приведено время работы алгоритмов со специально подобранными 10 прямоугольниками (см. Рис. 1), порядок следования прямоугольников 2,5,7,3,1,4,6,8.

В таблицах 3 и 4 в связи с небольшим числом прямоугольников генетический алгоритм не применялся.

Таблица 4*. Время. работы алгоритмов со специально подобранными 10 прямоугольниками*

Алгоритм	Время работы	Плотность упаковки
BL	1с	96%
BLDW	1с	63%
Shelf	1с	51%
Предложенный алгоритм	13с	96%

В таблице 5 при упаковке 500 прямоугольников (тесты, предложенные в [23]) генетический алгоритм используется.

***Таблица 5**. Среднее время работы алгоритмов с 500 прямоугольниками по тестам [23].*

Алгоритм	Время работы	Средняя плотность упаковки
BL	10 с	89%
BLDW	30 мин	89%
Предложенный алгоритм	60 мин	95%

В заключение заметим, что предложенный алгоритм может быть применен для произвольного числа прямоугольников и время его работы может быть легко спрогнозировано исходя из параметров алгоритма. Первоначальные параметры для работы генетического алгоритма следует выбирать в зависимости от соотношения размеров исходных прямоугольников и ширины полосы. В том случае, если размер прямоугольников не более, чем на порядок отличается от ширины полосы, первоначальное значение k можно выбрать, например, от 10 до 20. В этом случае время имеется возможность получить оптимальное решение за приемлемое время. Если ширина прямоугольников значительно меньше ширины полосы, число прямоугольников должно быть увеличено (в этом случае на практике достаточно взять $\varepsilon=0.001$, что значительно увеличивает скорость работы алгоритма упаковки).

Очевидно, что данный алгоритм обладает хорошими возможностями по распараллеливанию вычислений, поскольку после получения нескольких выборок для упаковки сама упаковка может проводится параллельно.

Данный алгоритм без модификаций может быть применен и для размерностей $n>2$. Для этого необходимо добавить ограничения для n-мерного случая в модифицированном методе Нелдера-Мида. Работа генетического алгоритма остается точно такой же. Также следует заметить, что изменив уравнения ограничений, мы можем применить данный алгоритм для упаковки произвольного типа выпуклых многоугольников в произвольного вида выпуклых областях.

Литература

[1] Back T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, New York, 1996.

[2] Baker B.S., Brown D.J. and Katseff H.P. A 5/4 algorithm for two-dimensional packing. Journal of Algorithms, 2:348--368, 1981.

[3] Baker B.S., Coffman E.G. Jr., Ronald L. Rivest: Orthogonal Packings in Two Dimensions. SIAM J. Comput. 9(4):846-855 (1980)
[4] Bansal L., Correre J.R., Kenyon C. and Sviridenko M. Bin Packing in Multiple Dimensions: Inapproximability Results and Approximation Schemes, Mathematics of Operations Research v.31 (2006), pp. 31-49.
[5] Bortfeldt A.; Gehring, H.: Two metaheuristics for strip packing problems. 5-th International Conference of the Decision Sciences Institute. Athen, Griechenland, 4.-7. Juli 1999.
[6] Bortfeldt A. Hermann G., A Parallel Genetic Algorithm for Solving the Container Loading Problem International Transactions in Operational Research Vol. 9 Issue 4 Page 497 July 2002
[7] Burke E.K., Kendall G. & Whitwell G., A New Placement Heuristic for the Orthogonal Stock-Cutting Problem, Operations Research, 52(4) (2004), 655-671.
[8] Danny Z. Chen, Xiaobo (Sharon) Huy, Yingping Huang, Yifan Li, Jinhui Xuz Algorithms for Congruent Sphere Packing and Applications, Symposium on Computational Geometry 2001: 212-221.
[9] Gilmore, P. C., Gomery, R. E., A Linear Approach to the Cutting-Stock Problem, Operations Research, 1961, Vol 9, pp 849-859.
[10] Goldberg D.E., Genetic algorithms in Search, Optimization and Machine Learning., Reading, MA: Addison-Wesley Publishing Company, 1989.
[11] Goldberg M., The packing of equal circles in a square, Math. Magazine 43, pages 24-30, 1970
[12] Goodman E.D, Tetelbaum A.Y. and Kureichik V.M. A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems, Release 3.01, Aug. 1, 1995, Technical Report 95-08-01, Intelligent Systems Laboratory and Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, 80pp.
[13] Dagli, C.H.; Hajakbari, A. Simulated annealing approach for solving stock cutting problem Systems, Man and Cybernetics, 1990. Conference Proceedings., IEEE International Conference on, Vol., Iss., 4-7 Nov 1990 Pages:221-223
[14] Dagli, C. H., Poshyanonda, P., "New Approaches to Nesting Rectangular Patterns"\, Journal of Intelligent Manufacturing, 1997, Vol. 3, No. 3, pp. 177-190.
[15] Hopper E. & Turton B.C.H., An Empirical Investigation of Meta-Heuristic and Heuristic Algorithms for a 2D Packing Problem, European Journal of Operational Research, 128(1)(2001), 34-57.
[16] Hopper E. & Turton B.C.H., Problem Generators for Rectangular Packing Problems, Studia Informatica Universalis, 2(1) (2002), 123-136.
[17] Kenyon, C., and Remila, E. A Near-optimal Solution to a Two-dimensional Cutting Stock Problem. Mathematics of Operations Research 25, 4 (Nov 2000), 645-656.
[18] S. Kravitz. Packing cylinders into cylindrical containers. Mathematics Magazine, 40:65-71, 1967
[19] Lodi A., Martello S., and Monaci M. Two-dimensional packing problems: a survey. European Journal of Operational Research, 141: 241-252,2002.
[20] Mitchell M. An Introduction to Genetic Algorithms. The MIT Press, Cambridge, MA, 1996.
[21] Nelder, J. A. and Mead. R. A Simplex Method for Function Minimization. Comput. J. 7, 308-313, 1965.
[22] Poshyanonda, P.; Bahrami, A.; Dagli, C.H. Two dimensional nesting problem: artificial neural network and optimization approach Neural Networks, 1992. IJCNN., International Joint Conference on, Vol.4, Iss., 7-11 Jun 1992 Pages: 572-577 vol.4

[23] Wang P.Y. & Valenzuela C.L., Data set generation for rectangular placement problems, European Journal of Operational Research, 134(2001), 378-391.
[24] Головистиков А.В., Задачи двумерной упаковки и раскроя: обзор. Информатика, выпуск N 20, 2008, с. 18-33.
[25] Жук С.Н. Онлайновый алгоритм упаковки прямоугольников в несколько полос с гарантированными оценками точности. Труды Института Системного программирования: Методы синтеза и анализа, т.12, 2007, с. 7-16.
[26] Жук С.Н. Приближенные алгоритмы упаковки прямоугольников в несколько полос. Дискретная математика, т. 18:1, 2006, с. 91-105.
[27] Канторович Л.В., Залгаллер В.А. Рациональный раскрой промышленных материалов. Наука. – Новосибирск, 1971, 299 с.
[28] Кузюрин Н.Н., Поспелов А.И. Вероятностный анализ шельфовых алгоритмов упаковки прямоугольников в полосу. Дискретная математика, т. 18:1, 2006, с. 76-90.
[29] Мухачева Э.А., Мухачева А.С., Чиглинцев А.В. Генетический алгоритм блочной структуры в задачах двумерной упаковки. Информационные технологии. Машиностроение. -М.:1999, N 11, с. 13-18.
[30] Мухачева А.С., Чиглинцев А.В., Смагин М.А., Мухачева Э.А.. Задачи двумерной упаковки: развитие генетических алгоритмов на базе смешанных процедур локального поиска оптимального решения. Приложение к журналу Информационные технологии. Машиностроение. -М.: 2001, N 10, 24 с.

Вероятностный анализ одного алгоритма упаковки прямоугольников в полосу

Н.Н. Кузюрин, А.И. Поспелов

Анотация. В статье предлагается и теоретически исследуется on-line алгоритм упаковки прямоугольников в полубесконечную полосу. Получено, что для незаполненной площади упаковок, порождаемых алгоритмом в среднем, справедлива оценка $O\left(N^{2/3}(\ln N)^{1/3}\right)$.

1. Введение

В задаче упаковки в полубесконечную полосу (strip packing) цель состоит в упаковке множества прямоугольников в вертикальную полосу единичной ширины так, что стороны прямоугольников должны быть параллельны сторонам полосы (вращения запрещены). При анализе по худшему случаю обычно минимизируют необходимую для упаковки высоту полосы, а при анализе в среднем — математическое ожидание незаполненной площади (от нижней границы прямоугольников в упаковке до верхней) [9, 4].

Таким образом входом задачи является конечный список прямоугольников $L = \{p_j\}_{j=1}^{N}$. Каждый прямоугольник p_j из списка L задан высотой h_j и шириной w_j. Эта задача возникает во многих контекстах и имеет много приложений, в частности, при разработке СБИС, построении оптимальных расписаний для кластеров и т.д. Ее частными случаями являются задача упаковки в контейнеры (bin packing) и задача об m-процессорном расписании Так, если высоты всех прямоугольников из входного списка равны, то получаем задачу об упаковке в контейнеры, где высота заполнения как раз и равна числу контейнеров. Эти задачи интенсивно исследовались ранее. В общей постановке задача об упаковке в полосу также привлекала внимание исследователей с начала 80-х годов [1]. Известно, что эта задача NP-трудна. Интерес, поэтому, представляет построение приближенных алгоритмов с гарантированными оценками точности.

Различные приближенные алгоритмы были предложены для этой задачи [1, 2, 14, 11]. С точки зрения приложений интерес представляют алгоритмы, работающие в *оперативном режиме*, т.е. размещающие очередной прямоугольник из списка, не используя информацию о последующих прямоугольниках (on-line алгоритмы).

В данной работе исследуется одна простая стратегия упаковки прямоугольников в полосу, в которой прямоугольники размещаются по мере поступления (т.е. в оперативном режиме), но при этом общее число прямоугольников известно заранее. Наш анализ качества алгоритмов в среднем заключается в оценке математического ожидания незаполненной площади после работы конкретного алгоритма (от основания полосы до верхней границы прямоугольников). Очевидно, что чем меньше отношение незаполненной площади к заполненной, тем выше качество упаковки.

Всюду в дальнейшем будем считать, что для каждого прямоугольника высота h_i и ширина w_i имеют равномерное распределение на отрезке $[0,1]$. Будем предполагать, что все случайные величины w_i, h_i — независимы в совокупности. В дальнейшем будем обозначать через Σ математическое ожидание площади, не заполненной прямоугольниками, между основанием полосы и верхней границей самого верхнего прямоугольника. Будем предполагать, что число прямоугольников N — бесконечно большая величина ($N \to \infty$). Отметим, что вероятностному анализу различных эвристик одно и двумерной упаковки посвящено много работ [3, 4, 5, 6, 7, 8].

2. Описание алгоритма

Прежде чем перейти к описанию алгоритма, введём некоторые обозначения. Будем обозначать через $[x]$ — максимальное целое, непревосходящее x. Пусть $\delta > 0$ — некоторый параметр, от которого будет зависеть поведение алгоритма. Будем предполагать, что мы знаем число прямоугольников N, которые надо будет разместить (как избавиться от данного предположения для подобного типа эвристик показано в [10]).

Далее, пусть $n(\delta) = N\delta + \sqrt{N\delta \ln N}$, а $h(\delta) = \frac{1}{2}n(\delta) + c\sqrt{n(\delta)\ln n(\delta)}$, где c — некоторая положительная константа, значение которой мы определим позже.

В основании полосы рассмотрим прямоугольную область высоты $kh(\delta)$ и ширины 1, где $k = [1/(2\delta)]$. Разобъём на прямоугольные области $\{P_i\}_{i=1}^{k}$ высоты $h(\delta)$ и ширины 1. Область P_i, $i = 1,2,\dots k$ разрежем на две подобласти P'_i и P'_{2k-i+1} высоты $h(\delta)$ и ширины $i\delta$, $1 - i\delta$, соотвественно. Таким образом, мы получим $2k$ областей $\{P'_i\}$ (см. рис. 1).

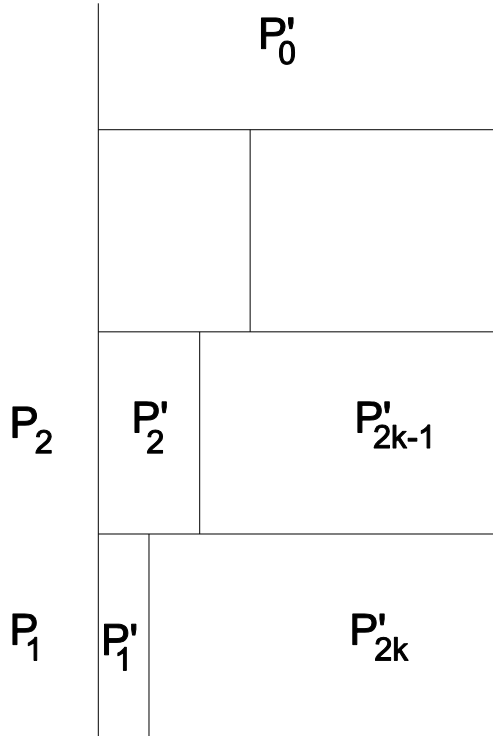


Рис. 1. Разбиение полосы на области.

Опишем теперь сам алгоритм упаковки. Алгоритм выбирает прямоугольники из списка по очереди и сразу же определяет место расположения, при этом прямоугольники упаковываются в одну из областей P'_i или в область P'_0 — часть полосы выше $kh(\delta)$. Упаковка происходит либо на основание области, либо на верхнюю границу упакованных в данную область прямоугольников. Пусть уже упаковано некоторое число прямоугольников и опишем упаковку очередного прямоугольника $p = (w, h) \in L$. Область для упаковки выбирается по следующим правилам.

• Если $w \leq 1 - \delta$, то рассматривается область P'_i, имеющая наименьшую ширину, превосходящую w.

- Если суммарная высота прямоугольников, упакованных к текущему моменту в P'_i, не превосходит величины $h(\delta) - h$, то упаковываем p в P'_i.

- Если суммарная высота больше $h(\delta) - h$, то p упаковывается в P'_0.

• если $h > 1 - \delta$, то p упаковывается в P'_0.

Алгоритм заканчивает работу, когда упакован последний прямоугольник из L. Ясно, что при выполнении правил упаковки внутренности упакованных прямоугольников не будут пересекать границы полос и пересекаться между собой. Будем обозначать описанный алгоритм $A1(\delta)$. Для краткости будем говорить, что прямоугольники $p = (w, h)$ является прямоугольником j-го типа (или $p \in T_j$), если $w \le 1 - \delta$ и ширина w больше ширины области P'_{j-1} и меньше ширины области P'_j.

3. Оценка качества алгоритма

Естественным критериями оценки качества упаковки, создаваемой алгоритмами подобного типа, является высота и незаполненная площадь. Зная ширину полосы и общую площадь всех упаковываемых прямоугольников, эти критерием могут быть пересчитаны один в другой. Понятно, что в общем случае описанный алгоритм не даёт оптимального качества упаковки с точки зрения этих критериев. Целью данной статьи является оценка эффективности данного алгоритма в среднем, точнее оценка величины

$$\Sigma = \mathbb{E}\left(H - \sum_{i=1}^{N} h_i w_i\right),$$

где H — общая высота упаковки.

Теорема Пусть $L = \{(h_i, w_i)\}_{i=1}^{N}$ — *набор прямоугольников, причём* h_i *и* w_i *независимы и равномерно распределены на отрезке* $[0,1]$, $\delta \geqslant c_0 N^{-1/2}$, *где* c_0 *— некоторая положительная константа, тогда для упаковки, получаемой с помощью алгоритма* $A1(\delta)$, *справедливо*

$$\Sigma = O\left(N\delta + \sqrt{\ln N}\sqrt{\frac{N}{\delta}}\right).$$

Ранее в [10] было показано, что для шельфового алгоритма, использующего эвристику First Fit $\Sigma = O(N^{3/4})$, а для шельфового алгоритма, использующего эвристику Best Fit $\Sigma = O(N^{2/3}(\log N)^{1/2})$.

Параметр δ может быть подбираться в зависимости от N. При этом эффективность алгоритма будет меняться. Понятно, что выбирая $\delta = (\ln N)^{1/3} N^{-1/3}$ можно получить для $A1(\delta)$ оценку, несколько улучшающую результаты, полученные в [10]

$$\Sigma = O\left(N^{2/3}(\ln N)^{1/3}\right).$$

Оценки получаемые в теореме предполагают, что число прямоугольников N известно заранее. Однако, как было показано в [10], если упаковывать

прямоугольники экспоненциально увеличивающимися по числу группами, алгоритм становиться оперативным и с той же оценкой качества.

Перейдем теперь к доказательству теоремы.

Доказательство теоремы Рассмотрим Σ. В силу свойств суммы и произведения математического ожидания и независимости случайных величин w_i, h_i имеем

$$\Sigma = \mathbb{E}(H - \sum_{i=1}^{N} h_i w_i) = \mathbb{E}H - \frac{N}{4}. \quad (1)$$

Оценим $\mathbb{E}H$. В силу используемого алгоритма упаковки высота H складывается из детерминированной величины $kh(\delta)$ и суммы высот прямоугольников, упакованных в область P'_0. Прямоугольники попадают в P'_0 либо если их ширина больше $1-\delta$, либо если не вместились в подходящую для них область P'_i. Оценим эти две составляющих по отдельности.

Математическое ожидание суммарной высоты прямоугольников, имеющих ширину больше $1-\delta$, равна

$$\mathbb{E}\left(\sum_{\{1 \le i \le N \mid w_i > 1-\delta\}} h_i\right) = \mathbb{E}\left(\frac{\mathrm{card}\{1 \le i \le N \mid w_i > 1-\delta\}}{2}\right) = \frac{N\delta}{2}. \quad (2)$$

Теперь оценим суммарную высоту остальных прямоугольников, попавших в P'_0. Пусть N_j — число прямоугольников j-го типа (т.е. область P'_j является минимальной, в которую они вмещаются по ширине). Поскольку вероятность попадания в подполосу равна δ, а общее число прямоугольников равно N, имеем

$$\mathbb{E}N_j = N\delta.$$

Пусть T'_j множество прямоугольников j-го типа, размещенных в P'_0. Обозначим через H_j суммарную высоту прямоугольников из T'_j, т.е.

$$H_j = \sum_{p \in T'_j} h.$$

Так как $T'_j \subset T_j$, математическое ожидание H_j может быть оценено следующим образом

$$\mathbb{E}H_j = \mathbb{P}\{T'_j \neq \emptyset\}\mathbb{E}\sum_{p \in T'_j} h \le \mathbb{P}\{T'_j \neq \emptyset\}\mathbb{E}\sum_{p \in T_j} h = \mathbb{P}\{T'_j \neq \emptyset\}N\delta/2. \quad (3)$$

Для того, чтобы оценить $\mathbb{P}\{T'_j \neq \emptyset\}$ рассмотрим два события. Первое — прямоугольников j-го больше чем $n(\delta)$. Второе — прямоугольников j-го не больше чем $n(\delta)$, но не всех их удалось разместить в P'_j. Понятно, что эти два события не пересекаются, и их объединение содержит событие $T'_j \neq \emptyset$. Таким образом, $\mathbb{P}\{T'_j \neq \emptyset\}$ можно оценить как

$$\mathbb{P}\{T'_j \neq \emptyset\} \le \mathbb{P}\{N_j > n(\delta)\} + \mathbb{P}\{N_j \le n(\delta)\}\mathbb{P}\left\{\sum_{i \in T_j} h_i > h(\delta) \quad N_j \le n(\delta)\right\}. \quad (4)$$

Для оценки первого из слагаемых в правой части (4) воспользуемся неравенством для вероятностей больших уклонений [12]:

$$\mathbb{P}\{|N_j - EN_j| > aEN_j\} \le 2\exp\{-(a^2/3)EN_j\},$$

или, вычисляя математические ожидания

$$\mathbb{P}\{|N_j - N\delta| > aN\delta\} \le 2\exp\{-(a^2/3)N\delta\}.$$

Выбирая $a = c\sqrt{\frac{\ln N}{N\delta}}$, получаем:

$$\mathbb{P}\{|N_j - N\delta| > c\sqrt{N\delta \ln N}\} \le 2\exp\{-(c^2/3)\ln N\}.$$

Откуда получаем, что

$$\mathbb{P}\{N_j > n(\delta)\} \le 2\exp\left\{-\left(\frac{c^2}{3}\right)\ln N\right\} = 2N^{-\frac{c^2}{3}}. \qquad (5)$$

Для оценки второго слагаемого в правой части (4) воспользуемся оценкой больших уклонений сумм независимых одинаково распределенных случайных величин $S_n = \sum_{i=1}^{n} x_i$, $a_i \le x_i \le b_i$ [13]:

$$\mathbb{P}\{S_n - ES_n \ge nx\} \le \exp\left\{-\frac{2n^2x^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right\}$$

В нашем случае $a_i = 0$, $b_i = 1$ для всех $i = 1, \dots, n$, $\mathbb{E}x_i = \mathbb{E}h_i = 1/2$, $n = n(\delta)$. Полагая $x = c\sqrt{\ln n(\delta)}/n(\delta)$, получаем:

$$\mathbb{P}\left\{\sum_{i \in T_j} h_i > h(\delta) \quad N_j \le n(\delta)\right\} \le \mathbb{P}\left\{\sum_{i=1}^{n(\delta)} h_i > h(\delta)\right\} \le$$

$$\le \exp\left(-\frac{2n(\delta)^2 \ln n(\delta) c^2}{n(\delta)^2}\right) = n(\delta)^{-2c^2}. \qquad (6)$$

Из (3), (4), (5) и (6) получаем, что

$$\mathbb{E}H_j \le N^{1-c^2/3} + n(\delta)^{1-2c^2}. \qquad (7)$$

В силу независимости случайных величин w_i, h_i, математическое ожидание вклада прямоугольников таких, что область P'_j является минимальной, в которую они вмещаются по ширине, не зависит от j. Поэтому из (1), (2) и (7) следует

$$\Sigma \le kh(\delta) + \frac{N\delta}{2} + k(N^{1-c^2/3} + n(\delta)^{1-2c^2}) - \frac{N}{4}.$$

Учитывая, что $\delta \geq c_0 N^{-1/2}$, и выбирая $c \geq 3/\sqrt{2}$, не сложно получить, что $N^{1-c^2/3} + n(\delta)^{1-2c^2}$ не превосходит некоторой константы c_1. Следовательно

$$\Sigma \leq c_1 + \left(\frac{1}{2}N\delta + \sqrt{N\delta \ln N} + c\sqrt{\left(N\delta + \sqrt{N\delta \ln N}\right)\ln\left(N\delta + \sqrt{N\delta \ln N}\right)}\right)/(2\delta) +$$

$$+ \frac{N\delta}{2} - \frac{N}{4} \leq c1\sqrt{N\delta \ln N} + c2\frac{N\delta}{2},$$

что даёт утверждение теоремы.

Литература

[1] B.S. Baker, E.J. Coffman and R.L. Rivest, Orthogonal packings in two dimensions, SIAM J. Computing, (1980) **9**, 846--855.

[2] B.S. Baker, J.S. Schwartz, Shelf algorithms for two dimensional packing problems. SIAM J. Computing, (1983) **12**, 508--525

[3] E.G. Coffman, Jr., C. Courcoubetis, M.R. Garey, D.S. Johnson, P.W. Shor, R.R. Weber, M. Yannakakis, Perfect packing theorems and the average-case bahavior of optimal and online bin packing, SIAM Review, (2002) **44** (1), 95--108.

[4] R.M. Karp, M. Luby, A. Marchetti-Spaccamela, A probabilistic analysis of multidimensional bin packing problems, In: Proc. Annu. ACM Symp. on Theorty of Computing, 1984, pp. 289--298.

[5] P. W. Shor, The average-case analysis of some on-line algorithms for bin packing. Combinatorica (1986) **6**, 179-200.

[6] P. W. Shor, How to pack better than Best Fit: Tight bounds for average-case on-line bin packing, Proc. 32nd Annual Symposium on Foundations of Computer Science, (1991) pp. 752-759.

[7] E. G. Coffman, Jr., D. S. Johnson, P. W. Shor and G. S. Lueker. Probabilistic analysis of packing and related partitioning problems, Statistical Science (1993) **8**, 40-47.

[8] E. G. Coffman, Jr., P. W. Shor. Packings in two dimensions: Asymptotic average-case analysis of algorithms, Algorithmica (1993) **9**, 253-277.

[9] J. Csirik, G.J. Woeginger, Shelf Algorithms for On-Line Strip Packing. Inf. Process. Lett. (1997), **63**(4), 171-175.

[10] Кузюрин Н.Н., Поспелов А.И., Вероятностный анализ шельфовых алгоритмов упаковки прямоугольников в полосу, Дискретная математика, 2006, т. 18, N 1, с. 76--90.

[11] C. Kenyon and E. Remila, A near optimal solution to a two-dimensional cutting stock problem, Mathematics of Operations Research, (2000) **25**, 645--656.

[12] R. Motwani and P. Raghavan, Randomized algorithms, Cambridge Univ. Press, 1995.

[13] W. Hoeffding, Probability inequalities for sums of bounded random variables, J. Amer. Sattist. Assoc., 1963, v. 58, N 301, pp. 13--30.

[14] E.J. Coffman, M.R. Garey, D.S. Johnson and R.E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms, SIAM J. Computing, (1980) **9**, 808--826.

Моделирование операционной семантики машинных инструкций

В.А. Падарян, М.А. Соловьев, А.И. Кононов
{vartan, eyescream, extreme}@ispras.ru

Аннотация. В работе предлагается модель, позволяющая описывать операционную семантику машинных инструкций для широкого класса целевых архитектур. Особенностью модели является то, что она предназначена для обратного по сравнению с классической компиляторной задачей тракта преобразований, но в то же время модель позволяет выполнять над ней различные оптимизирующие преобразования. Для описания целевой машины применяются внешние спецификации. Рассмотрена прототипная подсистема интерпретации модели.

1. Введение

Анализ бинарного кода, нацеленный на восстановление алгоритмов и повышение уровня их представления, включает в себя большое количество этапов. Часть из используемых этапов анализа применяется последовательно, с целью выделения интересующей части программы. Другие используются итеративно для восстановления отдельных аспектов семантики.

В настоящее время актуальна задача анализа кода, снабженного механизмами, препятствующими как отладке, так и статическому анализу. Примерами подобных механизмов являются различные запутывающие преобразования [1]. Значительная часть преобразований такого рода нацелена на противодействие статическому анализу. Так, преобразование «диспетчер» [2] не оказывает заметного усложняющего влияния при применении динамического анализа. Кроме того, использование динамического анализа в виде снятия трассы работы целевого процессора (стенда) и последующего ее анализа позволяет анализировать не только пользовательский код, но и код системный, например, драйверов.

В системе динамического анализа кода TrEx [3] применяется подход с использованием полносистемных симуляторов для сбора трасс. Среда поддерживает несколько целевых процессорных архитектур (в том числе Intel 64 и MIPS64) за счёт использования прослойки абстракции архитектуры. На данный момент указанная прослойка обладает возможностями декодирования инструкций в машинно-независимый вид и построения частичных графов зависимостей по данным отдельных инструкций. Указанные особенности

позволяют при реализации алгоритмов анализа, основанных на анализе зависимостей, абстрагироваться от целевой машины, что, в свою очередь, значительно упрощает разработку.

Возможность построения декомпозиции на зависимости позволяет проводить более точно различные виды анализа, в том числе слайсинг трассы [4, 5]. Тем не менее, вопрос повышения уровня представления выделенного алгоритма не может быть решен без возможности анализа семантики составляющих алгоритм инструкций по отдельности и в совокупности.

В настоящей работе предлагается модель, пригодная для описания операционной семантики машинных инструкций. Описывается алгоритм приведения бинарной машинной инструкции к модельному виду. Кроме того, рассматривается применение описываемой модели к задаче интерпретации инструкций.

Прежде чем перейти к изложению предлагаемой модели, необходимо указать требования, исходя из которых принимались решения при ее построении.

Модель должна позволять описывать операционную семантику как пользовательских, так и системных инструкций современных процессорных архитектур. Важно поддержать различные по своей природе процессорные архитектуры, как CISC, так и RISC. В набор архитектур, рассматривавшихся при построении модели, входят Intel 64, MIPS64, PowerPC, ARM v6, SPARC и Motorola m68k. Описание семантики инструкций этих архитектур должно быть достаточным для моделирования всех эффектов выполнения инструкции, в том числе побочных (влияние на слово состояния машины).

Необходимо обеспечить такой уровень модели, который позволял бы без существенных дополнительных затрат применять различные виды компиляторных оптимизаций, таких как удаление общих подвыражений или удаление мертвого кода. Это требование вызвано необходимостью противодействия тем видам запутывающих преобразований, которые намеренно усложняют выражения, входящие в операторы программы, или граф потока управления за счет введения дублирующих или нереализуемых путей. Наличие данных о семантике машинных инструкций, составляющих алгоритм, является необходимым требованием при решении этих задач. Тем самым, модель должна быть приближена к промежуточным представлениям, применяемым в современных компиляторах (SSA, трехадресный код и так далее).

Наконец, необходимо обеспечить возможность интерпретации модельного представления, а именно указать, каким образом из состояния машины перед выполнением моделируемой инструкции можно получить ее состояние после. Тем самым может быть обеспечен режим работы системы анализа, сходный с работой отладчика.

Дальнейшее повествование построено следующим образом. В разделе 2 приводится обзор связанных работ и выделяются те их особенности, которые

могут быть использованы при решении задачи моделирования семантики инструкций. В разделе 3 описывается построенная модель. Указываются отдельные объекты модели и их взаимосвязь. В разделе 4 дается краткое описание способа спецификации целевой машины и указывается способ построения модели инструкции при наличии такой спецификации. В разделе 5 кратко описывается прототип интерпретатора модели. В разделе 6 дается заключение и указываются направления дальнейших работ.

2. Обзор работ

Решаемая задача напрямую связана с двумя сферами компиляторных технологий: собственно построением компиляторов и бинарной трансляцией.

С точки зрения задачи построения компиляторов пересечение с настоящей работой находится на уровне промежуточного представления. Используя классификацию из [6], можно охарактеризовать это промежуточное представление как представление низкого уровня (LIR). В самом деле, исходными данными для моделирования являются машинные инструкции, оперирующие с конкретными регистрами и адресами в памяти (в противовес виртуальным регистрам или переменным). Несмотря на то, что такое представление может быть механически повышено до представления среднего уровня (MIR) за счёт того, что является его подмножеством, подобное механическое повышение бессмысленно, так ведет к повышению уровня представления лишь формально.

Одной из широко распространенных форм представления низкого уровня является форма RTL (register transfer list), применяемая, с различными модификациями, в большом количестве компиляторных продуктов, например, в открытом компиляторе GNU GCC [7]. В GCC также используется представление среднего уровня GIMPLE.

Следует учитывать, однако, что поставленная задача, в противовес задаче компиляции, предполагает обратный тракт: от машинных инструкций к более высоким представлениям. Такой тракт встречается в задаче бинарной трансляции, когда машинные инструкции одной процессорной архитектуры сначала повышаются до некоторого модельного представления, а затем используются для синтеза машинных инструкций другой архитектуры. Частным случаем можно считать среды для динамического инструментирования, в которых исходная и целевая архитектуры совпадают, например, Valgrind [8] и iDNA [9].

В работах [10, 11] описываются соответственно система статической и динамической бинарной трансляции. Обе системы среди поддерживаемых архитектур указывают и CISC-архитектуру Intel 64. В качестве промежуточного представления используется расширенная форма RTL, называемая авторами H-RTL. Следует отметить, что в данных работах рассматривается вопрос трансляции лишь пользовательских инструкций.

Среда Valgrind [8] использует в качестве промежуточного представления виртуальную RISC-машину с достаточно малым набором операций. При этом регистровый файл и пространство памяти целевой машины являются внешними по отношению к виртуальной машине, то есть трансляции не подвергаются. Valgrind не поддерживает системные инструкции. Кроме того, с точки зрения расширяемости среда Valgrind достаточно неудобна: трансляция в промежуточное представление и последующий синтез закодированы напрямую, а не при помощи внешних спецификаций. Кроме того, этапы декодирования инструкции и формирования ее модели не отделены друг от друга, что вносит дополнительные сложности.

Среди существующих систем бинарной трансляции реальный интерес в контексте данной работы представляют лишь те, которые, во-первых, в том или ином виде используют спецификации для описания машин и, во-вторых, способны проводить трансляцию из и в CISC-архитектуры (в частности, Intel 64). Трансляция в инструкции CISC-машины необходима для обеспечения эффективного решения задачи интерпретации. В то время как первое требование реализовано в достаточном количестве рассмотренных работ, второе значительно ограничивает этот список.

Проведенный обзор работ позволяет сделать вывод о необходимости реализации своей модели операционной семантики машинных инструкций, не обладающей недостатками рассмотренных работ и удовлетворяющей заявленным требованиям. Предлагаемая модель представляет собой комбинацию возможностей рассмотренных систем с дополнением оригинальными особенностями. Без изменений может быть использована модель адресных пространств, используемая в близких видах в [8] и [3]. Во многом удовлетворяет заявленным требованием модель, используемая в качестве промежуточного представления в [8], однако она должна быть дополнена возможностью подгрузки внешних спецификаций наподобие используемых в [11].

3. Модель операционной семантики

Перед изложением построенной модели необходимо обозначить некоторые сложности, с которыми приходится сталкиваться при моделировании современных наборов инструкций. В большей части указанные сложности относятся к CISC-архитектурам, в частности к Intel 64.

Первой сложностью является нетривиальный вид регистровых файлов современных процессоров. Так, регистры могут являться частями друг друга (Intel 64), располагаться в нескольких теневых наборах (MIPS64) или иметь организацию в виде регистровых окон (SPARC). Подобные идиомы в регистровом файле приводят к необходимости обеспечения достаточно гибкой его модели. Такой моделью может служить *модель адресных пространств* [3], используемая в TrEx при декомпозиции на зависимости и более подробно

описанная далее. Кроме того, сходный способ описания регистрового файла используется и в среде Valgrind.

Второй сложностью, характерной для CISC-архитектур, является наличие слова состояния машины, изменение которого в качестве побочного эффекта входит в семантику многих инструкций. В соответствии с требованием пригодности модели для проведения различных анализов необходимо исключить неявное обновление слова состояния. С другой стороны, некоторые из статусных флагов Intel 64, например флаг четности PF, сложно пересчитывать вручную (требуется как минимум 9 операций для обновления этого флага для 32-битного значения без применения таблиц пересчета). Таким образом, необходимо поддержать баланс между минимальным объемом побочных эффектов в операторах модели и объемом (с точки зрения числа примитивных операций) трансляций распространенных инструкций.

Указав основные сложности, перейдем теперь к описанию предлагаемой модели. Общая схема модели приведена на рис. 1.

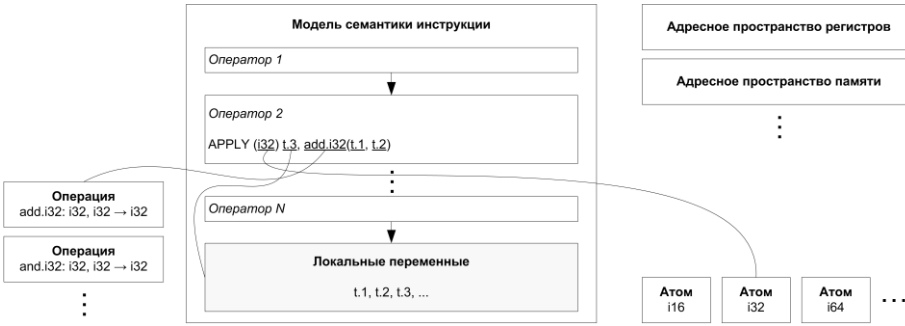


Рис. 1. Общая схема модели.

Модель семантики отдельной инструкции представляет собой упорядоченный набор *операторов*, описывающих действия, производимые процессором при выполнении данной инструкции. Отдельные операторы используются для применения *операций*, обращения в *адресные пространства* и ветвления. В качестве параметров и возвращаемых значений операций фигурируют *локальные переменные*. Осуществляется несложный контроль типов на уровне *атомов*.

Операторы читаются (при интерпретации — выполняются) последовательно. Область видимости локальных переменных ограничена единицей трансляции (то есть одной инструкцией). Следует пояснить, что в ходе дальнейших оптимизаций единица трансляции может быть расширена, однако первоначально производится построение модели для одной инструкции.

Каждая из указанных сущностей описана подробно в следующих подразделах.

3.1. Атомы

Первой из концепций, вводимых в модели, являются *атомы*. Под атомом будем понимать машинный тип, то есть элементарный блок данных, которым могут оперировать машинные инструкции.

Примерами атомов будут являться 32-битные целые числа, числа с плавающей точкой двойной точности, SIMD-векторы и т. п. Характеристикой атома является размер блока данных, им описываемого.

Некоторая избыточность, вводимая атомами, оказывается полезной для обеспечения корректности построенных моделей. В самом деле, можно было в качестве характеристики блока машинных данных использовать только их размер, не помечая их никаким атомом. Тем не менее это чревато попыткой применения операции над числами с плавающей точкой к, например, целочисленным данным. Такая попытка при использовании атомов будет пресечена на этапе проверки модели. Кроме того, тегирование данных атомами позволяет иметь более читаемый вид промежуточного представления: аналитик может, глядя на атом данных, получить представление о их природе.

Договоримся для дальнейшего изложения под *i8, i16, i32, i64* понимать соответственно 8-, 16-, 32- и 64-битные целочисленные атомы.

3.2. Операции

Как и при любом применении операционного подхода к описанию семантики, необходимо определять примитивные *операции*, через которые будет описываться поведение целевой машины.

В соответствии с требованием о простоте модели возникает естественное требование к отсутствию побочных эффектов у операций. К сожалению, подобный подход приведет к необозримым по объему моделям инструкций Intel 64 из-за обновления большого количества флагов слова состояния. Компромиссом здесь будет являться несколько облегченное требование к побочным эффектам операций: операция не должна иметь побочных эффектов, кроме, возможно, чтения и записи некоторых флагов модельного *слова состояния*.

Следует отметить, что при трансляции достаточно крупных регионов программы (например, суперблоков) с применением оптимизаций побочные эффекты, указанные явно, могут использоваться, так как весь излишний код будет исключен в результате оптимизации. Тем не менее, поскольку одной из требуемых возможностей является использование модели для работы системы в режиме отладки, от такого подхода приходится отказаться.

Модельное слово состояния представляет собой 16-битное значение, включающее флаги состояния Intel 64: AF, CF, OF, PF, SF, ZF. Как показывает практка, обновления этих флагов достаточно не только для моделирования Intel 64, но и RISC-машин, таких как MIPS64 или ARM.

Операции явно специфицируют *i-маску* и *o-маску*, то есть набор флагов слова состояния, которые данная операция просматривает и изменяет. Это позволяет проводить некоторые виды анализа, например, слайсинг, без необходимости различения поведения отдельных операций, лишь за счет точной спецификации зависимостей по данным, возникаемых в результате применения этих операций.

Объединяя все вышесказанное, приходим к следующим характеристикам операций: имя, возвращаемый атом, атомы параметров, i-маска и o-маска.

Укажем некоторые из стандартных операций модели.

1. Операция *add.i32* представляет собой целочисленное сложение 32-битных целых. Атом результата этой операции — *i32*, она имеет два параметра, каждому из которых также соответствует атом *i32*. I-маска этой операции пуста , а o-маска включает в себя флаги AF (BCD-переполнение), CF (переполнение), OF (знаковое переполнение), PF (четность), SF (знак), ZF (ноль).
2. Операция *smask.c.i16* получает в качестве параметра 16-битное целое значение (*i16*) и возвращает также *i16*. Ее поведение таково: все сброшенные биты параметра копируются в сброшенном виде в результат, а выставленные биты заменяются на значение флага CF из слова состояния. Таким образом, i-маска этой операции состоит из флага CF, а o-маска пуста.

3.3. Адресные пространства

Как было указано выше, для решения проблемы сложно организованого регистрового файла может быть применена модель адресных пространств. С точки зрения этой модели все методы адресации, с которыми может работать машина (регистры, память, порты ввода-вывода), сводятся к единой схеме: каждое адресумое данное представляется в виде ссылки на одно из адресных пространств и снабжается указателем в это адресное пространство.

Для пространств оперативной памяти такое отображение вводится естественным образом. Несложно расширить естествнное представление и на пространство портов ввода-вывода Intel 64.

Регистровый файл отображается в адресное пространство следующим образом. Выбирается размер адреса, позволяющий вместить все адресуемые регистры машины, после чего регистры размещаются в строящемся пространстве в произвольном порядке, но с учетом наложений и пересечений. Рис. 2 иллюстрирует эту идею для небольшой части регистрового файла Intel 64.

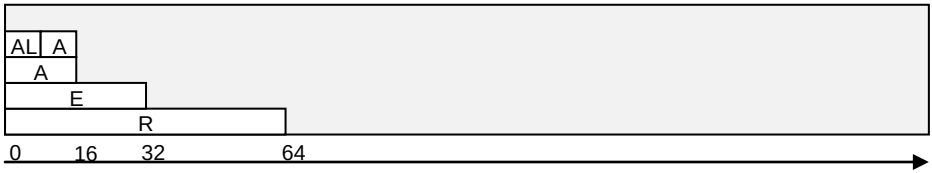


Рис. 2. Фрагмент адресного пространства регистров Intel 64.

При использовании такого подхода не только решается проблема сложной организации регистрового файла, но и вводится унифицированная модель адресуемых данных, что позволяет не рассматривать отдельно каждый из их видов.

Итак, в рамках предлагаемой модели адресные пространства машины включают в себя регистры, память, порты ввода-вывода и прочие подобные сущности и характеризуются атомом адреса (например, *i64* для 64-битной памяти). Для упрощения работы с адресными пространствами могут быть введены *псевдонимы*, т. е. имена для некоторых последовательных блоков адресов адресного пространства. Псевдонимы наиболее применимы для обозначения регистров, но могут использоваться и в других адресных пространствах (например, в качестве символов для закрепленных архитектурно адресов в памяти, в частности, адресов обработчиков исключений).

В целях приведения дальнейших примеров будем считать, что задано адресное пространство регистров, называемое *r*.

3.4. Локальные переменные

При построении операторов в описываемой модели могут использоваться локальные переменные. Время жизни локальных переменных ограничена единицей трансляции. Так, при трансляции отдельных инструкций локальные переменные локальны в рамках данной инструкции и используются для хранения результатов промежуточных вычислений. Локальные переменные характеризуются своими номерами. В целях упрощения дальнейшего анализа к локальным переменным предъявляются требования SSA-формы.

Принимая во внимание достаточно простой с точки зрения внутренних передач управления вид машинных инструкций, можно не вводить оператор φ в модель.

При определении локальной переменной с ней сопоставляется атом правой части соответствующего оператора.

Локальные переменные будем обозначать через *t.N*, где *N* — номер переменной.

3.5. Операторы модели

Модель предлагает следующие 8 операторов, в которые отображается операционная семантика транслируемых машинных инструкций.

1. Оператор NOP, не имеющий никакого эффекта.
2. Оператор INIT, инициализирующий локальную переменную константным значением. Константа задается в виде битовой последовательности и атома.

- Пример: INIT (i16) t.3, 0x40 => (i16) t.3 := 0x40.

3. Оператор APPLY, применяющий одну из модельных операций. В качестве параметров и результата используются локальные переменные. Может быть указана дополнительная о-маска, ограничивающая влияние на модельное слово состояния. Итоговая о-маска рассматривается как побитовое «или» дополнительной маски и маски применяемой операции.

- Пример: APPLY (i32) t.2, xor.i32(t.0, t.1) => (i32) t.2 := xor.i32(t.0, t.1).
- Пример: APPLY (i16) t.22, smask.c.i16(t.4) => (i16) t.22 := smask.c.i16(t.4).

4. Оператор BRANCH, осуществляющий передачу управления. Различаются три вида передачи управления: безусловная, условная по маске и условная по сложному предикату. Во всех случаях указывается знаковое смещение для передачи управления (в качестве единицы адреса используется один модельный оператор). При использовании условной передачи управления по маске указываются два 16-битных числа: *rmask* и *lmask*. Передача осуществляется тогда и только тогда, когда все выставленные биты в *rmask* выставлены и в модельном слове состояния и все выставленные биты в *lmask* не выставлены в модельном слове состояния. Наконец, условная передача управления по сложному предикату проверяет один из предикатов A (беззнаковое «больше»), AE (беззнаковое «больше или равно»), B (беззнаковое «меньше»), BE (беззнаковое «меньше или равно»), G (знаковое «больше»), GE (знаковое «больше или равно»), L (знаковое «меньше»), LE (знаковое «меньше или равно»).

- Пример: BRANCH.BE +6.
- Пример: BRANCH.MASKED +13, 0x0010, 0x0000.
- Пример: BRANCH +7.

5. Оператор LOAD, производящий загрузку значения из одного из адресных пространств (например, значения регистра). Для оператора LOAD указывается адресное пространство, из которого осуществляется загрузка, атом загружаемого данного, локальная переменная с адресом данного и локальная переменная, принимающая результат. При этом атом локальной переменной адреса должен совпадать с атомом адреса пространства.

- Пример: LOAD (i16) t.29, r:t.2.

6. Оператор STORE, записывающий значение в одно из адресных пространств. Для оператора STORE указывается адресное пространство для записи и локальные переменные адреса и значения. Атом локальной переменной адреса должен совпадать с атомом адреса пространства.

- Пример: STORE (i8) r:t.0, t.21.

7. Семейство операторов SPECIAL, указывающих на особое моделируемое состояние. Сюда относится оператор HALT (останов до возникновения внешнего события) и оператор TRAP (исключение или прерывание).

- Пример: HALT.
- Пример: TRAP 0.

8. Операторы ANNOTATION. Эти операторы игнорируются при интерпретации отдельной инструкции и не влияют на модельную семантику. Они используются для задания дополнительных аннотаций, которые могут быть использованы в отдельных видах анализа.

3.6. Пример

В качестве примера рассмотрим инструкцию ADD машины Intel 64, осуществляющую сложение значения регистра с константой и запись результата в тот же регистр.

Конкретный вид рассматриваемой инструкции[1]: ADD EAX, 1.

Модель для этой инструкции приведена далее.

```
1.  INIT    (i16)   t.-1, 0x0000             ; адрес регистра EAX в r
2.  INIT    (i8)    t.-2, 0x01               ; значение операнда-константы
3.  LOAD    (i32)   t.0, r:t.-1              ; загрузка значения EAX
4.  APPLY   (i32)   t.1, sx.i8.i32(t.-2)     ; знаковое расширение[2] конс-
    танты до i32
5.  APPLY   (i32)   t.2, add.i32(t.0, t.1)   ; сложение
6.  STORE   (i32)   r:t.-1, t.2              ; сохранение результата в EAX
7.  INIT    (i16)   t.3, 0x40                ; адрес регистра EFLAGS в r
8.  LOAD    (i16)   t.4, r:t.3               ; загрузка значения EFLAGS
9.  INIT    (i16)   t.5, 0x08D5              ; подготовка маски для
    EFLAGS
10. APPLY   (i16)   t.6, uf(t.4, t.5)        ; обновление EFLAGS с учетом
    маски[3]
```

[1] Константа 1 в этой инструкции кодируется как 8-битное число со знаком.

[2] Операция знакового расширения 8-битного числа до 32-битного.

```
11. STORE  (i16)   r:t.3, t.6                ; сохранение EFLAGS
```

Можно видеть, что большая часть модели представляет собой обновление слова состояния Intel 64. В RISC-архитектурах трансляция аналогичной инструкции сложения выглядит значительно компактнее: инструкция ADDIU машины MIPS64 при своем применении в виде ADDIU R1, R1, 1 транслируется в следующую модель.

```
1. INIT   (i16)   t.-1, 0x0010               ; адрес регистра R1 в r, первый операнд
2. INIT   (i16)   t.-2, 0x0010               ; адрес регистра R1 в r, второй операнд
3. INIT   (i16)   t.-3, 0x0001               ; значение операнда-константы
4. LOAD   (i64)   t.0, r:t.-2                ; загрузка значения R1
5. APPLY  (i32)   t.1, trunc.i64.i32(t.0)    ; ограничение значения 32 битами
6. APPLY  (i32)   t.2, sx.i16.i32(t.-3)      ; знаковое расшиние операнда-константы
7. APPLY  (i32)   t.3, add.i32(t.1, t.2)     ; сложение R1 и константы
8. APPLY  (i32)   t.4, sx.i32.i64(t.3)       ;расширение до размера регистра
9. STORE  (i64)   r:t.-1, t.4                ; сохранение результата в R1
```

4. Спецификация целевой машины

Для обеспечения построения моделей конкретных инструкций целевой машины необходимо предоставить ее спецификацию.

Спецификация включает в себя описание атомов, операций, адресных пространств и подстановок (см. далее), используемых в данной машине. Поскольку подобное примеру из раздела 4.6 низкоуровневое описание модели неудобно с точки зрения поддержки, был разработан язык спецификаций, позволяющий записывать некоторые распространенные идиомы более компактно.

Процесс работы со спецификациями устроен следующим образом.

1. Разработчик предоставляет спецификацию машины в виде текстового файла на разработанном в рамках данной работы языке Pivot.
2. Спецификация проверяется на корректность и строится ее низкоуровневый вид в виде метамодели, позволяющей впоследствие генерировать модели семантики конкретных инструкций. Эта операция выполняется инструментом PivotC.

---

[3] Операция обновления слова состояния из модельного слова состояния с учетом параметра-маски.

3. Результат работы PivotC — бинарный файл, называемый *архивом машины*. Для чтения этого архива разработана библиотека PivotPcma, использующаяся в дальнейших анализах и при интерпретации.

4.1. Структура спецификации

Спецификация представляет собой набор текстовых файлов на языке Pivot. Краткое описание этого языка дается в последующих подразделах.

Каждый файл представляет собой последовательность деклараций, описывающих один из аспектов специфицируемой машины. Сюда относятся декларации атомов, операций, адресных пространств и подстановок. Кроме того, файлы спецификаций могут включать (include) друг друга, тем самым позволяя использовать общие для нескольких машин декларации без дублирования их текста.

4.2. Атомы, операции и адресные пространства

Для спецификации атомов, операций и адресных пространств введен простой синтаксис, иллюстрируемый следующими примерами (все примеры взяты из спецификации машины 8086).

- **atom** i8 /.8
  Описывает атом *i8* размером 8 бит (число до точки — байты, число после точки — биты, любое из них может быть опущено).
- **operation** i16 cons.i16(i8, i8)
  Описывает операцию *cons.i8* с пустыми i- и o-масками.
- **operation** # and.#(#, #) **omask** “COPSZ” **with** i8, i16, i32, i64
  Описывает семейство операций *and.i8*, *and.i16*, *and.i32*, *and.i64* с пустой i-маской и o-маской, включающей CF, OF, PF, SF и ZF. Символ «#» — подстановочный, вместо него подставляются все атомы из блока **with**.
- **space** io(i16)
  begin
  end
  Описывает адресное пространство с атомом адреса *i16*.
- **space** r(i16)
  begin
  alias ax 0 /.16
  alias cx +8 /.16
  …
  alias ah ax + .8 /.8
  end
  Описывает адресное пространство с атомом адреса *i16* и задает в нем псевдонимы регистров. Адрес псевдонима может быть указан явно

(ax), относительно предыдущего псевдонима (cx) или относительно одного из ранее определенных псевдонимов (ah).

4.3. Подстановки

Для сопоставления моделей операционной семантики с машинными инструкциями применяются *подстановки*. С одной стороны, подстановка указывает, каким инструкциям целевой машины она отвечает, то есть какие модели каких инструкций могут быть построены при помощи этой подстановки. С другой стороны, подстановка специфицирует сами эти модели. Одна подстановка может описывать семейство сходных инструкций за счет достаточно мощного синтаксиса описания с возможностью применения шаблонов и автоматического вывода атомов.

Подстановка состоит из двух частей: заголовка и тела.

Заголовок подстановки включает в себя сопоставляемую мнемонику (как строку) и условия на операнды машинной инструкции.

Изучение современных процессорных архитектур показывает, что операнды инструкций можно разбить на четыре класса:

1. константы, т. е. прямо закодированные значения;
2. простые выражения над регистрами, например, значение регистра со сдвигом в ARM (например, R1 << 4);
3. регистры и прямо закодированные адреса в памяти;
4. вычисляемые адреса в памяти.

Первый и второй классы, таким образом, объединяют в себе операнды, используемые только по значению. Такие операнды характеризуются значением (возможно, вычисляемым) и типом данных. Это означает, что с точки зрения вводимой модели можно рассматривать такие операнды как значения, записанные в некоторые локальные переменные. Тип данных при этом определяется атомом этой переменной.

Третий и четвертый классы с точки зрения модели могут быть представлены как ссылки в адресные пространства. При этом в одной из локальных переменных будет храниться адрес. Как и в случае с константами и выражениями над регистрами, тип адресуемых данных указывается в виде атома.

Условия на операнды в подстановках выписываются в соответствии с указанной классификацией. Операнд помечается как *val*, если он принадлежит к классу 1 или 2, и как *ref*, если он принадлежит к классу 3 или 4. Кроме того, для каждого операнда сообщается его атом.

Например, рассмотренная выше машинная инструкция ADD EAX, 1 удовлетворяет такому заголовку подстановки:

match “ADD” **ref** i32, **val** i8.

При поиске подстановки для конкретной инструкции выбирается та, мнемоника которой совпадает с мнемоникой инструкции и условия на операнды которой выполнены для этой инструкции.

Тело подстановки представляет собой последовательность операторов языка спецификации, которые отображаются на операторы модели (NOP, INIT, APPLY, BRANCH, LOAD, STORE, SPECIAL, ANNOTATION). Перед любым из операторов может быть поставлена метка (синтаксис **label** *метка*:). Определены следующие операторы.

1. Оператор **nop**, отображаемый в NOP.
2. Оператор присваивания общего вида *lvalue* = *rvalue*. Части представляют собой одно из следующих выражений:
    a. Ссылка на операнд $*N*, например $1. Если *N*-й операнд помечен как *const*, то выражение раскрывается в значение операнда (то есть в соответствующую локальную переменную). Если *N*-й операнд помечен как *pointer*, то выражение раскрывается в чтение значения операнда (то есть в LOAD в правой части и STORE в левой части). Может встречаться только в правой части для *const*-операндов и в обеих частях для *pointer*-операндов.
    b. Ссылка на псевдоним *пространство*:*псевдоним*, например *r*:*eax*. В левой части раскрывается в соответствующий STORE, а в правой — в LOAD.
    c. Константа (*атом*) *значение*, например (*i16*) 0x08D5. Может встречаться только в правой части.
    d. Применение операции *операция*(*параметры…*) { **omask** *маска* }[4], например *add*.*i32*($1, $2). Раскрывается в оператор APPLY. В качестве *параметров* могут использовать произвольные выражения из данного списка. Применение операции возможно только в правой части. Дополнительно может быть указана *маска*, ограничивающая изменение модельного слова состояния.
    e. Локальная переменная **local** *имя*. Может встречаться в каждой из частей, но требуется единственность определения.
    f. Обращение к адресному пространству *пространство*[*адрес*], где *адрес* — произвольное выражение. Требуется, чтобы атом выражения *адрес* совпадал с атомом адреса пространства.
3. Вычисление побочного эффекта выражения **discard** *rvalue*. Оператор вычисляет правую часть, но не сохраняет результат, лишь обновляя слово состояния.

[4] Фигурные скобки обозначают необязательную часть.

4. Операторы **branch** *метка*, **branch.***условие метка* и **branch**(*rmask*, *lmask*) *метка*. Раскрываются в соответствующие операторы BRANCH.
5. Операторы **halt** и **trap** *номер*, отображаемые в соответствующие SPECIAL-операторы.

Обращения к операндам выполняются следующим образом: операнду с номером *N* сопоставляется локальная переменная *t.-N*, в которой при построении модели конкретной инструкции для *val*-операндов содержится значение операнда, а для *ref*-операндов — значение адреса. Эти локальные переменные видны на уровне модели, но скрыты от разрабочика спецификаций за счет выражений $*N*.

Транслятор PivotC содержит подсистему вывода атомов, что позволяет указывать атомы лишь при константах и некоторых чтениях из адресных пространств.

Кроме того, транслятор позволяет применять шаблоны при описании подстановок: как и в случае с операциями в качестве атома операнда можно указать символ «#». В этом случае вместо него будут последовательно подставлены все атомы, перечисленные в блоке **with**. Пример заголовка такой подстановки:

match “IN” **ref** #, **val** i8 **with** i8, i16.

Символ «#» может быть также указан и при применении операции в операторах тела подстановки. В этом случае PivotC выберет ту операцию соответствующего семейства, которая подходит по сигнатуре.

В качестве еще одного упрощения помимо классов *val* и *ref* для операторов вводится псевдо-класс *auto*, который подставляется и как *val*, и как *ref*. При этом если подстановка в качестве *val* невозможна (из-за наличия ссылки на операнд в левой части оператора присваивания), будет подставлен только *ref*-вариант.

Описанный язык позволяет достаточно быстро составлять спецификации машинных инструкций. Встроенные средства вывода атомов избавляют от необходимости указывать их вручную. Возможность шаблонного задания подстановок значительно облегчает их спецификацию для CISC-архитектур.

4.4. Примеры

В качестве первого примера рассмотрим спецификацию подстановки для инструкции NOR машины MIPS64, выполняющую операцию «не-или» над значениями второго и третьего операндов-регистров и сохраняющую результат в первый операнд-регистр.

match “NOR” **ref** i64, **ref** i64, **ref** i64
begin

```
        $1  = not.i64(or.i64($2, $3))[5]
end
```

Легко видеть, что построенная спецификация достаточно проста и практически «слово в слово» повторяет описание поведения инструкции из документации по процессору MIPS64. Такая ситуация характерна для RISC-архитектур в целом.

В качестве примера набора подстановок для CISC-инструкции рассмотрим инструкцию сложения ADD машины Intel 64. Документация на Intel 64 указывает 19 различных кодировок этой инструкции. Использование шаблонов и ключевого слова *auto* позволяет свести эти 19 случаев к трем подстановкам.

```
match "ADD" ref #, auto # with i8, i16, i32, i64
begin
        $1 = add.#($1, $2);  r:flags = uf(r:flags, (i16) 0x08D5)
end
match "ADD" ref #, val i8 with i16, i32, i64
begin
        $1 = add.#($1, sx.i8.#($2)); r:flags = uf(r:flags, (i16) 0x08D5)
end
match "ADD" ref i64, val i32
begin
        $1 = add.i64($1, sx.i32.i64($2)); r:flags = uf(r:flags, (i16) 0x08D5)
end
```

В результате подстановки атомов из блоков **with** и за счет использования ключевого слова **auto** этими тремя подстановками будут покрыты все 19 случаев кодировки.

Вторая подстановка для ADD при замене «#» на *i32* будет раскрыта транслятором PivotC в модель, приведенную в 3.6, за исключением первых двух операторов. Эти операторы будут генерироваться отдельно для каждого конкретного набора операндов.

5. Интерпретация модели

В рамках данной работы реализована прототипная система PivotI интерпретации модели операционной семантики инструкции. Поставленная перед системой интерпретации задача может быть формализована следующим образом.

Пусть задано полное состояние машины в некоторой точке времени (под полным состоянием машины понимаются значения всех ее регистров, в том

[5] Здесь и далее в примерах подстановок с целью упрощения читаемости опускается часть, связанная с продвижением указателя на текущую инструкцию (PC, EIP и т. п.).

числе регистра счетчика инструкций, а также содержимое памяти и иных пространств). Требуется получить полное состояние машины после выполнения *одной* инструкции по указателю счетчика инструкций (т. е. текущей инструкции).

Задачу будем решать в несколько этапов.

1. Необходимо провести декодирование текущей инструкции. Тем самым будет получена мнемоника этой инструкции (в виде строки), а также список ее операндов.
2. Среди подстановок для данной машины требуется найти ту, которая принимает данную инструкцию. При этом необходимо отобразить машинные типы на стандартные атомы (атомов *i8, i16, i32, i64* достаточно для подавляющего большинства пользовательских инструкций, за исключением FPU- и SIMD-инструкций, для которых определяются свои атомы, такие как *f64* и *v128*).
3. Сгенерировать код инициализации значений операндов-констант и адресов операндов-указателей (локальные переменные *t.-N*). В простых случаях этот код будет представлять собой набор операторов INIT; в более сложных (например, при использовании регистровых окон или косвенной адресации памяти) будет включать в себя соответствующую адресную арифметику.
4. Дополнить сгенерированный код операторами из подстановки, тем самым получив полную модель данной инструкции.
5. Выполнить модель.

Поскольку из перечисленных этапов лишь последний является неочевидным, рассмотрим его более подробно.

Обозначим сначала требования, которые интерпретатор будет предъявлять к знаниям о машине. Во-первых, для интерпретации необходимо знать смысл каждой из используемых в машине операций. Во-вторых, необходимо понимать природу используемых адресных пространств (запись в память и запись в порт ввода-вывода принципиально отличны с точки зрения влияния на состояние машины).

Первое из требований приводит к классическому замкнутому кругу при использовании операционного подхода к описанию семантики, так как фактически требует задания операционной семантики тех операций, которые, в свою очередь, используются для описания семантики машинных инструкций. Эту проблему можно разрешить, отобразив эти операции на набор инструкций машины, на которой выполняется сам интерпретатор, то есть синтезировав аналогичный native-код. Фактически, таким образом, интерпретатор производит бинарную трансляцию отдельных инструкций.

На данный момент прототип интерпретатора может использоваться на машине Intel 64; требуется доступность расширенных 64-битных регистров и машинных команд.

Второе требование предусматривает установку обработчиков чтения и записи на каждое из адресных пространств. Такие обработчики для некоторых адресных пространств будут тривиальными (например, для адресного пространства регистров общего назначения), а для некоторых (например, для пространства портов ввода-вывода) потребуют эмуляции набора оборудования машины. Впрочем, для интерпретации пользовательского кода эмуляция оборудования, как правило, не требуется.

Выполнение модели производится в рамках *контекста интерпретации*. Контекст интерпретации включает в себя следующие части.

1. Область локальных переменных. В этой области хранятся значения локальных переменных и их атомы.
2. Информация об операциях в виде списка соответствующих им встраиваемых функций.
3. Информация об адресных пространствах в виде списка соответствующих им обработчиков.
4. Область сгенерированного кода.
5. Информация о распределении локальных переменных по регистрам машины, на которой производится интерпретация.

Для решения подзадачи отображения операций модели на машинный код инструментального компьютера используются встраиваемые функции. К ним предъявляется перечень требований, закрепляющих, в частности, отсутствие пролога и эпилога, запрет использования стека и соглашение о распределении машинных регистров, через которые функция принимает значения параметров интерпретируемой операции и возвращает результат. Поскольку особенности реализации интерпретатора не являются основной темой данной работы, этот перечень требований опускается. Приведем лишь пример одной такой функции для операции сложения с обновлением слова состояния *add.i32*.

```
arithmetic_add_i32 PROC
        ;; Сложение.
        MOV     R15D, R8D
        ADD     R15D, R9D

        ;; Пересчет слова состояния.
        LAHF
        SETO    AL
        SHL     AL, 3
        AND     BH, 0FCh
        OR      BH, AL
        MOV     BL, AH
arithmetic_add_i32 ENDP
```

Особенностью такого подхода является возможность простого расширения набора поддерживаемых операций за счет расширения множества функций. Это, в свою очередь, позволяет определять операции, упрощающие интерпретацию для конкретной машины.

Важным моментом является то, что функции не вызываются, а подставляются в генерируемый блок кода полностью, что позволяет избежать накладных расходов на передачу параметров и сброс конвейера.

Интерпретатор включает в себя набор встраиваемых функций для «стандартных» операций (целочисленная арифметика, битовая логика и т. д.).

Похожим образом решается и вторая подзадача. Соответствующие блоки ассемблерного кода описывают процедуры чтения из адресных пространств и записи в адресные пространства. В интерпретатор включены так называемые «нулевой механизм» и «механизм по умолчанию».

«Нулевой механизм» используется при отладке. Он игнорирует все записи в адресное пространство, к которому привязано, а при чтениях возвращает нули.

«Механизм по умолчанию» выделяет блок памяти размером со все адресное пространство и реализует чтения и записи естественным образом. Механизм применим для регистровых адресных пространств, но неприменим для пространств памяти из-за их относительно большого объема. Для реализации механизма работы с моделируемой оперативной памятью требуется более сложная логика, в частности из-за необходимости трансляции виртуальных адресов в физические.

Генерация машинного кода для интерпретируемой инструкции происходит в два этапа: этап генерации кода и этап расстановки переходов.

Первый этап последовательно просматривает операторы модели и генерирует машинный код по следующим правилам.

1. NOP: подставляется инструкция NOP.
2. INIT: подставляется инструкция MOV (или XOR для инициализации нулем).
3. MOVE: подставляется соответствующая функция.
4. BRANCH: подставляется инструкция перехода по данному условию, однако адрес перехода не заполняется, т. к. еще неизвестен.
5. LOAD: подставляется обработчик-механизм загрузки из соответствующего адресного пространства.
6. STORE: подставляется обработчик-механизм выгрузки в соответствующее адресное пространство.
7. SPECIAL: подставляется код, выставляющий специальное состояние интерпретатора.
8. ANNOTATION: игнорируются.

На втором этапе заполняются адреса всех переходов, сгенерированных в результате синтеза кода для операторов BRANCH.

При генерации кода используется область локальных переменных, где хранятся значения и атомы, связанные с ними. Производится несложное распределение регистров, сокращающее обращения к памяти этой области. Такой подход в совокупности с минимальным количеством инструкций передачи управления в сгенерированном коде позволяет максимально использовать специфику Intel 64 (out-of-order issue и неявный параллелизм).

Сгенерированный код снабжается прологом и эпилогом и может быть выполнен для моделирования эффекта инструкции, что решает поставленную задачу. Особенностью является возможность одновременного существования произвольного количества контекстов интерпретации, содержащих трансляции отдельных инструкций, что позволяет, например, реализовывать пошаговую отладку многопоточных приложений.

К преимуществам применяемого в прототипной системе подхода следует отнести его простоту и гибкость. Очевидный недостаток — неэффективный генерируемый код. Эта проблема может быть решена при использовании вместо такой «наивной» генерации кода одного из алгоритмов динамического программирования для генерации оптимального кода (tree rewriting) в сочетании с набором оптимизаций. Поддержка такого способа интерпретации планируется в дальнейшем.

Тем не менее, отказываться от предложенной схемы полностью также неразумно, т. к. в случаях, когда инструкция выполняется лишь один раз данная схема будет показывать большую производительность, чем полноценный синтез за счет отсутствия расходов на анализ и оптимальную кодогенерацию. Окончательным вариантом интерпретации видится сочетание двух подходов (например, применение полновесного синтеза только для расширенных базовых блоков, лежащих на горячих путях).

6. Заключение

В настоящей работе выделены требования к модели операционной семантики машинных инструкций, выполнение которых необходимо для проведения широкого класса анализов, в совокупности нацеленных на восстановление алгоритмов. Предложена удовлетворяющая этим требованиям модель, опробованная в контексте широко распространенных CISC- и RISC-архитектур. Указаны способы упрощения построения спецификаций машин, описывающих приведение семантики инструкций к модельному виду. Эти способы нашли свою реализацию в средстве PivotC и библиотеке PivotPcma. С использованием указанной библиотеки разработан и опробован прототип системы интерпретации инструкций PivotI, общая схема функционирования которого также описана в работе.

В дальнейшем планируется полностью специфицировать распространенные машины (Intel 64, MIPS64, ARM, PowerPC) и на основании этих спецификаций сделать выводы о достаточном наборе модельных операций.

Планируется адаптировать алгоритм слайсинга трассы для использования с моделями операционной семантики инструкций трассы, что позволит отбирать в слайс не только машинные инструкции целиком, но и их части, выраженные в виде операторов модели.

Улучшение интерпретации за счет усложнения схемы кодогенерации также является важной задачей. Кроме того, планируется провести исследование возможности введения «границ интерпретации» на входе и выходе из известных библиотечных функций путем замены самих этих функций спецификациями влияния на состояние машины. Это позволит значительно ускорить интерпретацию и во многих случаях отказаться от необходимости эмуляции оборудования.

Введение данной модели позволяет вплотную подойти к решению важной и интересной задачи интерпретации нереализованных путей в трассе, которая может играть решающую роль при динамическом анализе. Неформально задача может быть поставлена следующим образом. Пусть дан набор трасс некоторой программы и известно, каким образом эти трассы связаны друг с другом (например, они снимались с одного и того же начального состояния, но с различным пользовательским вводом). Пусть в программе существует некоторое ветвление, одна из ветвей которых не реализовалась ни в одной из трасс. Выдвигается предположение о реализации этой ветви. Система, решающая задачу, должна предоставить условия на состояние машины для реализации ветви (в идеале эти условия должны быть распространены до пользовательского ввода), выбрать конкретный вид переменных, входящих в условие и в данных условиях провести интерпретацию ветви, тем самым получив дополнительную трассу выполнения, в которой интересующий путь уже будет реализован.

Кроме того, в список дальнейших работ включено также исследование возможности повышения уровня представления семантики, в частности, трансляция в LLVM [12].

Литература

[1] Christian Collberg, Clark Thromborson, Douglas Low. A taxonomy of obfuscating transformations. – Technical report, University of Auckland, New Zealand.

[2] Chenxi Wang, Jonathan Hill, John Knight, Jack Davidson. Software tamper resistance: obstructing static analysis of programs. – Technical report, University of Virginia, US. – 2000.

[3] В. А. Падарян, А. И. Гетьман, М. А. Соловьев. Программная среда для динамического анализа бинарного кода. // Труды Института системного программирования, том 16, с. 51-72. – 2009.

[4] B. Korel, J. Laski. Dynamic program slicing. // Information Processing Letters, vol. 29, issue 3, pp. 155-163. – 1988.

[5] B. Korel, R. Smith. Slicing event traces of large software systems. // Automated and algorithmic debugging. – 2000.
[6] Steven Muchnick. Advanced compiler design and implementation. – Morgan Kaufmann Publishers Inc. – 1997.
[7] GNU Compiler Collection. http://gcc.gnu.org/.
[8] Nicholas Nethercote, Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. // Proceedings of ACM SIGPLAN 2007 conference on programming language design and implementation. – San Diego, California, US. – 2007.
[9] Sanjay Bhansali, Wen-Ke Chen, Sturart de Jong, Andrew Edwards, Ron Murray, Milenko Drinic, Darek Mihocka, Joe Chau. Framework for Instruction-Level Tracing and Analysis of Program Executions. – Microsoft Corporation. – 2006.
[10] Cristina Cifuentes, M. Van Emmerik, Norman Ramsey, Brian Lewis. Experience in the design, implementation and use of a retargetable static binary translation framework. – 2002.
[11] Cristina Cifuentes, Brian Lewis, David Ung. Walkabout – a retargetable dynamic binary translation framework. – 2002.
[12] Low Level Virtual Machine. http://www.llvm.org/.

Энергосберегающая оптимизация кода за счет использования отключаемых компонентов процессора

И.И. Каретин, В.А. Макаров
IlyaK@astrosoft.ru, Vladimir.Makarov@novsu.ru

Аннотация. Статья посвящена рассмотрению вопроса снижения энергопотребления в процессе исполнения кода программ. Объектом исследования является техника отключения отдельных компонентов процессора. Описываются теоретические аспекты модификации исходного кода с целью повышения энергетического эффекта от отключения компонентов.

Цель оптимизирующего компилятора заключается в получении эффективного кода. В качестве критерия его эффективности может быть выбрано быстродействие, занимаемый объем или энергопотребление. В данной статье будут рассматриваться оптимизации по критерию энергопотребления кода.

Под энергопотреблением кода будем понимать то количество энергии, которое будет потрачено на целевой платформе при его выполнении. Как следует из данного определения, сравнение алгоритмов по эффективности по выбранному критерию возможно только с указанием платформы, для которой производится исследование. Будем говорить, что код более энергоэффективен, если его энергопотребление ниже.

Значение энергосберегающих компиляторов с течением времени только возрастает, так как аппаратные методики экономии энергии достаточно дороги и, как правило, требуют значительного перепроектирования архитектуры. В это же время область генерации энергоэффективного кода остается изученной явно недостаточно, хотя ее углубленная разработка способна оказать значительный эффект на время работы устройств, размеры их элементов питания, тепловыделение и прочие характеристики, связанные с показателями энергопотребления.

Данная работа посвящена исследованию возможности снижения энергопотребления кода за счет применения техники отключения некоторых компонентов процессора. Основной идеей является использование модификации исходного кода для повышения энергетического эффекта от отключения компонентов. Подробное описание такой целевой платформы

можно увидеть в работе [1]. Далее оно будет приведено только в объеме, необходимом для понимания предлагаемой методики.

Представим центральный процессор как систему, включающую в себя компоненты, которые могут быть включены или отключены в заранее определенные моменты времени. Для подключения и отключения этих компонентов используется специальный набор команд.

В данной архитектуре у нас есть 5 устройств, которые могут подключаться и отключаться в произвольные заранее заданные моменты. Это целочисленное АЛУ, целочисленный умножитель, вещественный сумматор, вещественный умножитель и вещественный делитель.

Предлагаемый метод оптимизации включает в себя несколько последовательных этапов. Сначала производится анализ потока управления программы и на его основе в коде программы устанавливаются потенциальные места для добавления специальных инструкций, отключающих либо активирующих отдельные компоненты процессора. Например, если в ходе вычислений нам продолжительное время не требуется модуль целочисленного умножения, то на определенном участке кода он может быть отключен для экономии энергии. При этом следует правильно оценивать время простоя этого модуля: нужно добиться того, чтобы накладные расходы на его отключение и последующее включение не превысили бы экономию за период отсутствия активности.

В данной работе будет рассматриваться метод оптимизации компонентов по отдельности. Взаимные влияния компонентов не анализируются, это будет темой отдельного исследования.

Обозначим рассматриваемый компонент символом C. Все дальнейшие рассуждения будут касаться только его. Затем данный метод должен быть применен для каждого из оставшихся компонентов.

При оценке целесообразности отключения компонента системы следует пользоваться такой формулой:

$$E_{\text{turn off}}(C) + E_{\text{turn on}}(C) \le \sum_{i=1}^{N} P_i(C),$$

Где: $E_{\text{turn off}}(C)$ – затраты энергии на отключение компонента *C*;

$E_{\text{turn on}}(C)$ – затраты энергии на активизацию компонента *C*;

N – количество инструкций в блоке, для которого компонент может быть отключен;

$P_i(C)$ – энергия, которая может быть сэкономлена за один цикл бездействия компонента *C* при выполнении инструкции *i*;

Следует отметить, что в отличие от формул, приведенных в работе [1], тут учитывается то, что энергетический эффект инструкции не является постоянной величиной и может отличаться при выполнении различных инструкций программы.

В том случае, если указанное неравенство выполняется, т.е. накладные расходы не превышают экономию энергии, то рассматриваемый участок кода специальным образом помечается, что означает, что далее в него нужно будет добавить инструкции для временного отключения компонента C. В противном случае код должен быть оставлен без изменений. При этом даже если условие не выполнилось, следует учитывать, что существует вероятность того, что к коду могут быть применены дополнительные оптимизации, в результате чего отключение компонента C все-таки может стать оправданным. Исследование условий, при которых возможно получение дополнительных участков, где выполняется указанное условие, будет темой отдельной работы.

В результате некоторые участки кода получили метки с указанием, какой из компонентов не требуется каждому из них.

Далее выполняется непосредственная оптимизация кода на основании собранной информации. Ниже будет рассмотрено несколько вариантов возможных оптимизаций, связанных с ветвлением или наличием циклов в программе. Все представленные ниже оптимизации объединяет то, что в результате преобразования кода не изменяется относительный порядок следования операторов. Изменение такого порядка потребовало бы дополнительного анализа зависимостей в программе, и потому оно будет темой отдельного исследования. Таким образом, в данной работе не будет рассматриваться оптимизации линейных участков программы, оптимизироваться будут только ветвления и циклы.

Итак, рассмотрим первый из возможных случаев применения оптимизации. Пусть перед ветвлением программы у нас есть n инструкций, в которых не требуется работа компонента C. Следует сразу определиться с условными обозначениями. На приведенных ниже схемах «N» обозначает блок из n инструкций, которому не требуется компонента C для работы. Аналогично «K_i» обозначает максимальный блок из k_i начальных инструкций в i-й ветви программы, которым не требуется компонент C. «C=0» обозначает, что компонент C в данном участке программы не требуется, а «C=1» обозначает границу блока инструкций, которым требуется компонент C. D_i обозначает возможную точку отключения (Disable), а E_i – возможную точку включения (Enable) компонента C соответственно. Следует отметить, что значения n и k_i лежат в диапазоне [0; s), где s – это общее количество инструкций программы. Другими словами, любой из этих блоков может быть пустым. Общее количество ветвей обозначим f. Таким образом: i Є [1, f]. Эти же обозначения будут использоваться и при рассмотрении всех последующих примеров.

На следующем рисунке показан общий случай ветвления в программе.

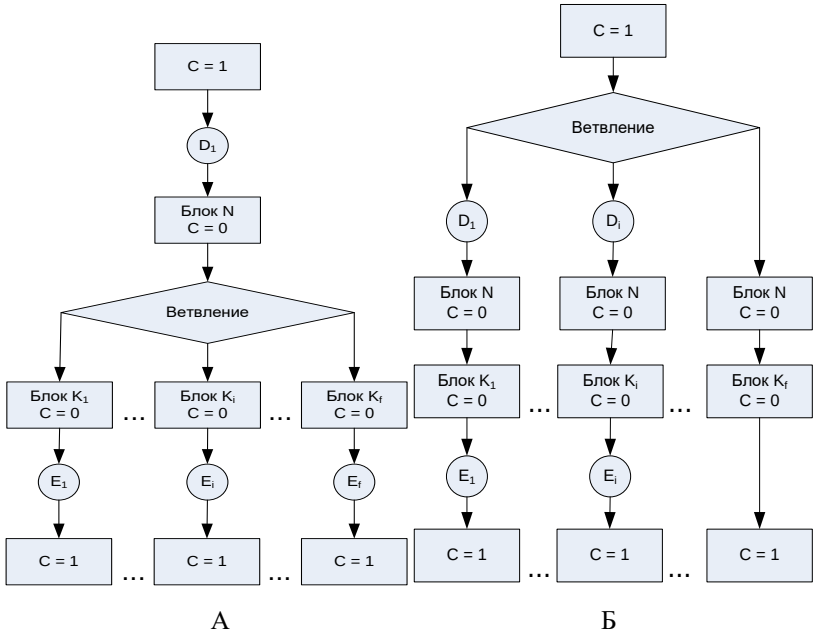


Рис. 1. Ветвление программы до (А) и после (Б) оптимизации.

После проверки условий ветвления, выполнение программы пойдет по одному из f различных путей. В данном случае из всего кода нас интересует только последний блок инструкций перед ветвлением (блок N) и первый блок инструкций в каждой из ветвей (блоки K_i). Линейные участки кода до ветвления и внутри ветвей не рассматриваются по описанным выше причинам.

Обозначим P(N) количество энергии, которое может быть сэкономлено в случае, если компонент C будет отключен при выполнении блока N. Аналогично $P(K_i)$ будет обозначать экономию энергии в первом блоке i-й ветви. Обозначим P_{min} то количество энергии, которое требуется на отключение и последующее включение компонента C.

В том случае, если выполняется условие $P(N) + \min_{i=1}^{f} (P(K_i)) \geq P_{min}$, то при движении по любой из ветвей мы получаем некоторый выигрыш от отключения компонента в точке D_1 и подключения его во всех точках E_i непосредственно перед инструкциями, обозначенными как C=1. В худшем случае, в некоторых ветвях экономия равна затратам на отключение и подключение компонента, но при этом существуют ветви, где энергия действительно экономится. Тогда никакого дополнительного изменения кода

не требуется и инструкции отключения и подключения компонента C могут быть добавлены в код в отмеченных местах.

В противном случае, если условие не выполняется, то отключение C для всех ветвей неэффективно. Тогда требуется перенос общих инструкций из блока N, не требующих компонента C, в каждую из ветвей для того, чтобы можно было индивидуально принимать решение о том, требуется ли отключать компонент C в данной ветви. В таком случае код должен быть преобразован к виду, показанному на рисунке 1-Б. Общий участок кода N дублируется в каждой из ветвей.

Как можно заметить, для упрощения анализа в данном случае опущены эффекты от операций проверок и переходов, обозначенных на схеме как «ветвление».

Аналогично следует рассмотреть и случай окончания ветвления, показанный на следующем рисунке 2-А.

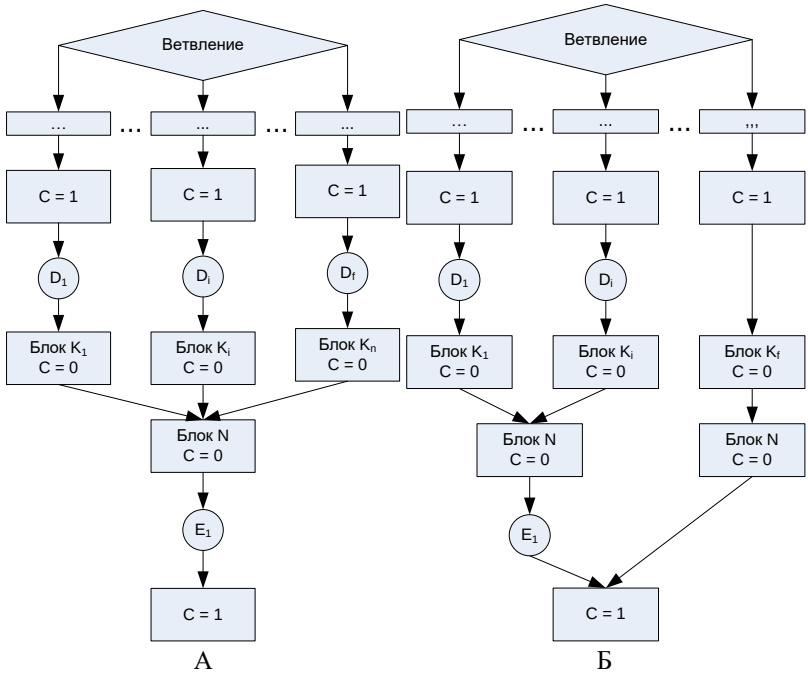


Рис. 2. Участок окончания ветвления программы до (А) и после (Б) оптимизации.

Аналогично предыдущему случаю, если согласно формуле (1) отключение компонента C будет эффективным во всех ветвях непосредственно перед

точкой объединения, то структуру программы можно оставить без изменений. Формула оценки целесообразности отключения компонента тут остается такой же, как и в прошлом случае: $P(N) + \min_{i=1}^{f} (P(K_i)) \geq P_{min}$.

Если хотя бы в одной из вервей не эффективно отключение компонента C, то необходимо производить дополнительный перенос инструкций. В результате данный участок программы будет приведен к виду, показанному на Рис. 2-Б.

Как и в прошлый раз, все ветви разделяются на 2 класса – те, в которых производить отключение компонента C эффективно и те, в которых оно не приводит к положительному эффекту. После этого общий набор инструкций (N) дублируется и каждая из двух его копий помещается после своего класса ветвей. Далее для ветвей, где производилось отключение компонента C, добавляется команда на включение этого компонента (E_1). В ветвях, где отключения не производилось, нет необходимости и для добавления команды на его подключение.

Теперь рассмотрим, как нужно оптимизировать код в случае циклов.

На Рис. 3-А изображен цикл, перед началом которого есть блок из n_1 инструкций, не требующих работы компонента C. Также в начале тела цикла у нас есть блок, состоящий из k_1 инструкций, которым также не требуется компонент C. Важной особенностью этого участка кода является то, что на первой итерации цикла блок K_1 выполняется непосредственно после блока N, а на всех последующих итерациях цикла – после блока K_f.

В конце цикла есть блок K_f, который не требует компонента C, а сразу после окончания цикла есть еще блок N_2, также не требующий компонента C. Следует помнить, что любые из этих блоков могут быть пустыми.

Данный код может оставаться в неизменном виде, если выполняются все условия:

1) $P(N_1) + P(K_1) \geq P_{min}$
2) $P(K_f) + P(K_1) \geq P_{min}$
3) $P(K_f) + P(N_2) \geq P_{min}$

Другими словами, если от момента отключения компонента до момента его включения всегда выполняется достаточное количество инструкций, чтобы эффект экономии стал положительным, то такая схема программы имеет право на существование.

В противном случае энергосбережение будет не оптимальным и инструкции должны быть переупорядочены.

Если у нас выполняется условие $P(N) + P(K_1) \geq P_{min}$, то значит, мы можем получить экономию по крайней мере при выполнения первой итерации цикла. В этом случае имеет смысл произвести развертывание цикла и вынести первую итерацию так, чтобы для нее можно было бы добавить отдельный набор инструкций для отключения компонента C. Для того чтобы уменьшить

количество копируемого кода можно выносить из цикла не все его тело, а только блок K_1. Тогда программа примет вид, показанный на рисунке 3-Б.

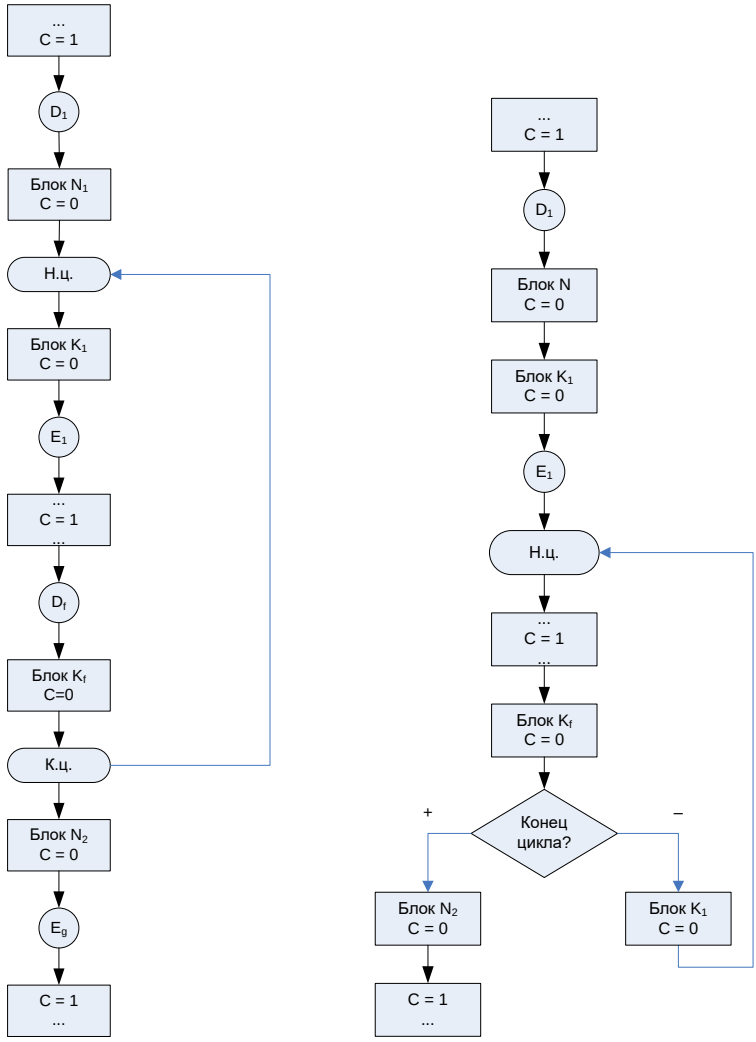


Рис. 3 – цикл до оптимизации (А) и после оптимизации первой итерации (Б).

Если у нас выполняется только условие $P(K_f) + P(N_2) \geq P_{min}$, то требуется вынести из тела цикла только его последнюю итерацию и вставить инструкцию D_g только в нее, оставив также инструкцию E_g после выполнения

блока инструкций, расположенных за циклом. Все остальные случаи оптимизации этого цикла различаются только тем, какие из трех условий будут выполняться. В зависимости от комбинации условий, отличия в оптимизациях будут заключаться только в том, какие из итераций (первая или последняя) должны быть вынесены из цикла, и в каких из указанных на рисунке 3-А позиций D_i и E_i должны быть установлены инструкции отключения или включения анализируемого компонента.

В заключении следует еще раз отметить, что предметом рассмотрения данной статьи были модификации кода без изменения относительного порядка следования инструкций. В ходе работы над статьей было еще раз отмечено, что для эффективного решения поставленной задачи должны быть дополнительно рассмотрены и некоторые смежные области. Так, например, в данной работе не производилось анализа возможностей переупорядочивания инструкций на линейных участках программы. Для того чтобы проделать эту работу требуется произвести анализ зависимостей в исходном коде. Такое исследование будет произведено в одной из следующих статей в этом направлении. За основу такого анализа можно взять статью [2], где описываются общие методы оптимизации, а также рассматриваются виды зависимостей.

Еще одним направлением дальнейших исследований является анализ результатов применения широко известных оптимизаций быстродействия для решения задач энергоэффективности. Сделанные наблюдения и анализ литературы показывают, что хотя данные области и располагаются достаточно близко друг к другу, но, тем не менее, на данный момент нельзя точно обосновать зависимость получаемого энергетического эффекта от таких оптимизаций.

Более того, задача построения оптимальной последовательности оптимизаций быстродействия на данный момент также не решена в общем виде, а значит, нет надежных метрик, позволяющих сравнить два различных набора оптимизаций, чтобы выбрать наиболее эффективный из них. Вопрос построение набора метрик также станет темой отдельного исследования.

Следует учитывать также взаимные влияния отключаемых компонентов друг на друга. В данной статье было произведено только исследование независимых компонентов. Анализ возможных зависимостей между ними также станет темой отдельной статьи.

Литература

[1] Yi-Ping You, Chingren Lee, Jenq Kuen Lee, Compilers for leakage power reduction, ACM Transactions on Design Automation of Electronic Systems, v.11 n.1, p.147-164, January 2006

[2] David F. Bacon , Susan L. Graham , Oliver J. Sharp, Compiler transformations for high-performance computing, ACM Computing Surveys, v.26 n.4, p.345-420, Dec. 1994

Восстановление формата данных

А.И. Гетьман, Ю.В. Маркин, В.А. Падарян, Е.И. Щетинин
{thorin, ustas, vartan, schet}@ispras.ru

Аннотация. Одной из распространенных практических задач анализа бинарного кода является восстановление структуры полученного программой сетевого сообщения или считанного файла. В случае работы аналитика с защищенным бинарным кодом трудоемкость восстановления формата данных становится недопустимо большой. В статье предлагается метод автоматизированного восстановления формата данных, базирующийся на динамическом анализе бинарного кода. Метод позволяет восстановить иерархическую структуру изучаемых данных и выявлять определенную семантику полей. Помимо того, представлены прототипная версия ПО, поддерживающего данный метод, и результаты работы данного ПО на модельном примере.

1. Введение

В настоящее время задача восстановления протоколов и файлов является весьма актуальной в сфере обратной инженерии и информационной безопасности. Решение этой задачи вручную является весьма затратным по времени и ресурсам, поэтому в настоящее время разрабатывается большое количество программных средств, позволяющих автоматизировать этот процесс. Восстановленная спецификация может использоваться при предотвращении несанкционированного доступа (intrusion prevention), для реализации технологии глубокого инспектирования пакетов (DPI), а также при тестировании приложения по методу «чёрного ящика» (black-box fuzzing) и сравнительном тестировании разных реализаций серверов.

Рассматривается задача восстановления формата сообщений используемых программой в рамках некоторого протокола. Под протоколом понимается набор правил, по которым осуществляется обмен сообщениями (байтовыми буферами известного размера) между двумя процессами. В частности, одним из участников обмена может являться файловая система – файл, в этом случае, рассматривается как совокупность сообщений. Основными компонентами протокола являются автомат состояний протокола и набор форматов сообщений, входящих в протокол. Задача восстановления автомата состояний в данной работе не рассматривается, так как её решение напрямую зависит от точности и полноты решения основной задачи – восстановление формата сообщений.

Формат сообщения, это совокупность знаний о полях сообщения, которые хранят базовые типы данных, а также о группировке полей в структуры и последовательности. Кроме того, в понятие формата входит информация о семантике полей – их роли в структуре сообщения. Примером семантически нагруженного поля может служить поле длины – оно хранит значение, равное размеру некоторого другого поля. От этого значения, следовательно, зависит длина сообщения.

Спецификация сообщения содержит, таким образом, следующие данные:

- Границы отдельных полей в сообщении.
- Группировка полей в структуры и последовательности.
- Семантика полей.
- Информация о значениях полей, входящих в сообщение.

Предлагаемое в данной работе решение включает в себя систему автоматического восстановления формата сообщений на основе динамического анализа трассы программы-разборщика, систему взаимодействия с пользователем, которая помимо основного результата – формата сообщения выдаёт данные, на основе которых пользователь может вручную уточнить полученный формат, а также систему хранения восстановленный форматов на основе базы данных MySQL.

Особенностью предлагаемого решения является отказ от полностью автоматического подхода в пользу автоматизированной системы анализа. Причиной такого выбора является то что, как показывает анализ предлагаемых программных решений, автоматический анализ в ряде случаев не может полностью решить проблему, а ручной анализ в этом случае весьма затруднён. В данной работе делается попытка с одной стороны использовать все преимущества автоматического анализа, а с другой предоставить пользователю гибкий инструмент для уточнения и коррекции, полученных в ходе автоматического анализа результатов.

Статья организована следующим образом. Во втором разделе рассмотрены известные подходы к решению задачи восстановления формата данных, проанализированы их слабые и сильные стороны. Третий раздел описывает схему работы предлагаемого метода восстановления формата данных. Далее более подробно рассматриваются отдельные аспекты этого метода. В четвертом разделе показано как входной буфер разделяется на поля примитивных типов данных, которые затем группируются в иерархическую структуру. Пятый раздел посвящен выявлению семантики отдельных полей. В шестом разделе рассматриваются способы обобщения восстановленного формата, если аналитик располагает несколькими экземплярами полученных сообщений. Практические результаты работы предложенного метода и заключение даны в седьмом разделе.

2. Обзор работ

Существует несколько различных подходов к восстановлению формата сообщений. Основными являются:

1. Анализ потока данных (например, сетевого трафика) с выделение повторяющихся паттернов. Основной недостаток – сложность определения семантики полей. Этот подход рассматривается в работе [1]
2. Статический анализ кода одного из участников обмена. Основной недостаток – наличие защиты от отладки и применение обфускации в современных программных средствах. Этот подход рассматривается в работе [2]
3. Динамический анализ кода одного из участников обмена. Основной недостаток – для анализа доступна только выполнявшаяся часть кода, вследствие чего формат сообщения не всегда может быть восстановлен полностью. Этот подход рассматривается в работах [3,4,5,6].

Для реализации был выбран динамический подход, так как в случае защищённых приложений он способен дать наиболее точные результаты.

Восстановление формата сообщений состоит в восстановлении двух типов информации: структурной и дополнительной. Структурная информация состоит из двух типов данных – границ полей и семантической информации, которая необходима для правильного разбора сообщения: список полей, задающих длину (length fields), поля-разделители (delimiter fields), поля-указатели (pointer fields), поля содержащие «магические константы» протокола и ключевые значения (key fields). Кроме того, к структурной информации относятся вариации сообщений одного типа, например наличие или отсутствие некоторого «плавающего» (floating) поля и иерархическая структура сообщения (вложенность одних полей в другие). Дополнительная информация включает в себя знания о содержимом полей: ключевые слова, имена файлов, идентификаторы сессии. В рассматриваемых ниже работах предлагаются некоторые методики для восстановления как структурной, так и дополнительной информации.

В работе [2] предлагается методика восстановления формата данных генерируемых некоторой программой, с помощью статического анализа этой программы. Для работы анализа от аналитика требуется специфицировать функции вывода, использующиеся программой. На основании этих данных строится межпроцедурный граф потока управления, который затем проецируется на вызовы функций, специфицированных пользователем. На основании этого графа генерируется формат в виде регулярного выражения. Для группировки полей в структуры используется алгоритм ASI[7]. Основным недостатком этой работы является игнорирование семантической информации, такой как поля длины и разделители. Кроме того, требование

ручной спецификации функций вывода сильно ограничивает применение данного подхода.

В работах [3,4,5,6] используется динамический анализ трасс, основанный на dynamic taint analysis [8], позволяющий выделить подтрассу обработки сообщения. В работе [3] описывается инструмент Polyglot, в котором впервые реализован ряд методик анализа семантики полей, таких как поле длины и разделитель, которые, с незначительными изменениями используются в последующих работах. Областью применения данного инструмента следует считать преимущественно текстовые протоколы (HTTP). Это накладывает определённый отпечаток на реализованные алгоритмы. В частности, большинство эвристик, применяемых в инструменте, используют статистические данные, полученные на основе использования для указанного класса задач. Работа [4] снимает ограничение на область применимости – рассматриваются как текстовые, так и бинарные протоколы, однако перед началом анализа необходимо указать, к какой именно группе принадлежит анализируемый протокол. Кроме того, восстанавливается не только «плоский» формат сообщения (последовательность полей), но также иерархическая структура и зависимости между полями. Взаиморасположение двух элементов в структуре могут быть 3-х типов: последовательность, вложенность, параллельность. Пример формата приведён на рис. 1.

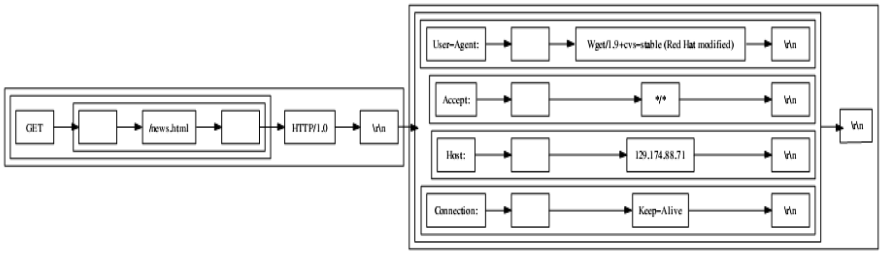


Рис.1. Пример структуры формата.

Применение подхода ограничено необфусцированными программами. Эвристика выделения параллельных полей основана на статистике по некоторому набору протоколов, что также ограничивает её применимость. Помимо прочего, в инструментах [3,4] отсутствует возможность уточнения формата на основе анализа программы на разных входных данных. Две различные методики, предоставляющих возможность обобщения формата сообщения на основе анализа нескольких трасс описываются в статьях [5,6]. Для обобщения формата требуется предварительно сопоставить элементы разных структур. Для решения этой задачи в работе [6] применяется анализ контекстов обработки этих элементов (наборов инструкций (их адресов), которые их обрабатывают). Преимуществом этого подхода является его

сравнительно высокая точность. Однако этот подход неприменим, во-первых, к обобщению на основании анализа разных реализаций обработчика (в том числе разных версий), во-вторых, при наличии навесной защиты сопоставление инструкций на основании адресов может быть некорректным. Методика, предложенная в работе [5], не имеет этих недостатков. Она основана на применении модифицированного алгоритма Нидлмана-Вунша для сравнения иерархических структур данных (Hierarchical Needleman-Wunsch).

В работе [6] описывается программный комплекс Tupni. Показывается эффективность этого инструмента при восстановлении формата на широком классе протоколов и файлов. В статье предложена оригинальная методика выделения последовательностей и структур в сообщении на основе анализа циклов в графе потока управления, которая в уточнённом виде использовалась при реализации описываемой системы. Восстановление семантики полей длины и разделителей осуществляется на основании анализа терминальных условий циклов. Среди ограничений подхода можно указать байтовую гранулярность при анализе данных – это не позволяет указывать поля, содержащие флаги. Помимо этого, для анализа циклов используется граф потока управления, полученный в ходе предварительного статического анализа программы. Такой анализ для защищённых приложений возможен далеко не всегда.

3. Описание системы восстановления формата

Решение поставленной задачи предполагает реализацию автоматизированной системы анализа, основными компонентами которой являются:

1. Набор алгоритмов анализа формата
2. Системы получения и отображения информации, на основе которой аналитик может оценить и проверить точность полученных результатов и принять решение о необходимости их ручной корректировки

Особенностью реализуемой системы, является предоставление пользователю как основных результатов анализа – спецификации формата сообщения, так и дополнительных данных, которые могут облегчить пользователю ручную коррекцию полученной спецификации. В качестве дополнительных данных выдаётся привязка выделенных элементов структуры к блокам шагов трассы, участвующим в их обработке. Кроме того, выдаётся полная подтрасса обработки сообщения, по которой может быть построен граф зависимостей, отражающий процесс преобразования каждого поля. Эта подтрасса также может служить входными данными для дальнейшего анализа, например, в случае зашифрованного протокола по этой трассе можно вычислить буферы, куда помещены расшифрованные данные, и рекурсивно повторить процедуру восстановления для этих буферов. Схема работы системы представлена на рис. 2.

Дополнительным источником информации о структуре сообщения могут быть спецификации некоторых функций, участвующих в обработке сообщения. Например, если некоторое поле передаётся как параметр размера в функцию malloc, то оно, вероятно, является размером некоторой последовательности. Другие примеры использования спецификаций есть в работах [2,6]. Предоставление такой информации, также отражено на схеме пунктиром.

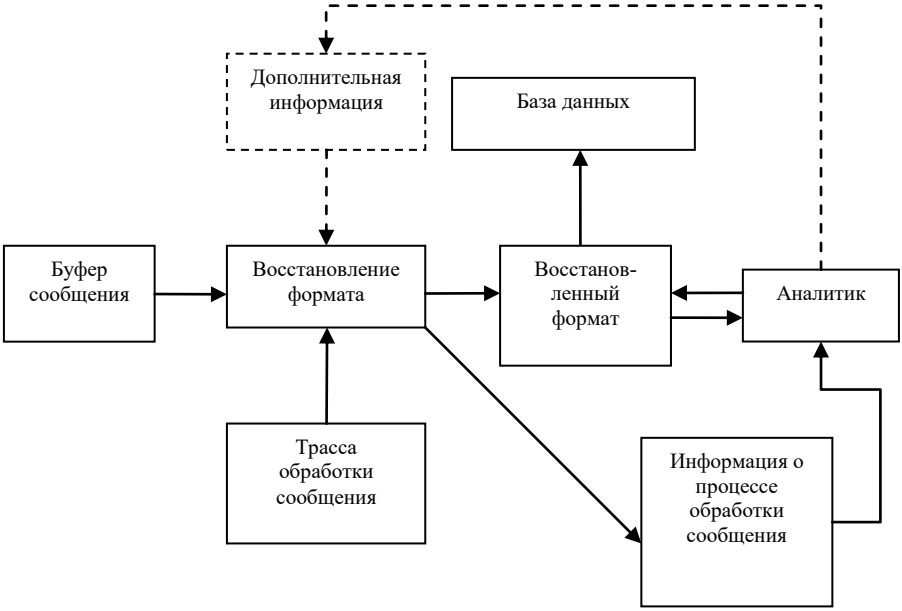


Рис. 2. Упрощённая схема работы системы восстановления..

3.1. Представление формата данных

Опишем сначала структуру формата, восстанавливаемую в ходе анализа.

Формат представляется в виде дерева, корневой вершиной которого является само сообщение, листовые вершины – поля, соответствующие базовым типам данных, а внутренние вершины подразделяются на несколько типов, определяющих группировку полей. Один из типов группировки – последовательность, аналог массива языка C, отличие заключается в том, что элементы последовательности могут иметь разную структуру и размер, вследствие чего доступ к ним осуществляется последовательно, а не в произвольном порядке. Кроме того, длина последовательности может быть неизвестна на момент начала её обработки, если она определяется полем разделителем (как в случае ограниченной нулем строки). Другой тип группировки – структура, аналог `struct` языка C. Поля, объединённые в структуру, логически связаны между собой и обрабатываются

локализованным кодом. Варианты, представляющие собой аналог `union` языка C, введены для группировки элементов последовательности, которые имеют разную структуру. Это сделано для того чтобы отразить факт, неоднородности последовательности. Всем вершинам сопоставлен размер соответствующих структурных единиц и их смещение относительно родительской вершины.

Семантическая информация хранится в виде атрибутов-меток, которые отражают некоторые функциональные свойства полей: разделитель, длина, указатель, функция, ключ, флаг. Пометка разделитель определяет поле, значение которого интерпретируется как конец последовательности. Пометка длина обозначает поле, определяющее количество идущих подряд полей или структур, составляющих последовательность. Пометка указатель обозначает поле, с помощью которого вычисляется указатель на некоторое другое поле сообщения. Пометка функция обозначает поле, имеющее произвольную функциональную зависимость от значений других полей, например, в случае вычисления контрольной суммы. Пометка ключ обозначает поле, в зависимости от значения которого определяет способ разбора некоторой части сообщения. Поля флаги интерпретируются, как совокупность отдельных битов, значение которых определяет поведение разборщика. Ещё один атрибут поля – признак виртуальности, этот атрибут означает, что последовательность байт, соответствующая этому полю, не обрабатывалась в анализируемой трассе программы. Следовательно, об этой последовательности байт на основании анализа текущей трассы ничего сказать нельзя.

3.2. Структура модуля восстановления формата

Перечислим основные компоненты анализа и опишем связи между ними.

1. Компонент выделения циклов в графе потока управления
2. Алгоритм проектирования выделенных циклов на трассу
3. Алгоритм выделения подтрассы обработки буфера сообщения
4. Алгоритм поиска границ полей сообщения
5. Компонент выделения последовательностей
6. Компонент выделения структур
7. Компонент построения дерева формата
8. Компонент обобщения формата
9. Компонент анализа семантики, который включает в себя следующие части:
    - поиск полей длины выделенных последовательностей
    - поиск полей-разделителей выделенных последовательностей
    - поиск полей-указателей
    - поиск полей, содержащих ключевые значения
    - поиск полей-флагов

Общая структура анализа может быть представлена в виде схемы на рис. 3.

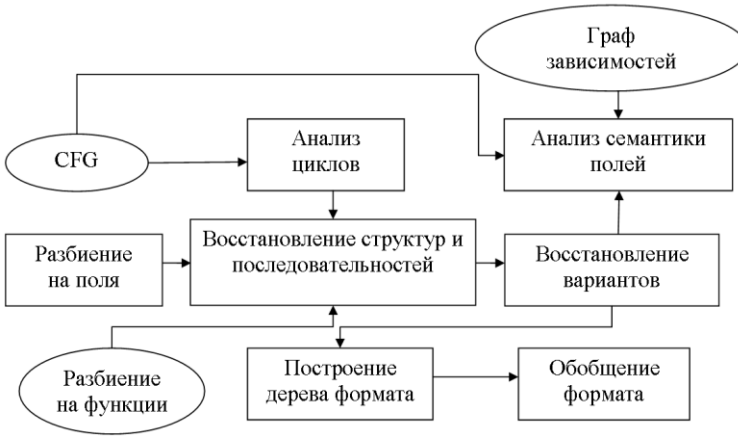


Рис. 3. Общая структура анализа..

Эллиптические вершины на рисунке обозначают данные, которые необходимы алгоритму для дальнейшего анализа. Прямоугольные вершины обозначают работу алгоритмов.

4. Анализ структуры сообщения

Источником информации о структуре сообщения служит трасса выполнявшихся инструкций процесса разбора этого сообщения. Для начала анализа требуется определить точку в трассе, в которой буфер сообщения уже заполнен, но разбор ещё не осуществлялся. Затем, методом прямого слайсинга, из трассы выделяется алгоритм обработки буфера. На основании размеров операндов инструкций доступа к буферу сообщения восстанавливаются размеры отдельных полей. По трассе частично восстанавливается статическое представление программы – граф потока управления (далее CFG). Следует отметить, что в данном случае его неполнота не сказывается на результатах восстановления. Анализ циклов, выделенных из CFG и спроецированных на трассу, позволяет сгруппировать поля в структуры и последовательности. Дополнительный семантический анализ условий выхода из циклов позволяет восстановить семантику отдельных полей. Полученная таким образом информация представляется в виде дерева формата. Затем производится обобщение этого дерева на случай произвольного сообщения данного типа. Это необходимо, для того чтобы можно было объединять структурную информацию, полученную при анализе разных сообщений одного типа. Основа методик, использующихся при анализе структуры сообщения, взяты из работы [6]. Они были усовершенствованы, что позволило с одной стороны получать больше информации на тех же примерах (флаговые поля), а с другой расширило

область применения методик (защищённый код, последовательности с полями указателями).

4.1. Восстановление полей

Под восстановлением полей имеется в виду задача выделения из последовательности байт сообщения таких непрерывных подпоследовательностей, каждая из которых интерпретируется обрабатывающим кодом как одно число.

Рассматриваются инструкции, обрабатывающие данные лежащие в буфере. При этом инструкции копирования не учитываются, так как они не осуществляют обработку данных. На основе размеров операндов (в битах) этих инструкций соответствующих данным буфера производится выделение полей. Последовательности байт буфера, к которым доступ не осуществлялся, помечаются как виртуальные поля.

Проблему представляет неоднозначная интерпретация сообщения в обрабатывающем его коде. Например, в основном цикле обработки сообщения, его поля могут восприниматься, как 4х байтовые целые, а в процессе подсчёта контрольной суммы, всё сообщение может восприниматься как массив однобайтовых величин. Для решения этой проблемы применяется эвристика, основанная на сопоставлении каждому выделенному полю веса, равного количеству инструкций, которые к нему обращались. В этом случае задача выделения полей сводится к определению покрытия буфера полями с максимальным весом. Вследствие высокой сложности алгоритма поиска оптимального решения применяется жадный алгоритм, который показал хорошие результаты в процессе применения к реальным задачам. Результатом работы алгоритма является разбиение буфера сообщений на поля, соответствующие базовым типам данных сообщения.

4.2. Восстановление последовательностей записей

В рамках данной задачи рассматривается взаимное положение разных полей. Вводится эвристическое предположение, что поля, обрабатывающиеся в цикле, образуют последовательность, причём на каждой итерации цикла обрабатывается одна запись – группа полей. Другой возможностью обработки последовательностей являются рекурсивные вызовы, однако ресурсоёмкость такого способа гораздо выше, а скорость ниже. На практике такой способ не встречался. Для каждой последовательности можно определить способ задания её длины. По аналогии с работой [6] выделяются следующие возможности:

- длина фиксирована протоколом
- длина явно задаётся значением поля (поле длины)
- длина неявно задаётся полем разделителем.

Алгоритм, определяющий способ задания последовательности является частью анализа семантики.

Примером последовательности записей может служить 0-терминированная строка, считывающаяся в цикле.

Для начала работы алгоритма требуется информация о статической структуре программы в виде графа потока управления. В данной реализации CFG предварительно строится на основе информации из трассы – не использую статический анализ бинарного кода. Это связано с тем, что в современных программах применяется большое количество средств защиты от анализа. К ним можно отнести применение программ-загрузчиков и шифрация кода. В результате применение статического анализа может быть значительно затруднено. Построение CFG на основе динамической информации из трассы позволяет преодолеть эти проблемы. Следует отметить, что, так как в трассе, как правило, реализуются далеко не все пути программы, то полученный CFG будет неполным – в нём будет отсутствовать информация о нереализованных ветвях исполнения. Однако его неполнота не сказывается на результатах восстановления. Поясним сказанное.

В рамках задачи восстановления последовательностей требуется найти циклы в CFG. Поиск циклов осуществляется с помощью алгоритма из книги [9]. Этот алгоритм основан на поиске обратных рёбер – для каждого обратного ребра может быть выделен цикл, которому оно соответствует. Если тело цикла было выполнено более одного раза, то обратное ребро попадёт в трассу и будет присутствовать в восстановленном CFG. Для случая вырожденного цикла, когда тело выполнялось всего один раз - обратное ребро в трассе отсутствует. Для того чтобы восстановить цикл в этом случае, применяется методика анализа условных переходов, которая на основании вычисления целевого адреса перехода позволяет восстановить ребро графа потока управления, если базовый блок, на который указывает переход присутствует в графе. Для циклов последнее условие всегда выполнено, поэтому все исполнявшиеся циклы будут присутствовать в восстановленном CFG.

Каждый выделенный на CFG цикл проектируется на трассу. Из полученных проекций выбираются только те, внутри которых есть обращение к данным буфера сообщения. Для этих циклов производится анализ отдельных итераций. Для каждой итерации строится множество релевантных инструкций. В него добавляются инструкции, которые осуществляют доступ к некоторому полю только на этой итерации – это позволяет отделить от полей последовательности поле длины, обращение к значению которого осуществляется более чем на одной итерации. Если то множество не пусто на каждой итерации, быть может, кроме последней, то такой цикл назовём итерационно-зависимым. Поля, обрабатывающиеся в релевантных инструкциях группируются в последовательность если:

- Поля образуют непрерывную байтовую последовательность на каждой итерации, если их дополнить виртуальными полями.

- Поля обрабатываются в прямом порядке, то есть адреса всех полей обрабатывающихся на итерации n меньше чем адреса полей, обрабатывающихся на всех итерациях k, где k > n.

Дополнительные сложности при выделении возникают при наличии внутри последовательности полей-указателей. При этом часто возникает ситуация, когда поля, доступ к которым осуществляется по этим указателям, также обрабатываются в цикле. В этом случае нарушается условие непрерывности обработки.

Для того чтобы восстанавливать последовательности и таких случаях была разработана дополнительная методика анализа доступа по полям-указателям. По результатам анализа – все инструкции доступа разделяются на две группы – доступ по полю указателю и доступ без указателя. Эти группы при анализе циклов обрабатываются отдельно, что позволяет восстанавливать последовательности точнее.

4.3. Восстановление структур

Под восстановлением структур понимается задача группировки последовательно расположенных полей, которые логически связаны друг с другом и обрабатываются кодом, относящимся к одному блоку шагов трассы. В качестве блоков шагов может выступать трасса отдельного вызова функции или трасса одной итерации некоторого цикла. В текущей реализации, в качестве блоков шагов рассматриваются только отдельные итерации циклов, так как анализ проведённый в статье [4] показывает, что применение вызовов функций для группировки полей ограничено текстовыми протоколами типа HTTP.

При восстановлении записей может применяться информация, полученная от пользователя – определения того, что пользователь считает блоками шагов. Для текстовых протоколов, пользователь может передать в частности границы вызовов функций. Это позволит уточнить полученные результаты.

Группируемые поля должны обладать свойством непрерывности – с учётом виртуальных полей они должны образовывать непрерывную последовательность байт.

Для того чтобы определить границы структур обрабатывающихся на разных итерациях цикла применяется следующая методика. Среди релевантных инструкций на первой итерации цикла выбираются те, которые осуществляют доступ по минимальному смещению. Это смещение считается началом первой структуры. Кроме того, предполагается, что эти же инструкции обращаются к началу каждой следующей структуры на последующих итерациях. Конец структуры на итерации n определяется началом структуры на итерации n + 1. Конец последней структуры определяется как максимальное смещение, по которому есть обращение на последней итерации.

Структуры могут быть вложенными друг в друга, то есть одна структура может являться частью другой. Эта ситуация может возникать, например, при

вложенных вызовах (внешний вызов обрабатывает охватывающую структуру, а вложенный - вложенную) или вложенных циклах. Таким образом, в процессе восстановления структур проявляется иерархическая структура сообщения.

5. Анализ семантики полей

Задача выявления семантики полей подразумевает определение роли некоторых полей (их значений) в формировании структуры сообщения. Рассмотрим подробнее предлагаемые алгоритмы.

5.1. Поиск полей длины

Значение поля длины тем или иным способом задаёт размер последовательности, которому оно соответствует. Оно может содержать, как размер последовательности в байтах, так и количество элементов в последовательности. В любом случае, это значение должно использоваться в условии выхода из цикла (которое проверяется на каждой итерации), при этом на каждой итерации это условие должно не выполнятся, а на последней должно быть выполнено. Именно на этих свойствах поля длины основан алгоритм его поиска. На первом шаге определяется поле, значение которого использовалось в условии выхода на последней итерации. Далее, на каждой итерации, проверяется, что в условии выхода используется значение этого поля и, при этом условие не выполняется ни на одной итерации.

5.2. Поиск полей-разделителей

Поле разделитель, это поле, которое следует сразу после некоторой последовательности, а его значение интерпретируется обработчиком, как терминальный символ, завершающий последовательность. По определению, его значение – это некоторая константа или набор констант (как в случае HTTP), значение которых известно обработчику. Для того чтобы определить конец последовательности обработчик сравнивает каждый элемент последовательности с терминальным символом (константой). Это используется в алгоритме поиска полей-разделителей.

Для цикла осуществляется поиск всех сравнений с константами, среди релевантных инструкций. Среди этих сравнений оставляют только те, которые выполнены на последней итерации и не выполнены на всех предыдущих. Поля, к которым осуществляется доступ в оставшихся инструкциях на последней итерации, объявляются разделителями. В текстовых протоколах таких полей может быть несколько (в HTTP как разделитель используется пара символов “\r\n”), в бинарных поле, как правило, одно.

5.3. Поиск полей-указателей

Основное свойство таких полей – использование в коде их значений для формирования указателя на некоторое другое поле сообщения. Их выявление основано на анализе способа формирования адреса для доступа к полям. В

случае если адрес формируется на основе значения некоторого поля, поле добавляется во множество указателей. Приведём пример:

```
MOV AX, dword ptr [BX + CX]
```

Если ячейка памяти является полем сообщения A, а значение BX или CX зависит от значения некоторого другого поля сообщения B, то поле B является полем-указателем. Стоит отметить, что в отдельных случаях значения полей длины также могут использоваться подобным образом, поэтому из множества полей-указателей должны быть удалены поля длины.

5.4. Поиск ключевых полей

Под ключевыми полями имеются в виду поля, которые могут содержать заранее известный для данного протокола набор значений. В процессе анализа сообщения, разборщик проверяет значения таких полей, сравнивая их с этим набором и, в зависимости от результата сравнений, осуществляет дальнейший разбор. Таким образом, значение данного поля определяет конкретный вид и интерпретацию некоторой части сообщения. Примером такого поля является поле «Тип ресурсной записи» протокола DNS, значение которого определяет, что содержит данная ресурсная запись.

Поиск ключевых полей определяется способом их обработки – требуется найти поля, значения которых сравнивается с некоторым набором констант, причём одно из сравнений истинно, и по результатам сравнения осуществляется условный переход в обработчике. Шаблон поиска:

```
CMP field, const
ConditionalJump target
```

Константы, фигурирующие во втором операнде, при анализе интерпретируются как ключевые значения протокола. Однако только константа, сравнение с которой дало истину действительно является ключевым сообщением. Другие же сравнения подходящие под шаблон могут являться как перебором разборщиком ключевых значений, так и результатом применения обфускации, вставляющей мёртвый код (сравнения, которые заведомо не выполнятся). В результате требуется принять решение – применять консервативный подход и учитывать только константы истинных сравнений или оптимистичный, предполагающий отсутствие такого рода обфускации, сохраняющий все константы. В данной работе принят промежуточный подход – сохраняются все константы, но при этом они разбиты на две группы: встречающиеся в истинных сравнениях и остальные. При этом константы первой группы точно являются ключевыми значениями протокола, а константы второй группы являются таковыми при условии отсутствия обфускации.

Следует отметить, что этому шаблону также соответствуют поля-разделители. Поэтому из множества найденных ключевых полей требуется удалить поля разделители.

5.5. Поиск полей флагов

Битовая гранулярность работы с памятью даёт возможность анализировать работу с отдельными битами сообщения. В архитектуре x86 существует набор операций (BT, BTC, BTR, BTS), позволяющий оперировать отельными битами регистров и ячеек памяти. Однако на практике при анализе флагов в полях сообщения чаще используется другой шаблон работы – применение логического «И» к значению некоторого поля, причём в качестве второго операнда выступает константа, значение которой определяет, какой бит (или группа битов) интересует программу. В архитектуре x86 такая операция представлена двумя инструкциями – AND и TEST, первая из которых разрушает исходное значение ячейки, а вторая только устанавливает флаги по результатам применения логической операции. По результатам проверки битового поля осуществляется условный переход. Соответственно, поиск битовых полей осуществляется по двум шаблонам –

```
BT field, bitNum
ConditionalJump target
```

и

```
“AND” field, const
ConditionalJump target
```

Границы битового поля в обоих шаблонах определяются вторым операндом первой инструкции.

5.6. Особенности применения шаблонов

Дополнительной трудностью при применении шаблонного анализа является наличие обфускации и оптимизированного кода. В результате между инструкциями сравнения (первой инструкцией шаблонов) и условным переходом (вторая инструкция шаблона) может присутствовать мёртвый код (результат применения обфускации) или инструкции, которые можно выполнить параллельно с инструкцией сравнения (результат оптимизаций). Это приводит к тому, что указанные шаблоны лучше применять не на последовательном потоке инструкций, а непосредственно на графе зависимостей, в котором между инструкцией сравнения и инструкцией условного перехода будет присутствовать зависимость по данным. Таким образом, шаблоны преобразуются к тривиальному виду, показанному на рис. 4.

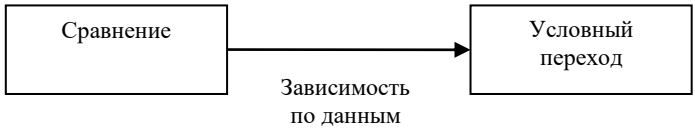


Рис. 4. Шаблон поиска на графе зависимостей.

6. Дерево формата

На основании данных о границах полей и их группировке в структуры и последовательности строится дерева формата. Вершинами дерева являются: само сообщение, последовательности, структуры, поля. Элемент A соответствующий диапазону адресов [x1, y1] сообщения, является предком элемента B [x2, y2] тогда и только тогда, когда x1 <= x2 и y1 >= y2.

6.1. Обобщение дерева формата

Одной из целей автоматического восстановления формата сообщения является автоматическая генерация парсера для всех сообщений данного типа. Следствием применения динамического анализа является то, что для анализа доступна только часть программы, участвующая в обработке некоторого конкретного сообщения. Для преодоления этого ограничения производится обобщение формата конкретного сообщения, таким образом, чтобы результирующий формат соответствовал более широкому классу сообщений данного типа. В качестве примера рассмотрим следующую возможность:

Пусть разобранное текстовое сообщение представляет собой ограниченную нулем строку ASCII-символов «Hello world!» длиной 12.

Восстановленный формат будет соответствовать классу сообщений, изображённому слева на рис. 5. Если предположить, что все сообщения данного типа содержат ровно 12 символов, то поле-разделитель является избыточным. Так как сетевые сообщения, как правило, не содержат избыточности, то это приводит к выводу, что на самом деле текстовое сообщение может быть любой длины. Именно такой формат и будет результатом применения операции обобщения к восстановленному формату – он показан на рис.5 справа. Такой формат позволит сгенерировать парсер для более широкого класса сообщений, чем исходный.

Обобщение включает в себя три основные операции:

1. Определение элементов, длина которых может быть неопределённой
2. Распространение неопределённого смещения и размера по дереву формата
3. Определение неоднородных последовательностей (последовательностей, элементы которых имеют разную структуру) и группировка таких полей с помощью элементов типа вариант

Неопределённую длину могут иметь два типа элементов – последовательности, длина которых задаётся с помощью поля разделителя или поля длины и варианты, если их дети имеют разную длину. Алгоритм, определяющий является ли некоторая последовательность однородной, а также описание типа вариант будет описано в следующем разделе. Далее приводится алгоритм распространения неопределённости в смещении и размере по дереву формата.

1. Пусть U – множество элементов, длина которых неопределенна.

2. Берём следующий элемент u из множества U, элемент p = parent(u) – его предок в дереве формата, если его размер имеет определённое значение, то делаем его неопределённым и добавляем p в множество U.
3. Для p, вычисляем множество childs(p), такое что, для каждого его элемента c выполнено c.offset > u.offset. Для всех элементов множества childs(p) выставляем неизвестное смещение.
4. Удаляем u из множества U.
5. Переходим на шаг 2.

На рис. 5 можно видеть результат применения операции 1 (элемент с неопределённой длиной – последовательность A) и 2 (смещение поля B в обобщённом формате не определено).

На рис. 6 приведён пример применения операции 3 к другому типу исходного сообщения, в котором фигурирует последовательность, чётные элементы которой имеют длину 1, а нечётные – длину 2. Так как информация о том, какие именно элементы имеют тот или иной размер анализатору недоступна, то алгоритм консервативно предполагает, что элемент последовательности с любым номером может иметь как размер 1, так и размер 2. При этом, так как не найден способ задания поля длины, предполагается что длина последовательности фиксирована и равна 12. В этих условиях, сколько именно полей будет содержать последовательность в конкретном сообщении, неизвестно – она может содержать от 6 (все поля длины 2), до 12 (все поля длины 1) элементов.

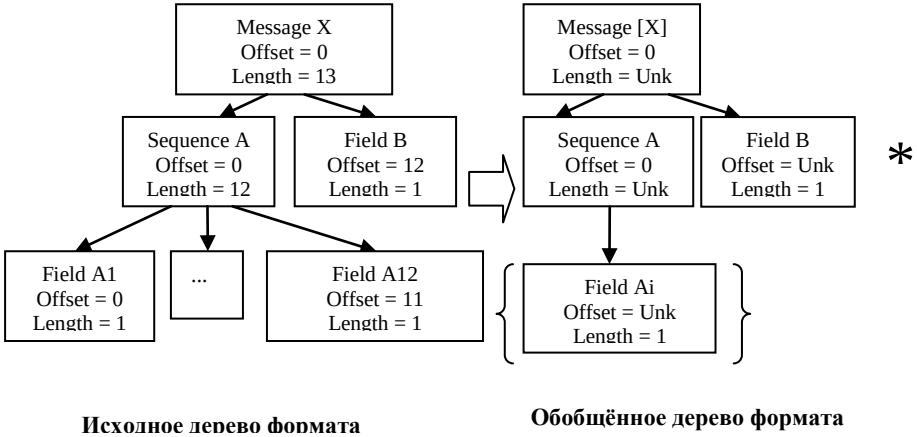


Рис. 5. Пример №1 обобщения формата.

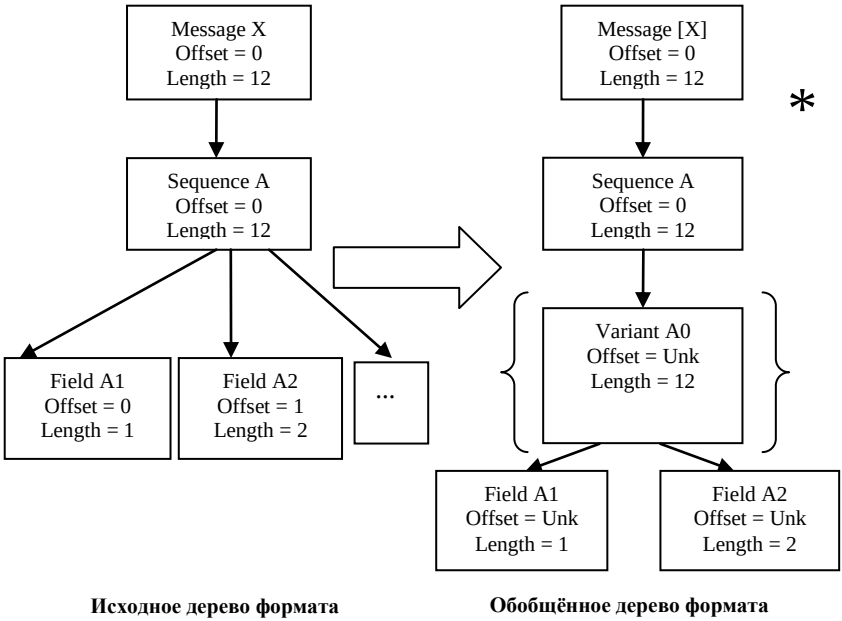


Рис. 6. Пример №2 обобщения формата.

6.2. Восстановление вариантов

Задача восстановления вариантов включает в себя анализ элементов некоторой последовательности с целью определения того, являются ли они объектами одного и того же класса или нет, а также генерацию элемента дерева типа вариант, дети которого соответствуют представителям всех обнаруженных классов. Элементу последовательности сопоставляется класс на основании следующих его характеристик:

- тип (поле, структура, последовательность);
- размер;
- семантика (в случае полей);
- сигнатура.

Сигнатура сопоставляется элементам структуры на основании особенностей их обработки в трассе. Алгоритм создания сигнатуры:

1. Если элемент – поле, то сигнатура это адрес инструкции первой обработки этого поля.
2. Если элемент – последовательность, то сигнатура это адрес инструкции входа в цикл обработки этой последовательности. Такой способ задания сигнатуры для последовательности позволяет выявить эквивалентность между последовательностями разной длины, и разного состава

элементов, которые, тем не менее, обрабатываются одним и тем же кодом.

3. Если поле – структура, то сигнатура это объединение сигнатур, полученных рекурсивным применением правил 1 и 2, для элементов, образующих структуру (её потомков в дереве формата).

Для каждого выделенного класса создаётся его представитель. Если выявлен ровно один класс, то его представитель включается в дерево, как единственный ребёнок последовательности. В случае если количество классов больше 1, создаётся элемент типа вариант, который интерпретируется, как представитель всех элементов последовательности. Он включается в дерево как единственный ребёнок последовательности, а в качестве его детей добавляются представители выделенных классов.

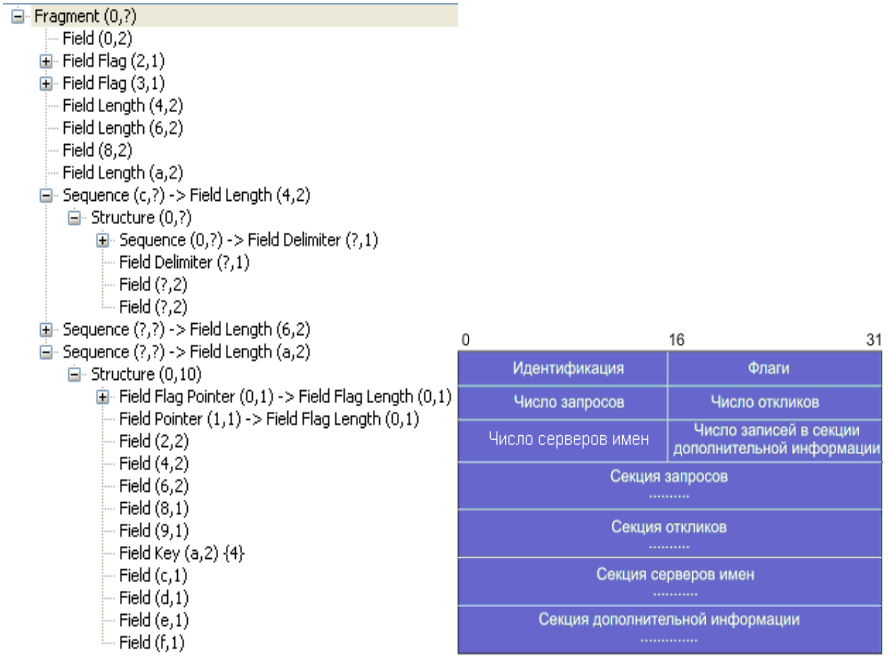
Результаты работы алгоритма восстановления вариантов показан на рис. 6.

7. Заключение

Предложенный в разделах 3-6 метод автоматизированного восстановления формата данных был поддержан прототипным программным средством, реализованным в рамках среды динамического анализа защищенного бинарного кода TrEx. Реализация представляет собой модуль-расширение с именем «ru.ispras.ctt.trex.algs.FormatRecovery»; модулю требуется для корректной работы наличие разметки в трассе вызовов функций и графы потока управления для каждой функции.

В качестве теста, для проверки точности восстановления был выбран протокол DNS. Его спецификация детально описана в документе RFC-1035. Протокол достаточно функционален и область его потенциального применения достаточно широка. При этом протокол содержит всего один тип сообщения, которое, однако, имеет сложный формат, содержащий все типы семантических полей – поля длины, разделители, указатели, флаги, ключевые поля. Тестировалась реализация разборщика DNS-сообщений в ОС Windows XP, nslookup. Для проверки работы алгоритма была снята трасса отправки запроса типа SOA (start of authority record — ресурсная запись DNS о сервере, хранящем эталонную информацию о домене) и получения соответствующего ответного сообщения. На рис. 7 слева приведена спецификация формата на основе RFC-1035, а справа спецификация, восстановленная системой. По результатам сравнения можно заметить следующие особенности – 2х байтовое поле флагов восстановилось как два отдельных флаговых поля – при проверке вручную, выяснилось, что данная реализация действительно обрабатывает эти 2 байта отдельно. Поле длины соответствующее числу серверов имён не восстановлено, так как при данном конкретном запросе секция серверов имён была пуста, и это поле не использовалось как поле длины. Таким образом, причина не полностью восстановленной семантики – принципиальное ограничение динамического подхода.

Для преодоления этого ограничения возможно использовать одну из методик объединения информации от анализа трасс разных сообщений. Две подобные методики рассмотрены в работах [5,6]. Для того чтобы объединить информацию о типах сообщений требуется сначала произвести операцию выравнивания, то есть сопоставить записи в одном сообщении соответствующим записям в другом, а затем произвести объединение форматов. Рис. 8 демонстрирует описанные операции. В работе [5] для реализации выравнивания используется модифицированный алгоритм Нидлмана-Вунша, впервые описанный в работе [10].



Спецификация DNS по RFC-1035 Фрагмент восстановленной спецификации

Рис. 7. Сравнение результатов восстановления формата DNS-сообщения с его спецификацией.

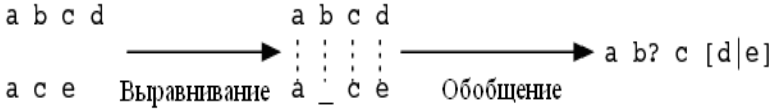


Рис. 8. Обобщение форматов двух сообщений. Буквами обозначены записи разных типов.

Ещё одна неточность восстановленной спецификации – не восстановленная последовательность байт содержащая IP-адрес сервера (последние 4 поля) и соответствующее ей поле длины (восстановленное как ключевое поле со значением 4). При ручном анализе соответствующего кода было выяснено, что он оптимизирован с применением алгоритма разворота цикла. Для преодоления этого типа оптимизаций требуется разработать соответствующий алгоритм шаблонного поиска развёрнутых циклов на графе потока управления.

Тем не менее, прототипная версия ПО уже используется на практике, что позволяет аналитику существенно сократить время, затрачиваемое на анализ соответствующего класса программ.

В дальнейшем предполагается продолжить работы по улучшению предложенной методики восстановления формата сообщений и развитию ее программной реализации.

Литература

[1] W. Cui, J. Kannan, H.J. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. // 16th USENIX Security Symposium, 2007. Pp. 199–212.

[2] J. Lim, T. Reps, B. Liblit. Extracting Output Formats from Executables. // Proceedings of the 13th Working Conference on Reverse Engineering, 2006. Pp. 167 – 178.

[3] J. Caballero, H. Yin, Z. Liang, D. Song. Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. // Proceedings of the 14th ACM conference on Computer and communications security, 2007. Pp. 317 – 329.

[4] Z. Lin, X. Jiang, D. Xu, X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. // Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[5] G. Wondracek, P. Milani Comparetti, C. Kruegel, E. Kirda. Automatic Network Protocol Analysis. // Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[6] W. Cui, M. Peinado, K. Chen, H.J. Wang, L. Irun-Briz. Tupni: Automatic Reverse Engineering of Input Formats. // Proceedings of the 15th ACM conference on Computer and communications security, 2008. Pp. 391 - 402.

[7] G. Ramalingam, J.Field, F. Tip. Aggregate Structure Identification and its Application to Program Analysis. // In Symp. on Principles of Programming Languages, 1999. Pp. 119 – 132.

[8] J. Newsome, D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. // In Proceedings of the Network and Distributed System Security Symposium (NDSS), 2005.

[9] Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы. Принципы, Технологии и Инструментарий. / Вильямс, 2008 г.

[10] S. Needleman, C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. // Journal of molecular biology. 1970 Mar;48(3):443-53.