

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

На правах рукописи

Шапоренков Дмитрий Александрович

ЭФФЕКТИВНЫЕ МЕТОДЫ ИНДЕКСИРОВАНИЯ ДАННЫХ И
ВЫПОЛНЕНИЯ ЗАПРОСОВ В СИСТЕМАХ УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ В ОСНОВНОЙ ПАМЯТИ

05.13.11 — Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание ученой степени
кандидата физико-математических наук

Санкт-Петербург
2006

Работа выполнена на кафедре системного программирования
математико-механического факультета Санкт-Петербургского
государственного университета.

Научный руководитель: доктор физико-математических наук,
профессор Новиков Борис Асенович

Официальные оппоненты: доктор технических наук,
Кузнецов Сергей Дмитриевич

кандидат физико-математических наук
Округин Михаил Борисович

Ведущая организация: Южно-Уральский государственный университет

Защита диссертации состоится “___” _____ 2006 года в __ часов на
заседании диссертационного совета Д212.232.51 по защите диссертаций
на соискание ученой степени доктора наук при Санкт-Петербургском
государственном университете по адресу: 198504, Санкт-Петербург,
Старый Петергоф, Университетский пр., д. 28, математико-механический
факультет Санкт-Петербургского государственного университета.

С диссертацией можно ознакомиться в Научной библиотеке Санкт-
Петербургского государственного университета по адресу: 199034, Санкт-
Петербург, Университетская наб., д. 7/9.

Автореферат разослан “___” _____ 2006 года.

Ученый секретарь
диссертационного совета
доктор физико-математических наук,
профессор

Б.К.Мартыненко

Общая характеристика работы

Актуальность темы

Системы управления базами данных (СУБД) уже на протяжении нескольких десятилетий являются неотъемлемой частью человеческой деятельности. Эффективность работы практически всех крупных предприятий, организаций, банковских учреждений зависит от эффективности организации процесса обработки больших объемов информации, и СУБД являются важнейшей частью этого процесса. Обеспечивая надежное хранение информации и быстрое получение ответов на интересующие пользователей запросы, СУБД играют роль 'ядра', на основе которого строятся информационные системы, предоставляющие удобный для человека пользовательский интерфейс или позволяющие обмениваться данными с другими системами.

СУБД в ее классическом понимании хранит данные на постоянном носителе информации - жестком диске. Жесткий диск является в определенном смысле 'компромиссным' устройством хранения информации, поскольку, с одной стороны, имеет достаточно высокую (по сравнению с такими носителями информации как компакт-диски и магнитная лента) скорость доступа к информации, а с другой стороны не теряет записанную информацию при выключении питания [1]. Эти свойства сделали диск самым популярным устройством хранения информации, к тому же объем жестких дисков постоянно растет и к настоящему времени (2006 год) измеряется уже терабайтами, а стоимость единицы объема уменьшается. Все это приводит к тому, что хранение информации на жестких дисках является дешевым решением, доступным любым организациям. Кроме того, существует специальное аппаратное обеспечение, такое как RAID-диски, также основанное на жестких дисках, но обеспечивающее повышенную отказоустойчивость за счет хранения избыточной информации [1, 3].

Однако, в отличие от объема жестких дисков, который за последние десятилетия увеличился в тысячи раз, скорость доступа к данным на

жестких дисках не прогрессировала столь стремительно и увеличилась лишь в десятки раз [1, 7]. Это приводит к тому, что жесткий диск становится ‘узким местом’ современных вычислительных систем [7], поскольку за тот же период времени скорость работы процессора (CPU) претерпела еще более впечатляющие улучшения, чем объем жестких дисков [7]. Поэтому выполнение операции ввода-вывода приводит к тому, что процессор вынужден ожидать необходимых для работы данных от жесткого диска (так называемый ‘простой’ процессора). СУБД и другие программы используют буферизацию данных в основной (оперативной) памяти для минимизации негативного эффекта этих ‘простоев’ [3, 1].

Подобно объему жестких дисков и скорости работы процессора, объем оперативной памяти также увеличился в тысячи раз за последние десятилетия. Это позволяет поместить базу данных среднего размера целиком в основную память и приводит к идее *СУБД в оперативной (основной) памяти, (СУБД-ОП)*, являющихся предметом исследования в данной работе. В СУБД-ОП все данные и вспомогательные структуры, такие как индексы, находятся в основной памяти, поэтому жесткий диск для доступа к ним не используется. С учетом того, что скорость доступа к основной памяти на несколько порядков выше, чем скорость доступа к данным на жестком диске [7, 1], это позволяет надеяться на значительное ускорение обработки данных.

Однако традиционные СУБД имеют сложившуюся архитектуру, рассчитанную на хранение данных на медленном носителе - жестком диске [3, 1, 2] и направленную на минимизацию числа обращений к нему. Оказывается, что такая архитектура не является оптимальной для СУБД-ОП [5, 9, 4, 8] и требует пересмотра. Например, традиционные индексные структуры - В+-деревья, параметризованные размером дисковых страниц, являются недостаточно эффективными для СУБД-ОП [9]. Скорость доступа к данным в основной памяти намного выше, чем на диске, но все еще на много порядков ниже, чем скорость работы процессора [7, 5], и, подобно технике буферизации дисковых страниц, в современных компьютерах используется быстрая *кэш-память* [11], работающая со скоростью, почти эквивалентной скорости самого

процессора, но имеющая очень ограниченный объем (порядка нескольких мегабайт). Для уменьшения ‘простоев’ процессора программа (в частности, СУБД-ОП) должна стремиться как можно больше активно используемых данных уместить в кэш-памяти [7, 5, 9].

Цели работы

Данная работа исследует некоторые алгоритмы выполнения реляционных операций и индексных структур в контексте СУБД-ОП. Цели, преследуемые работой, включают:

- В предположении хранения всех данных в основной памяти, определение источников ‘простоев’ процессора в программе, оказывающих негативное влияние на производительность СУБД-ОП.
- Построение простой модели ‘локальности’ обращений к памяти в программе, показывающей, насколько хорошо для данной программы будет работать кэширование часто используемых данных (или, другими словами, насколько эффективно программа может использовать кэш-память процессора).
- Исследование возможностей предвычисления результатов операции естественного соединения (как одной из наиболее распространенных в СУБД) и хранения их в виде специальной структуры.
- Поиск эффективных алгоритмов выполнения операций соединения по предикату над множественнозначными атрибутами.

Основные результаты

В работе получены следующие основные результаты:

- На основе простых экспериментов продемонстрировано, что эффективное использование кэш-памяти процессора играет решающую роль в программах, включающих интенсивный обмен данными между процессором и основной памятью.

- Предложены критерии оценки эффективности структур данных и алгоритмов для СУБД-ОП, учитывающие неоднородность доступа к основной памяти в современных вычислительных системах.
- Построена классификация, обобщающая большинство известных методов оптимизации структур данных и алгоритмов для эффективного использования кэш-памяти процессора. Эта классификация обобщает предшествующие работы и сводит методы оптимизации использования кэш-памяти к нескольким простым идеям.
- Предложен и экспериментально проверен метод оптимизации операции соединения на основе мульти-индексов в СУБД-ОП, показана его сравнительная эффективность по сравнению с методом хэш-соединения в зависимости от параметров входных отношений.
- Исследованы различные алгоритмы для выполнения операции соединения по предикату над множественнозначными атрибутами в СУБД-ОП, предложена модификация алгоритма, основанного на использовании инвертированных файлов, для лучшего использования кэш-памяти и экспериментально продемонстрировано, что эта модификация приводит к уменьшению времени выполнения соединения.

Научная новизна

В данной работе впервые детально исследованы операции соединения по предикатам над множественнозначными атрибутами в контексте СУБД-ОП, с учетом такого фактора как неоднородность доступа к памяти. Остальные полученные результаты также являются новыми и дополняют результаты предшествующих работ.

Практическая и теоретическая ценность

С теоретической точки зрения, в работе предложена простая модель локальности ссылок на данные в программе, позволяющая быстро оценить, насколько эффективно для ссылки будет работать кэширование. Кроме того, в работе получены аналитические оценки эффективности нескольких алгоритмов на основе инвертированных списков для выполнения операции соединения по предикатам над множественнозначными атрибутами.

Практическая ценность работы состоит в том, что предложенные алгоритмы могут быть использованы при построении эффективной реализации СУБД-ОП. Экспериментальные результаты, содержащиеся в работе, могут послужить основой для выбора наиболее подходящего алгоритма для выполнения соединения по предикату над множественнозначными атрибутами в зависимости от параметров отношений, что важно, в частности, для создания оптимизатора запросов для СУБД-ОП.

Апробация работы

Результаты работы докладывались

- на Девятой Восточно-Европейской конференции "Advances in Databases and Information Systems" (Таллин, Эстония, сентябрь 2005)
- на Шестой конференции "Baltic Conference on Databases and Information Systems" (Рига, Латвия, июнь 2004)
- на Первом и Втором коллоквиумах "Spring Colloquium for Young Researchers in Databases and Information Systems (SYRCoDIS)" (Санкт-Петербург, май 2004 и июнь 2005)
- на семинарах группы теории баз данных при лаборатории исследования операций НИИММ

Публикации

Основные результаты представлены в работах [1]-[4].

Структура и объем диссертации

Диссертация состоит из введения, 3 глав, заключения и списка литературы. Основной текст диссертации занимает 120 страниц машинописного текста. Библиография содержит 67 наименований. Общий объем диссертации 128 страниц. Рисунки и таблицы нумеруются по главам.

Содержание работы

Введение является **первой главой** и содержит предварительную информацию о предмете исследования.

Вторая глава демонстрирует актуальность СУБД-ОП. В ней содержится краткое изложение особенностей архитектуры современных компьютеров, оказывающих существенное влияние на эффективность программ. Большая разница в скорости работы процессора и оперативной памяти компенсируется за счет нескольких уровней кэш-памяти, устанавливаемой внутри процессора. Однако, эффективное использование кэш-памяти, как и *буфера трансляции адресов*, ускоряющего преобразование виртуальных адресов в физические, является обязанностью программы. Это иллюстрируется примером простой программы, выполняющей доступ к элементам большого массива с определенным шагом. От величины шага зависит количество кэш- и TLB-промахов и, тем самым, время выполнения программы.

Вторая часть второй главы посвящена обзору архитектуры традиционных СУБД и выяснению того, насколько пригодна эта архитектура для СУБД-ОП, в предположении того, что все данные и вспомогательные структуры находятся в основной памяти. Выясняется, что наиболее существенные изменения должны претерпеть оптимизатор запросов и индексные структуры. Эти изменения связаны с тем, что

оптимизатор запросов в традиционных СУБД обычно выбирает план, доставляющий минимум функции стоимости, которая включает в себя вычислительную стоимость операции (в терминах количества каких-либо примитивных операций) и количество операций ввода-вывода. В СУБД-ОП второй компонент функции стоимости заменяется на количество ‘простоев’ процессора из-за кэш-промахов, ошибок предсказания перехода и других причин, поэтому для одного и того же объема входных данных оптимизатор в традиционной СУБД и СУБД-ОП могут считать наилучшими разные планы. Аналогично, индексные структуры в традиционных СУБД-ОП стараются минимизировать количество операций ввода-вывода во время поиска (как, например, В+-дерево), и их устройство и параметры (например, размер узла В+-дерева) выбираются из этих соображений. Но такой выбор зачастую оказывается неоптимальным в СУБД-ОП, где главной задачей является минимизация числа кэш- и TLB-промахов - в частности, большие узлы ‘дисковых’ В+-деревьев приводят в СУБД-ОП к большому количеству кэш-промахов во время поиска внутри узла [9, 6, 10].

В **третьей главе** рассматриваются общие методы оптимизации использования кэш-памяти в алгоритмах и структурах данных. Предложенная классификация основана на обобщении приемов, встречающихся в более ранних работах. Отдельно рассматривается оптимизация структур данных и алгоритмов. Для структур данных рассматривается простая модель, где все структуры данных состоят из *узлов*, которые включают поля двух видов - *ключевые поля* и *указатели*. В ключевых полях содержится информация, относящаяся к хранимым данным, а указатели необходимы для поддержания связей в самой структуре данных. Методы оптимизации структур данных удаляют неиспользуемые для поиска ключевые поля из узла, группируют поля таким образом, чтобы совместно используемые поля находились рядом в узле, используют сжатие для уменьшения размера ключевых полей и удаляют указатели из узла, заменяя их на вычисление адресов дочерних узлов за счет специального устройства структуры данных. Можно также пытаться улучшать взаимное расположение узлов, располагая на нечетных

уровнях дочерние узлы в том же кэш-блоке, что и родительский узел, если количество дочерних узлов у данного узла невелико.

В разделе третьей главы, посвященном оптимизации алгоритмов, используется модель *локальности* ссылок, которая характеризует *темпоральную* и *пространственную* локальность инструкции обращения к памяти (ссылки) в программе. Пространственная локальность выражается с помощью *пространственного интервала*, который равен среднему расстоянию между адресами двух последовательных исполнений инструкции обращения к памяти. *Интервал переиспользования*, характеризующий темпоральную локальность, равен среднему интервалу времени между исполнениями ссылки, которые попадают в один и тот же кэш-блок. Мы рассматриваем также зависимость между адресами последовательных исполнений ссылки - в некоторых случаях можно сказать (с точностью до нескольких возможных значений), чему будет равен адрес очередного исполнения ссылки, зная адрес текущего исполнения. Эта зависимость моделирует списки, деревья и другие подобные структуры, где узел содержит указатели на следующие узлы. Введенные понятия позволяют более формально определить *последовательный*, *произвольный* и *псевдо-произвольный* характер доступа к памяти, что используется в данной и следующей главах.

Методы оптимизации алгоритмов, рассматриваемые в третьей главе, подразделяются на методы, уменьшающие пространственный интервал (т.е. улучшающие пространственную локальность), и методы, уменьшающие интервал переиспользования. Среди методов первого вида *введение временных структур данных*, которые более компактны, чем исходные, за счет того, что содержат лишь данные, требуемые в конкретном алгоритме, и *изменение порядка обхода* структуры данных, если порядок обработки узлов не имеет значения (как, например, в случае многих операций над матрицами, которые можно выполнять как 'по строкам', так и 'по столбцам'). Методы, уменьшающие интервал переиспользования, повышают вероятность того, что активно используемые данные все еще находятся в кэш-памяти к моменту их следующего использования. *Разбиение структур данных на блоки*

применимо при последовательном характере доступа и уменьшает размер активного множества данных (и, следовательно, интервал переиспользования элемента этого множества) за счет обработки структуры данных блоками фиксированного размера. *Распределение структур данных* выполняет предварительный проход структур данных с целью распределения их по разделам в соответствии с некоторой хэш-функцией. Затем исходный алгоритм применяется к каждому разделу, что обеспечивает интервал переиспользования, не превышающий размер раздела. *Логическое распределение* не генерирует разделы в явном виде, но заменяет их на многократный проход по структуре данных, обрабатывая на каждой итерации лишь элементы, относящиеся к соответствующему разделу. Рассматривается также использование инструкций процессора для *явной предвыборки* в кэш-память, что возможно при псевдопроизвольном методе доступа, поскольку на очередной итерации известен возможный адрес следующего узла.

Четвертая глава посвящена алгоритмам для выполнения операций естественного соединения и соединения по предикатам над множественнозначными атрибутами. Для операции естественного соединения рассматриваются *мульти-индексы* - индексы, которые, в отличие от традиционных индексов, используемых в СУБД, строятся по нескольким, а не по одному отношению. Мульти-индекс позволяет за одну операцию поиска находить записи всех индексируемых отношений, имеющие данное значение индексируемого атрибута. При изменении индексируемых отношений мульти-индекс обновляется, а естественное соединение сводится к однократному просмотру мульти-индекса. Для мульти-индекса предлагается двухуровневая структура, на нижнем уровне которой находится стандартная страничная структура для хранения *индексных записей*, а на верхнем уровне - индексная структура, отображающая значения индексируемого атрибута в индексные записи. Возможны различные варианты организации верхнего уровня мульти-индекса, выбор между которыми должен выполняться в соответствии с соображениями частоты обновления индексируемых отношений, доступного объема основной памяти и т.д. Приводятся

результаты экспериментальной проверки мульти-индекса в среде Memphis, демонстрирующие, что соединение с его использованием выполняется в ряде случаев более эффективно, чем с помощью метода, основанного на хэшировании. Здесь также содержится краткое описание среды Memphis, реализованной в рамках данной работы и использованной при проведении экспериментов в данной главе.

Вторая часть четвертой главы описывает исследование алгоритмов выполнения операции соединения по предикатам над атрибутами, значения которых не атомарны, а являются множеством элементов. Сначала рассматриваются несколько известных алгоритмов, предварительные эксперименты с которыми показывают, что наиболее эффективными являются алгоритмы, основанные на инвертированных списках. Два из этих алгоритмов, IFNL (Inverted File Nested Loops) и IFJ (Inverted File Join), являются предметом улучшения для СУБД-ОП. К этим алгоритмам применяются методы, описанные в третьей главе, в частности, метод логического распределения. Эта оптимизация оказывается эффективной для алгоритма IFJ, но не IFNL, что подтверждается приведенными экспериментальными результатами и объясняется аналитически. В результате данного исследования получен алгоритм IFJ(k), параметризуемый числом разделов - просмотров инвертированного файла. Дальнейшие эксперименты исследуют этот алгоритм: выясняется, как ведет себя алгоритм в зависимости от числа разделов и как выбирать оптимальное число разделов; изучается эффект сжатия инвертированных списков на поведение алгоритма; алгоритм сравнивается с другими известными алгоритмами и демонстрируется его преимущество. Данный алгоритм показывает лучшую скорость работы, чем конкурирующие алгоритмы, и лучше масштабируется с ростом размера входных отношений. Дополнительную экономию памяти может обеспечить применение сжатия инвертированных списков.

Заключение содержит список основных результатов, полученных в работе.

Список литературы

- [1] *Гектор Гарсиа-Молина, Джеффри Д. Ульман, Дженнифер Уидом.* Системы баз данных. Полный курс. — Вильямс, 2003.
- [2] *М.Р.Когаловский.* Энциклопедия технологий баз данных. — Финансы и статистика, 2002.
- [3] *К.Дж.Дейт.* Введение в системы баз данных. 8-е издание. — Вильямс, 2005.
- [4] Dalí: A High Performance Main Memory Storage Manager. / Н. V. Jagadish, D. F. Lieuwen, R. Rastogi et al. // Proc. of the VLDB 1994 Conference. — Morgan Kaufmann, 1994. — P. 48–59.
- [5] DBMSs on a Modern Processor: Where Does Time Go? / A. Ailamaki, D. J. DeWitt, M. D. Hill, D. A. Wood // Proc. of the VLDB 1999 Conference. — Morgan Kaufmann, 1999. — P. 266–277.
- [6] *Hankins R. A., Patel J. M.* Effect of Node Size on The Performance of Cache-Conscious B⁺-trees. // Proc. of the SIGMETRICS 2003 Conference. — ACM, 2003. — P. 283–294.
- [7] *Hennessy J. L., Patterson D. A.* Computer Architecture: A Quantitative Approach. — 3rd edition. — Morgan Kaufmann publishers, 2002.
- [8] *Lehman T. J., Carey M. J.* A Study of Index Structures for Main Memory Database Management Systems. // Proc. of the VLDB 1986 Conference. — Morgan Kaufmann, 1986. — P. 294–303.
- [9] *Rao J., Ross K. A.* Cache Conscious Indexing for Decision-Support in Main Memory. // Proc. of the VLDB 1999 Conference. — Morgan Kaufmann, 1999. — P. 78–89.
- [10] *Rao J., Ross K. A.* Making B⁺-Trees Cache Conscious in Main Memory. // Proc. of the SIGMOD 2000 Conference. — ACM, 2000. — P. 475–486.

- [11] *Smith A. J.* Cache Memories. // *ACM Comput. Surv.* — 1982. — Vol. 14, no. 3. — P. 473–530.

Работы автора по теме диссертации

1. D.Shaporenkov. Multi-Indices - A Tool for Optimizing Join Processing in Main Memory. *Proceedings of the Sixth International Baltic Conference on Databases and Information Systems (Doctoral Consortium), Vol. 2* Riga, Latvia, 2004. - P. 105-114.
2. D.Shaporenkov. Performance Comparison of Main-Memory Algorithms for Set Containment Joins. *Proceedings of the First Spring Colloquium for Young Researchers in Databases and Information (SYRCoDIS)*, Saint-Petersburg, Russia, 2004. - P. 17-21.
3. D.Shaporenkov. Partitioning Inverted Lists for Efficient Evaluation of Set-Containment Joins in Main Memory. *Proceedings of the Second Spring Colloquium for Young Researchers in Databases and Information (SYRCoDIS)*, Saint-Petersburg, Russia, 2005. - P. 40-46.
4. D.Shaporenkov. Efficient Main-Memory Algorithms for Set Containment Join Using Inverted Lists. *Proceedings of the Ninth East-European Conference on Advances in Databases and Information Systems*, Tallinn, Estonia, 2005. - P. 139-152.