САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет
кафедра системного программирования

Стоимостная модель оптимизации вычисления XPath-выражений

Дипломная работа студента 5 курса
Богатова Алексея Олеговича

Научный руководитель	проф. Новиков Б.А.
Рецензент	Лукичёв М.С.
Допустить к защите: Зав. кафедрой	проф. Терехов А.Н.

Санкт-Петербург
2007 г.

Оглавление

1 Введение

1.1 Оптимизация запросов

Применяемые в настоящее время методы оптимизации запросов к базам данных описаны в статье Янниса Иоаннидиса *Query Optimization* [7]. Там рассматривается прежде всего случай реляционных СУБД, но приведенная в схема оптимизатора во многом применима и к прирожденным (native) XML СУБД.

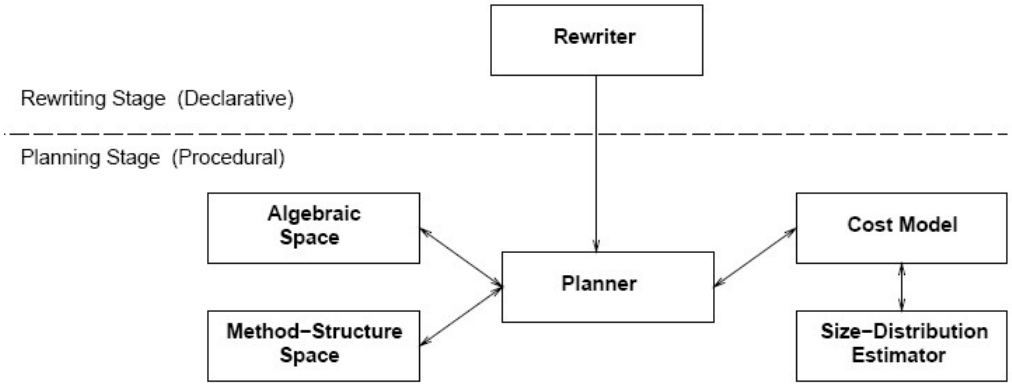

Рис. 1: Общая схема работы оптимизатора запросов

Рассмотрим схему подробнее. Модуль Rewriter (Переписывающий) осуществляет преобразование запроса в эквивалентный на декларативном уровне. Модуль Planner (Планировщик) является ключевым в системе. Он производит поиск оптимального плана запроса в Алгебраическом пространстве (Algebraic Space) и Пространстве методов и структур (Method-Structure Space), используя модель стоимости (Cost Model) и статистические данные, или оценки размера и распределений (последний модуль в статье назван Size-Distribution Estimator). Его цель – выбрать из пространства семантически эквивалентных планов оптимальный по стоимости, то есть тот, который предположительно будет выполняться быстрее остальных. Запрос представляется в виде выражения из алгебраических операций (в случае реляционной модели – это реляционная алгебра), и Алгебраическое пространство порождается преобразованиями подобного выражения. Пространство методов и структур соответствует различным физическим методам выполнения операции и применяемым в них структурам данных. Это могут быть, к примеру, последовательное сканирование таблицы или обращение к индексу СУБД. Модель стоимости служит для оценки стоимости исполнения запроса. Она задает стоимость выполнения различных физических операций в виде некоторых формул. В формулы подставляются статистические оценки: о хранимых данных собирается (возможно, единожды; возможно, с некоторой периодичностью; возможно, при каждом новом запросе) статистика, позволяющая оценить время выполнения некоторой операции

на этих данных. Например, можно хранить число записей, имеющих определенное значение заданного атрибута.

Таким образом, качество работы оптимизатора в немалой степени зависит от стоимостной модели. Ясно, что всегда можно оценить стоимость выполнения запроса абсолютно точно – просто выполнив запрос; но это было бы странно – стоимость необходимо знать для выбора оптимального плана до его исполнения. Стоимостная модель основывается на компромиссе между точностью оценок и скоростью вычисления стоимости.

1.2 Специфика XML СУБД

В настоящее время среди технологий баз данных появилась новая модель данных XML [4], а также определённый над этой моделью язык запросов XQuery [26]. Эта модель данных и язык запросов предназначены для управления XML-данными [27]. Построение эффективных систем управления XML-данными невозможно без разработки развитых средств оптимизации запросов.

СУБД, способные работать с XML, бывают двух типов. СУБД первого типа хранят данные в виде отношений реляционной модели, и запросы преобразуются в выражения реляционной алгебры, для которых существуют достаточно качественные оптимизаторы. В системах второго типа – прирожденных (native) XML СУБД – хранение данных основано на документах XML. В большинстве таких систем оптимизация запросов ограничивается применением некоторых правил перезаписи запроса, заведомо улучшающих время выполнения запроса, например, опускание предиката с целью уменьшения размера промежуточных результатов. При такой реализации не анализируется алгебраическое пространство и оптимальные планы выполнения запроса не рассматриваются.

Мы проводим данное исследование в рамках программного проекта XAnswer [22]. Это исследовательский проект прирожденной XML СУБД, в рамках которого в группе СПбГУ проводятся исследования на темы, так или иначе связанные с запросами к XML-документам.

Чтобы иметь возможность применить для оптимизации запросов в прирожденных XML СУБД схему, аналогичную описанной выше, необходимо ввести алгебру операций с достаточно хорошими свойствами, удобную для оптимизации запросов на языке XQuery и, в частности, навигационных выражений. В проекте XAnswer используется одноименная (XAnswer) алгебра, базирующаяся на алгебрах Xtasy [14] и XAT [19]. Промежуточные результаты выполнения запросов хранятся в специальной структуре под названием Envelop, которая представляет собой отношение на значениях переменных. В случае XPath-выражения переменная соответствует шагу выражения, ее значения – это соответствующие шагу узлы документа.

XAnswer использует многие структуры хранения и индексирования из СУБД eXist (http://exist.sourceforge.net/). В eXist существует индекс коллекций, индекс всех узлов, индекс элементов (из которого за один просмотр можно выбрать элементы с заданным именем) и индекс слов.

СУБД eXist использует схему нумерации Dynamic Level Number (DLN). Номер узла в ней – это последовательность чисел, разделенных точками. Длина последовательности соответствует уровню узла в дереве (для корня 1; для ребенка на 1 больше, чем для родителя). Пример: 1.2.2. Добавление узлов также поддерживается этой схемой: вместо числа может стоять составной индекс из чисел, разделенных слешем. Так, 1.2.1/1 – это узел того же уровня, что и 1.2.2. Очевидно, зная DLN-номер узла, мы можем проверить, является ли узел родителем либо ребенком заданного узла, за время $O(1)$.

1.3 Модель данных XML и язык XPath

Язык XQuery основан на модели данных XML [4]. Эта модель определяет абстрактное состояние одного или нескольких документов или их фрагментов. Основополагающим понятием этой модели является понятие последовательности. Последовательность (sequence) – упорядоченный набор нулевого или большего числа элементов. Элемент (item) может быть узлом или атомарным значением. Атомарное значение (atomic value) – экземпляр одного из встроенных типов данных, определенных во второй части описания "XML Schema"[24], таких как строки, целые и десятичные числа и даты. Узел (node) соответствует одному из шести видов: элементы, атрибуты, тексты, документы, комментарии и команды обработки. Узел, относящийся к виду «элементы», может иметь другие узлы в качестве потомков, что позволяет организовывать иерархию узлов. Элементы и атрибуты имеют имена и типизированные значения. Типизированные значения (typed value) – это последовательность из нуля или большего числа атомарных значений. Для всех узлов иерархии имеется глобальный порядок, называемый порядком документа (document order), в соответствии с которым каждый узел предшествует своему потомку и правому брату в XML дереве. Порядок документа соответствует порядку, в котором следовали бы узлы, если бы иерархия узлов представлялась в формате XML. Порядок документа между узлами в разных иерархиях определяется в реализации, но он должен быть последовательным, т.е. все узлы одной иерархии должны располагаться либо до, либо после всех узлов в другой иерархии. Глобальный порядок документа и локальный порядок в некоторой последовательности могут быть различными. Последовательности могут быть неоднородными, т.е. могут содержать смесь узлов и атомарных значений разного типа. Однако последовательность не может быть элементом другой последовательности. Все операции, создающие последовательность, определены так, что результат операции – одноуровневая последовательность. Не проводится различия между элементом и последовательностью единичной длины, т.е. узел или атомарное значение считаются идентичными последовательности единичной длины, содержащей этот узел или атомарное значение.

Допускаются последовательности нулевой длины, иногда они используются для представления отсутствующей или неизвестной информации.

Помимо последовательности в модели данных XML определяется специальное значение – значение ошибки (error value), которое является результатом вычисления

выражения, содержащего ошибку. Значение ошибки не может присутствовать в последовательности вместе с каким-либо другим значением.

Навигационное выражение в XPath [23] состоит из серии шагов, разделённых символами "/"или "//". Значением навигационного выражения является последовательность узлов, которая формируется на последнем шаге. Результат каждого шага – последовательность узлов. В результат шага по оси child ("/") попадают дети узлов из результата предыдущего шага, имеющие заданный тэг. В результат шага по оси descendant-or-self ("//") включаются потомки узлов из результата предыдущего шага, а также сами эти узлы, при условии, что они имеют заданный шагом тэг.

Всего в XPath 13 осей (ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self), но мы ограничимся рассмотрением только двух упомянутых выше.

1.4 Цель работы

Цель работы заключается в следующем:

- составить представление об известных методах стоимостной оптимизации навигационных выражений;
- реализовать некоторые методы исполнения навигационных выражений, добавить статистические структуры, отсутствующие в XAnswer;
- разработать и реализовать стоимостную модель для навигационных выражений с осями child и descendant-or-self в XAnswer;
- провести эксперименты для оценки качества стоимостной модели.

2 Обзор близких работ

Впервые серьёзное внимание оптимизации запросов при построении систем реляционных баз данных было уделено в проекте System R [17, 16]. В 1996 году была написана обзорная статья Я. Иоаннидиса [7], посвященная оптимизации запросов.

В этом обзоре мы сосредоточимся на стоимостной оптимизации вычисления навигационных выражений XPath и особое внимание уделим стоимостной модели.

В работе [5] впервые вводится статистическая структура, позволяющая объединять кратные пути. Она носит имя DataGuide, или путеводитель по данным. Изначально структуру предлагалось использовать для объектной модели, СУБД Lore и языка запросов Lorel. Путеводитель для объектной БД может оказаться и не деревом. На этой структуре базируются и дерево навигационных выражений, или PathTree ([1]), и описывающая схема в СУБД Sedna [15].

В статье [1] вводятся две структуры: Path Trees и Markov Tables. Деревья навигационных выражений – это приспособленные к XML путеводители по данным. Идея марковских таблиц – в том, чтобы хранить селективности коротких путей, а селективности длинных вычислять исходя из них, с умножением и делением хранимых селективностей. По сути получится марковский процесс, учитывающий историю некоторой наперед заданной длины. Эти статистические структуры важны для нас, и позже мы рассмотрим их подробнее.

Авторы статьи [25], представленной на конференции VLDB [18] в 2002 году, применяют онлайновый метод XPathLearner для сбора XML-статистики – то есть статистика собирается лишь при исполнении запросов. Достоинства подхода в том, что он позволяет не сканировать весь документ и не хранить такую статистику, которую ни один запрос не использует. Кроме того, метод приспособлен к работе с обновлениями: статистика изменяется с изменением данных. Запросы, с которыми работают авторы – как простые навигационные выражения, так и выражения со строковыми значениями.

Три года спустя те же авторы создали более сильный онлайновый метод [8]. В корзине хранятся какие-то запросы, обладающие определенными значениями признаков. Для вновь прибывающего запроса считается условная вероятность: в какую корзину он попадет с такими значениями признаков, которые у него есть. Далее алгоритм выполняет запрос, и если он ошибся в предсказании корзины, то распределение признаков для корзины, куда запрос реально должен попасть, обновляется.

Алгоритмы, предлагаемые в статье [9], направлены на выполнение twig queries – запросов, представимых в виде древовидного образца, дуги которого – parent-child либо ancestor-descendant, при этом в качестве выходного результата может потребоваться любая вершина. Есть два подхода: structural joins и holistic twig joins. Поэтому авторы остановились на структурных соединениях, точнее – на их физической реализации с использованием хеширования.

Хороший алгоритм хеширующего соединения должен быть однопроходным, избегать действий по удалению дубликатов и сортировке, позволять хэшировать любую из двух последовательностей. Рассмотрены 12 алгоритмов: ChildHashA, ChildHashB, ParHashA, ParHashB, AncHashA, AncHashB, DescHashA, DescHashB, ChildFullHashA, ChildFullHashB, DescFullHashA, DescFullHashB. Они различаются осью (parent-child или ancestor-descendant), а также тем, какую из двух последовательностей хэшировать и что возвращать. О многих из них мы расскажем подробнее в основной части работы.

Работа [20] рассматривает проблему соединения по оси AD двух множеств, описываемых при помощи предикатов. Авторы выделяют предикаты двух видов: «элемент имеет такой-то тэг» и «элемент имеет такое-то содержимое». Результаты соединения оцениваются при помощи двумерных позиционных гистограмм (такая

гистограмма строится для конкретного предиката). По горизонтальной оси откладывается номер элемента при префиксном обходе, по вертикальной некоторый аналог постфиксного номера. (Напомним, префиксный обход дерева производится в порядке «корень – поддеревья слева направо», а постфиксный обход – в порядке «поддеревья слева направо – корень».) Таким образом, предки располагаются слева сверху, потомки – справа снизу. Известно, что ни для какого узла нет точек справа снизу и слева сверху от него. Исходя из этого, оцениваются вероятности попадания в каждую клетку точки, удовлетворяющей соединению предикатов. Всего для оценки селективности для одного операнда проводится суммирование по всем клеткам оценок, связывающих клетку гистограммы этого операнда с гистограммой другого операнда.

Число непустых клеток в позиционной гистограмме зависит от ее размера линейно (а не квадратично). Алгоритм объединения трехпроходный; можно предварительно посчитать вспомогательные данные на гистограмме для каждого предиката, что заметно ускорит его работу.

Статья [21] рассматривает запросы, представимые в виде дерева (с осями child и descendant, с предикатами ветвлениями). Запрос такого вида выполняется структурными соединениями. Физическая реализация соединения в статье во внимание не принимается, обсуждается только порядок выполнения операций соединения. Предлагается 5 методов, первые два из которых гарантированно находят оптимум (по функции стоимости): собственно метод динамического программирования и его модификации, отсекающие разные классы планов.

Работа [10] от 1999 года – одна из первых, посвящена стоимостной оптимизации выполнения навигационных выражений для XML. Стандартов XPath / XQuery тогда еще не существовало, исследование проводилось для СУБД Lore. Если проводить аналогию с XML СУБД, в Lore 3 индекса: по всем значениям, по всем тэгам и по тэгам детей узла. Кроме того, есть путеводитель по данным. Перебираются различные способы преобразования логического плана в физический. Две основные операции логической алгебры – выбор элементов по тэгу и структурное соединение. Каждый шаг навигационного выражения (связывающий еще какие-то переменные) можно выполнять с использованием какого-либо индекса, DataGuide или простого спуска по дереву. Хранится (для каждого простого пути ограниченной длины) следующая статистика: количество достижимых по этому пути узлов, количество детей таких узлов для каждого тэга ребенка, количество родителей таких узлов для каждого тэга родителя. Она используется для оценки селективности очередного шага вниз или вверх. Модель стоимости основана в большей степени на обращении к диску, чем на времени выполнения; используются, например, селективности, а также оценки обращений к индексам. Алгоритм выбора плана жадный: для логической операции однократно выбирается оптимальный физический план. Как говорят эксперименты, работает метод хорошо, но не всегда правильно оценивает распределение значений (гистограммы не использовались).

Статья [2] описывает разработки методов стоимостной оптимизации для XML, которые ведутся в университете Кайзерслаутерна. Исследования авторы проводят на базе собственной XML СУБД, названной XML Transaction Coordinator. Документы хранятся в B^*-деревьях. Для XPath-выражений рассматривается дерево структурных соединений, вытянутое в одну сторону (не предпринимается никаких попыток сделать его кустистым), и для каждого шага выбирается оптимальный метод соединения (из таких методов, как hash joins и nested loops). Собираемая статистика – количество детей и родителей узла, имеющих различные тэги (например, детей элемента book с тэгом title – 150), структура хранения статистики носит название An XML Structural Statistics (AXS-Stat).

Диссертация (master thesis) [12], защищенная в 2003 году в Universität Konstanz, посвящена проблемам стоимостной оптимизации XPath в СУБД MonetDB [11]. Это так называемая main memory database – данные, к которым происходят запросы, хранятся в оперативной памяти. Ключевая структура хранения в MonetDB – бинарные ассоциативные таблицы (binary associative tables, BAT), хранящие отношения двух атрибутов. Любое отношение с бóльшим числом атрибутов формируется как результат соединения. BAT практичны, поскольку из-за малого размера отдельных записей позволяют активно использовать кэш процессора.

Запросы выполняются при помощи XPath Accelerator [6]. Для выполнения запросов используется нумерация Дитца [3]: каждый узел получает два номера – префиксный и постфиксный. Неравенства на префиксные и постфиксные номера узлов задают разбиение всего документа на 4 части (относительно некоторого зафиксированного узла), а именно parent, child, preceding, following. Это свойство нумерации можно наглядно изобразить на координатной плоскости (по осями при этом будут отложены pre-номер и post-номер). Таким образом, шаг навигационного выражения -– это window query (SELECT для некоторого диапазона значений одного атрибута). Размер окна можно уменьшить, используя другие неравенства, связанные с высотой дерева).

Оценивается селективность шага, примененного к последовательности узлов, или контекстному множеству шага (context set). Для этого в случае разных осей используются самые разнообразные статистические данные: количество узлов с заданным тэгом, среднее количество исходящих дуг для узлов с заданным тэгом; а также префиксный и постфиксный номера узла.

Стоимость выполнения операции принимается зависящей от количества обращений к памяти (memory access cost) и от времени, затрачиваемого процессором на выполнение операции (CPU cost). Для оценки количества обращений к памяти используются знания об алгоритме выполнения операции, кроме того, учитываются обращения к процессорному кэшу (приближенно вычисляется количество элементов, которых не будет в кэше на момент обращения к ним и которые придется получать из оперативной памяти). Для оценки процессорной стоимости необходима калибровка машины. Время, затрачиваемое на операции процессора, берется равным всему времени исполнения запроса, из которого вычли время на обращения к памяти,

и считается линейно зависящим от длины обрабатываемых последовательностей. Калибровка дает коэффициенты этой зависимости.

3 Физические операции и статистические структуры

3.1 Контекст работы: XAnswer

Рассмотрим подробнее схему работы оптимизатора XAnswer.

Запрос XQuery, подаваемый на вход оптимизатору, разбирается и переводится (с помощью W3C parser) в представление W3C AST (абстрактное синтаксическое дерево). Это представление конвертируется в формат XAnswer, т.е. в XAnswer AST. На данном этапе мы имеем дерево из логических операций.

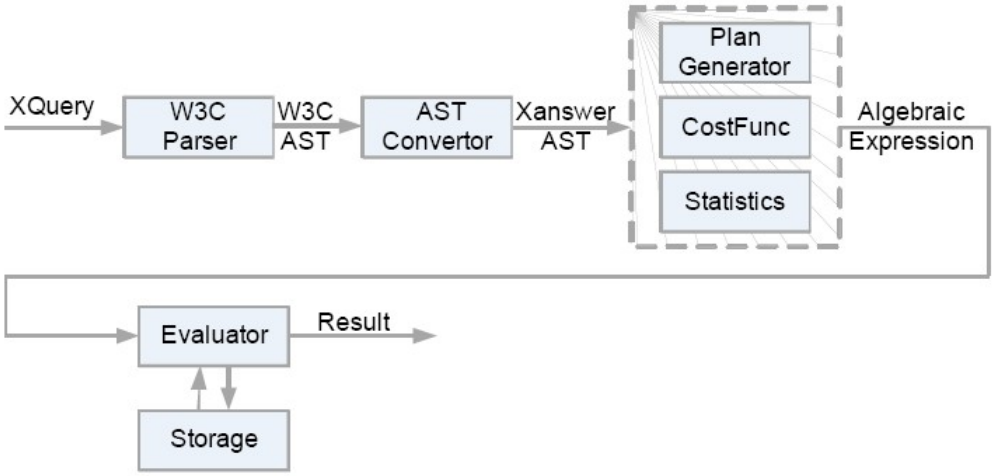

Рис. 2: Схема выполнения запроса в XAnswer

Напомним, что Envelop – это структура, которую можно представить в виде таблицы. Столбцы таблицы соответствуют переменным, строки – кортежам узлов. В ячейке таблицы стоит значение узла XML-документа (часто мы для удобства будем говорить просто «узел»). Ясно, что проекцию Envelop на некоторый атрибут (т.е. один столбец) можно рассматривать как последовательность.

Для нас важны следующие логические операции (все они возвращают Envelop):

1. Операция GetDocumentRoot имеет своим единственным параметром документ и выдает узел, являющийся его корнем.

2. Операция GetElement возвращает узлы из заданного множества документов DocSet, удовлетворяющие заданному условию NodeTest. В простейшем случае это условие на тэг, т.е. возвращаются узлы с заданным тэгом – и тогда будем для удобства писать GetElementByName.

V_A$	V_B$	V_C$	V_D$
a1	*b1*	*c1*	*d1*
a2	*b2*	*c2*	*d2*
an	*bn*	*cn*	*dn*

Рис. 3: Структура Envelop

3. Операция Structural Join – структурное соединение двух последовательностей по некоторой оси. Возвращается новый Envelop, в котором элементы соответствующих столбцов находятся в отношении, определяемом осью. Пример работы StructuralJoin можно видеть на рисунке 4. Здесь происходит соединение по оси child.

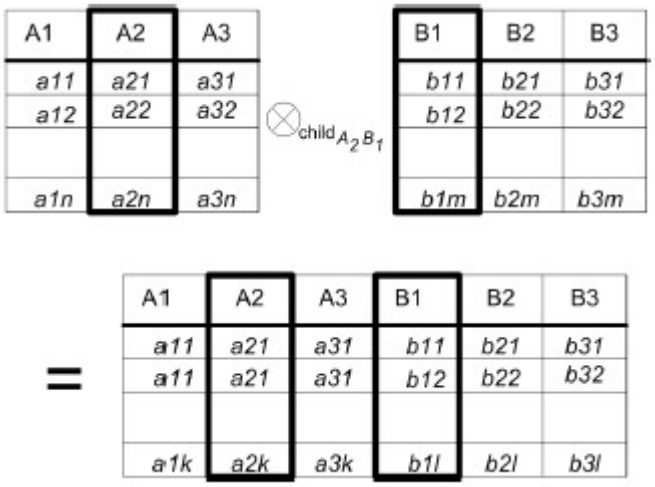

Рис. 4: StructuralJoin: пример

Получив план исполнения запроса, представленный в виде дерева из логических операций, оптимизатор строит различные физические планы, подставляя вместо логических операций соответствующие им физические. За эту фазу работы оптимизатора отвечают 3 модуля: Plan Generator (генератор планов), Cost Function (функция стоимости) и Statistics (модуль статистики). Первый из них генерирует физические планы из логического по некоторому заданному алгоритму. Второй позволяет оценить стоимость выполнения физического плана, используя третий модуль.

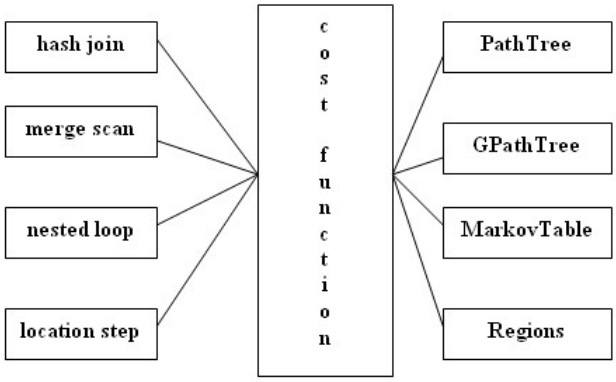

Рис. 5: Место функции стоимости в оптимизаторе

3.2 Навигационные выражения

Мы рассматриваем оптимизацию выполнения навигационных выражений с осями child и descendant-or-self. Такое выражение может быть представлено в виде

$$[x]AxBxCx\ldots,$$

где x – это либо / (ось child, или PC), либо // (ось descendant-or-self, или AD), A, B, C – тэги узлов.

Известны следующие методы исполнения навигационных выражений:

- Up-bottom: выполняется верхний шаг навигационного выражения, получается некоторая последовательность узлов. Для каждого узла из нее (т.е. для растущего из этого узла поддерева) выполняются следующие шаги навигационного выражения, то есть происходит рекурсивный вызов метода.

- Bottom-up: алгоритм начинает с нижнего шага, для каждого узла последовательности рекурсивно выполняется верхняя часть навигационного выражения.

- Структурные соединения: на каждом шаге навигационного выражения берутся две последовательности и соединяются по некоторой оси – PC либо AD. В каком порядке исполняются шаги – заранее неизвестно, это определяется методом динамического программирования. Конкретный план выполнения навигационного выражения с использованием структурных соединений можно наглядно представить в виде дерева (см. рисунок 6).

  Отдельно взятая операция структурного соединения может быть выполнена следующими способами:

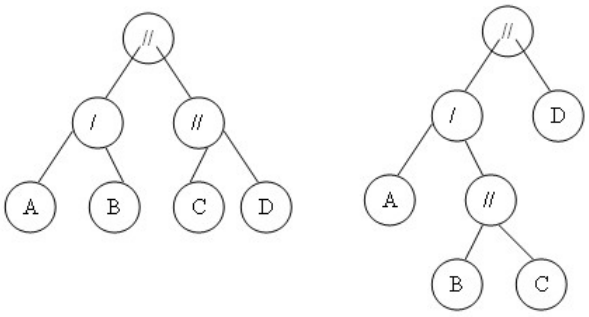

Рис. 6: Структурные соединения –
2 различных плана выполнения XPath-выражения $A/B//C//D$

- Nested loop. Одна последовательность просматривается во внешнем цикле, другая – во внутреннем, и каждая пара узлов проверяется на принадлежность отношению.
- Hash joins. Значения некоторой хеш-функции от узлов одной из последовательностей (или от родителей узлов, или от предков узлов – в зависимости от оси и алгоритма) записываются в хеш-таблицу, после чего другая последовательность просматривается в поисках совпадений значений хеш-функции.
- Merge scan. Обе последовательности сортируются в соответствии с порядком элементов в документе, соединение производится алгоритмом слияния. При этом алгоритм проверяет, упорядочена ли последовательность в порядке документа (document order), и если нет – предварительно производит ее сортировку.

В рамках данной работы реализованы следующие физические методы выполнения структурных соединений.

- Nested loop (метод вложенных циклов), как уже было сказано, просматривает одну последовательность во внешнем цикле, а другую – во внутреннем. При этом любую из двух последовательностей можно рассматривать и как внешнюю (в циклах), и как внутреннюю – это скажется только на направлении проверяемого условия. В любом случае мы проверяем, является ли один узел родителем либо предком другого узла.
- Hash join. Рассматривается выражение вида A/B либо $A//B$, в зависимости от оси (PC или AD), где A и B – последовательности. A будем называть последовательностью потенциальных родителей / предков, B – последовательностью потенциальных детей / потомков соответственно. Реализованы 4 метода.

1. ChildHashA: хешируется последовательность потенциальных родителей, для каждого элемента последовательности потенциальных детей вычисляется хеш-код родителя.
2. ParHashB: для каждого элемента последовательности потенциальных детей хешируется родитель элемента, далее проверяется последовательность потенциальных родителей.
3. DescHashA: сначала хешируется последовательность потенциальных предков. Затем для каждого элемента из последовательности потенциальных потомков считаются значения хеш-функции от всех его предков по дереву и ищутся в хеш-таблице.
4. AncHashB: для каждого элемента последовательности потенциальных потомков хешируются все его предки по дереву, и после этого проверяются значения хеш-функции от элементов последовательности потенциальных предков.

- Merge scan. Метод получает на вход 2 последовательности, отсортированные в порядке документа (document order), и передвигает указатель по одной из них, просматривая при этом другую. По номерам DLN двух узлов легко определить, состоят ли они в отношении parent-child или ancestor-descendant. После того как указатель фиксируется на некотором элементе первой последовательности, алгоритм проходит по второй последовательности и, обнаружив в ней первый удовлетворяющий отношению элемент, движется по ней линейно в поисках последнего элемента, являющегося потомком носителя указателя (и, найдя, смещает указатель на следующий элемент первой последовательности). При этом если выполняется слияние по оси AD, то все промежуточные элементы второй последовательности попадут в результат, а если по оси PC, то требуется дополнительная проверка, состоят ли элементы в отношении parent-child.

3.3 Сбор статистики

В проекте XAnswer реализованы следующие статистические структуры.

1. Path Tree (ДНВ – дерево навигационных выражений) строится по исходному XML-документу так: все пути, состоящие из узлов с одинаковыми тэгами, сливаются в один узел ДНВ, и число объединенных путей записывается в этот узел. Таким образом, каждый узел результирующего дерева соответствует навигационному выражению с шагами только по оси "parent-child", и по записанным в узлах числам легко узнать селективность навигационного выражения.

   Структура впервые предложена в статье [1].

   Пример ДНВ можно видеть на рисунке 7 (данные о селективности на картинке не приведены). В выделенный узел b записывается число 4.

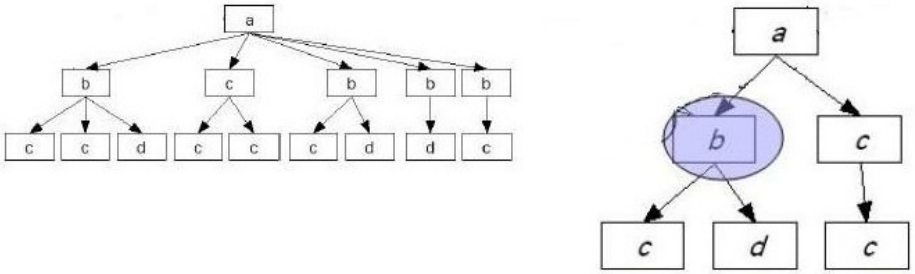

Рис. 7: ДНВ – пример

2. В структуре Generalized Path Tree, общий принцип построения которой такой же, допускаются дуги «ancestor-descendant» – соединяющие не образы узлов, находившихся в исходном дереве в отношении «родитель-ребенок», а образы узлов, находившихся в отношении «предок-потомок».

3. Regions – усовершенствование структуры, предложенной в работе [13].

4. MarkovTable (марковские таблицы): ещё одна статистическая структура из [1].

   В таблице хранится селективность наиболее коротких простых путей (т.е. навигационных выражений с малым числом шагов по оси child). Максимальная длина пути, представленного в таблице, обычно берется равной 2 или 3 – как показали эксперименты авторов [1], при таком значении этого параметра оценки селективности достаточно точны, при его увеличении страдает производительность, а выигрыш по качеству невелик. Для длинных путей число соответствующих пути узлов вычисляется по формуле

$$S(A_1/\ldots/A_n) = S(A_1/\ldots/A_m) \times \prod_{i=1}^{n-m} \frac{S(A_{i+1}/\ldots/A_{i+m})}{S(A_{i+1}/\ldots/A_{i+m-1})}$$

   Простой пример:

$$S(A/B/C/D/E) = S(A/B/C) \cdot \frac{S(B/C/D)}{S(B/C)} \cdot \frac{S(C/D/E)}{S(C/D)}$$

   Идея формулы в том, что длинное навигационное выражение рассматривается как марковский процесс, помнящий историю ограниченной длины.

   В рамках данной работы структура MarkovTable была реализована в XAnswer.

4 Стоимостная модель

4.1 Общий подход

Итак, наша цель – разработать модель стоимости выполнения навигационных выражений. У нас есть логические операции GetDocumentRoot, GetElementBy-

Name и StructuralJoin. Физическую реализацию первых двух операций мы отдаем на откуп структуре хранения eXist и его индексам, считая важным лишь количество элементов, возвращаемых по GetElementByName; и сосредотачиваемся на структурных соединениях. Если навигационное выражение выполнять исключительно структурными соединениями, то на каждый его шаг приходится одна операция GetElementByName; таким образом, эти операции выполнятся в любом плане запроса, а значит, ими действительно можно пренебречь. Для StructuralJoin были перечислены несколько физических операций, соответствующих этой логической. Для каждой из них будет построена формула, оценивающая стоимость. Что касается статистических данных, то с их помощью вычисляются значения селективности, подставляемые в формулы стоимости. Мы введем методы оценки селективности для PathTree и для MarkovTable и сравним вычисление стоимости при использовании этих двух структур.

Примем такое допущение: стоимость выполнения операции зависит только от времени, затрачиваемого на чтение операндов, то есть от их размеров, и ни от чего более. Тогда требуется оценивать размеры последовательностей, получаемых на каждом шаге выполнения навигационного выражения.

Естественно представлять себе навигационное выражение, выполняемое с использованием структурных соединений, в виде дерева (см. рисунок 6), и вычислять стоимость выполнения выражения как стоимость операции в корне дерева, прибавленную к сумме стоимостей двух поддеревьев. Этот подход рекурсивно переносится с корня на остальные узлы дерева.

4.2 Оценка селективности

Напомним, навигационное выражение имеет вид

$$[x]AxBxCx\ldots,$$

где x – это либо / (ось child, или PC), либо // (ось descendant-or-self, или AD), A, B, C – тэги узлов. Поскольку ось, стоящая перед первым тэгом, не вносит существенных изменений в задачу оценки селективности, для простоты изложения опустим ее.

Так как навигационное выражение выполняется структурными соединениями, можно считать, что на каждом шаге правому и левому операндам соответствуют подвыражения исходного выражения, разделенные стоящей в вершине (а в выражении – между ними) осью. Требуется оценить селективность каждого такого подвыражения.

Если используется MarkovTable с максимальной длиной хранимого пути 2, то селективности для простых путей длины 1 ($S(X)$) или 2 ($S(X/Y)$) хранятся в таблице.

Для навигационного выражения длины 2 с осью AD представляется естественным применить суммирование:

$$S(X//Y) = S(X/Y) + \sum_{Z_1} S(X/Z_1/Y) + \ldots + \sum_{Z_1,\ldots,Z_n} S(X/Z_1/\ldots/Z_n/Y),$$

где n – настраиваемый параметр. Интуитивно ясно, что чем больше n, тем дольше проходят вычисления и тем они точнее. Кроме того, выбор n определяется количеством уровней в документе: чем меньше уровней, тем более точные оценки должны получаться при малых значениях n.

Теперь необходимо оценить селективность для путей большей длины. Пусть P_1/P_2 – некоторый путь, $|P_1| > 1,\ |P_2| > 1$. Если действовать по аналогии с формулой для простых путей в MarkovTable, то мы получим оценку

$$S(P_1/P_2) = S(P_1) \cdot S(P_2),$$

которая обычно далека от реальности и значительно превышает истинное значение. Поэтому мы домножим ее на число, меньшее 1, которое возьмем равным

$$\frac{S(A_1/A_2)}{S(A_1)S(A_2)},$$

где $P_1 = \ldots/A_1,\ P_2 = A_2/\ldots$

Получим в результате

$$S(P_1/P_2) = S(P_1) \cdot S(P_2) \cdot \frac{S(A_1/A_2)}{S(A_1)S(A_2)}$$

Формулу можно объяснить следующим образом: есть пути вида P_1 и пути вида P_2. Упорядоченных пар, в которых первый элемент имеет вид P_1, а второй P_2, ровно $S(P_1) \cdot S(P_2)$. Процент же тех пар, в которых первый элемент расположен в дереве непосредственно над вторым, с некоторой степенью точности оценивается процентом подобных пар для путей длины 1.

На основе этой формулы легко построить и формулы «более высокого порядка», то есть

$$S(P_1/P_2) = S(P_1) \cdot S(P_2) \cdot \frac{S(T_1/T_2)}{S(T_1)S(T_2)},$$

где $T_1 = A_1/B_1,\ T_2 = A_2/B_2,\ P_1 = \ldots/A_1/B_1,\ P_2 = A_2/B_2/\ldots$, а $S(T_1/T_2)$ вычисляется тем же методом. И так далее, наращивая длину смежных с границей подпутей. Можно проверить экспериментально, насколько более точную оценку они дают.

Для пути, разделенного на 2 части осью AD, формула получается комбинированием двух уже введенных.

$$S(P_1//P_2) = \sum_{i=1}^{n} \left(\sum_{Z_1,\ldots,Z_i} S(P_1/Z_1/\ldots/Z_n/P_2) \right)$$

Для PathTree мы применим простейший способ оценки селективности. Мы будем по сути выполнять запрос, но не на исходном дереве, а на PathTree. При малом количестве различных тэгов и небольшой высоте дерева размер PathTree существенно меньше размера исходного дерева документа – поэтому для таких документов способ можно считать практичным.

4.3 Вычисление стоимости плана

Теперь оценим собственно стоимость выполнения различных физических операций структурного соединения, считая, что знаем размеры обеих входных последовательностей (оценки этих размеров были построены в предыдущем параграфе). Считаем стоимость как количество считываемых элементов.

1. ChildHash и ParHash. Поскольку обе последовательности, внешняя и внутренняя (обозначим их *outer* и *inner*), просматриваются единожды, и просматриваются независимо друг от друга, получаем

$$|outer| + |inner|.$$

2. DescHash и AncHash. В этом случае не удастся обойтись суммой размеров последовательностей, потому что мы алгоритм обращается еще и ко всем предкам узлов одной из последовательностей. Мы будем считать среднее число предков по всем узлам, имеющим заданный тэг, и использовать для этих целей PathTree. Узлы из последовательности гарантированно имеют один и тот же тэг – но могут быть другие узлы с тем же тэгом, в этом и заключается неточность приближения. Формулу можно записать так:

$$|outer| + \sum_{tag|tag \cap inner \neq \varnothing} (|inner \cap tag| \cdot meanAncNum(tag))$$

3. Nested loop. Для обеих осей имеем

$$|outer| \cdot |inner|,$$

потому что по DLN-номеру как отношение parent-child, так и ancestor-descendant проверяется за $O(1)$.

4. Merge scan. Сначала необходимо отсортировать последовательности по document order – но только в том случае, если мы не уверены, что они отсортированы перед выполнением операции. Известно, что на выходе структурного соединения, выполняемого слиянием, получается отсортированная последовательность – и это надо учитывать. А сортировка (если она нужна для обеих последовательностей) имеет стоимость

$$|outer| \cdot \log |outer| + |inner| \cdot \log |inner|.$$

Понятно, что если одна из последовательностей уже отсортирована, то отвечающее ей слагаемое исчезает, если отсортированы обе – вклад сортировки в стоимость равен нулю.

Далее, для каждого элемента первой последовательности мы просматриваем все элементы второй, являющиеся его потомками (что определяется по DLN-номеру). Значит, осталось оценить размер поддерева для каждого из элементов внешней последовательности – и формула получится следующего вида:

$$\sum_{a \in outer} |subtree(a) \cap inner|$$

Размер поддерева мы возьмем из PathTree, усреднив размеры всех поддеревьев для элементов внешней последовательности. А размер пересечения полученного множества с внешней последовательностью приблизительно оценим так: пусть все элементы последовательности $inner$ лежат также в множестве tag (т.е. имеют такой тэг, причем tag – множество всех элементов с этим тэгом). Тогда $|subtree(a) \cap inner|$ оцениваем как

$$|subtree(a) \cap tag| \cdot \frac{|inner|}{|tag|}.$$

5 Эксперименты

5.1 Описание экспериментов

Эксперименты проводились на машине с процессором Intel Pentium IV 2.4 ГГц и оперативной памятью 512 Мб.

В качестве тестовых данных использовались созданные вручную файлы размером 10 и 20 Кб, файлы с пьесами Шекспира (от 150 до 300 Кб) и сгенерированные файлы размером 500 Кб и 2 Мб (все размеры указаны приблизительно). Во всех файлах количество различных тэгов не превышало 20, а высота дерева – 7, что было необходимо для получения компактных статистических структур, удобных для последующей оценки селективности.

К каждому из тестовых файлов были составлены 2 запроса в виде XPath-выражения из 4 шагов – 2 по оси PC и 2 по оси AD. Каждый шаг как логическая операция структурного соединения мог быть выполнен четырьмя разными способами (для оси parent-child это Nested Loop, Merge Scan, ParHashB и ChildHashA, а для оси descendant-or-self – Nested Loop, Merge Scan, AncHashB и DescHashA). Кроме того, несложно подсчитать, что различных бинарных деревьев с 9 вершинами и 5 листьями существует ровно 14. Бинарное дерево отвечает порядку выполнения операций структурного соединения: в листьях можно поставить тэги, в промежуточных узлах – оси, при этом обход дерева в инфиксном порядке («левое поддерево – корень – правое поддерево») позволяет восстановить исходное XPath-выражение. Итого каждый запрос имел $4^4 \cdot 14 = 3584$ физических планов выполнения. Сначала из $4^4 = 256$ вариантов расстановки физических операций структурного соединения были случайно отобраны 8. Все оставшиеся к данному этапу физические планы (в количестве $8 \cdot 14 = 112$) исполнялись с замером времени выполнения, после чего из них отбирались 8 с тем, чтобы сузить исследуемое пространство. Последняя фаза отбора

была проведена так: множество планов было разбито на 14 непрерывных интервалов по времени исполнения, и из каждого интервала был случайно выбран 1 план (за исключением крайних интервалов, из которых были взяты наихудший и наилучший планы).

По файлам были построены статистические структуры PathTree и MarkovTable. Затем были вычислены стоимости 8 планов – двумя способами по алгоритмам, описанным в предыдущем разделе. При этом замерялась скорость вычисления стоимости.

Полученные в ходе экспериментов результаты отражены на графиках.

5.2 Качество функции стоимости

Приведены 9 графиков, показывающие время выполнения планов, их стоимость, вычисленную с использованием MarkovTable, и их стоимость, вычисленную с использованием PathTree.

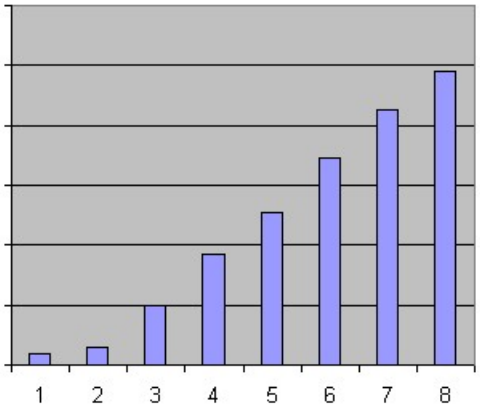

Рис. 8: Маленький файл – время выполнения

Как видно из графиков (рисунки 8, 9, 10, 11, 12, 13), стоимость выполнения планов вычисляется с приемлемой точностью и достаточно хорошо выделяет оптимальный план. Кроме того, видно, что модель селективности с использованием PathTree несколько лучше оценивает стоимость.

5.3 Скорость вычисления стоимости

Из приведенных трех графиков (рисунки 14, 15, 16) можно видеть, что при различных размерах файлов метод оценки селективности с использованием PathTree существенно проигрывает методу, использующему MarkovTable. Это ожидаемый результат – первый метод в процессе вычисления стоимости совершает трудоемкие операции обхода поддеревьев.

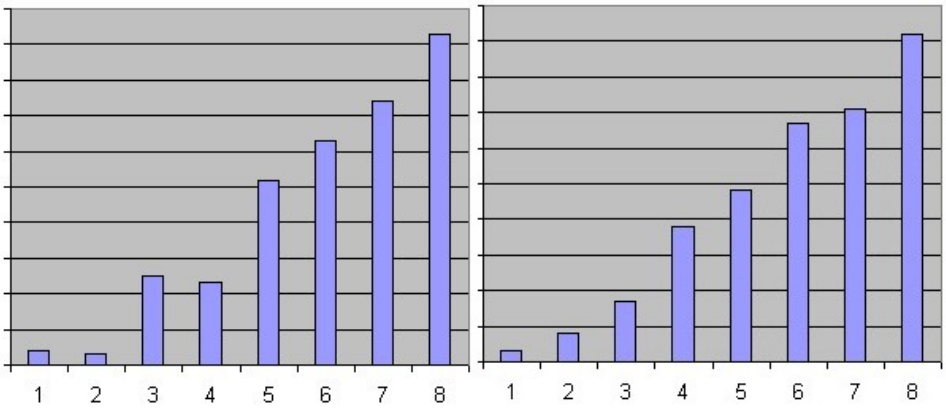

Рис. 9: Маленький файл – MarkovTable, PathTree

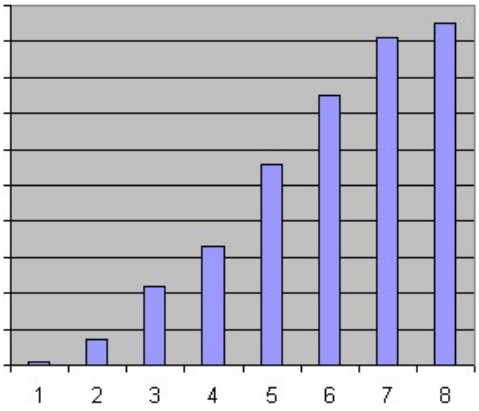

Рис. 10: Средний файл – время выполнения

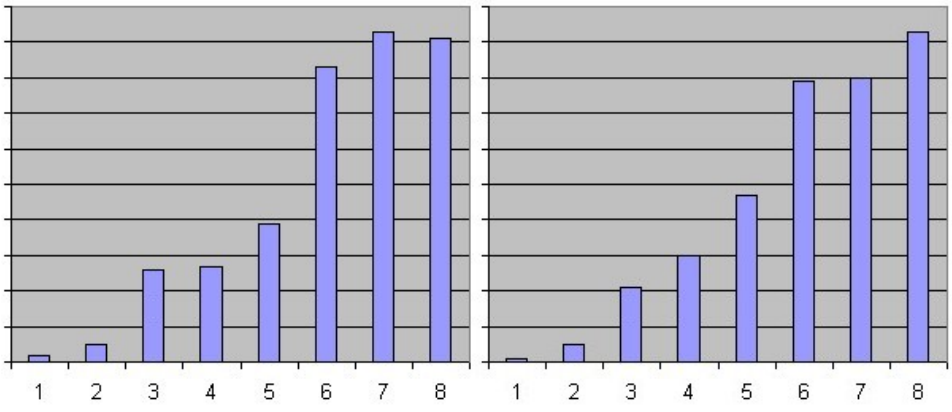

Рис. 11: Средний файл – MarkovTable, PathTree

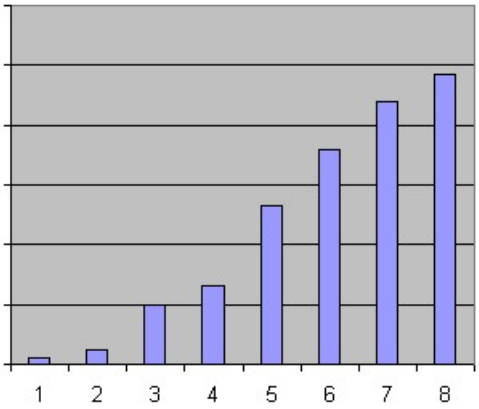

Рис. 12: Большой файл – время выполнения

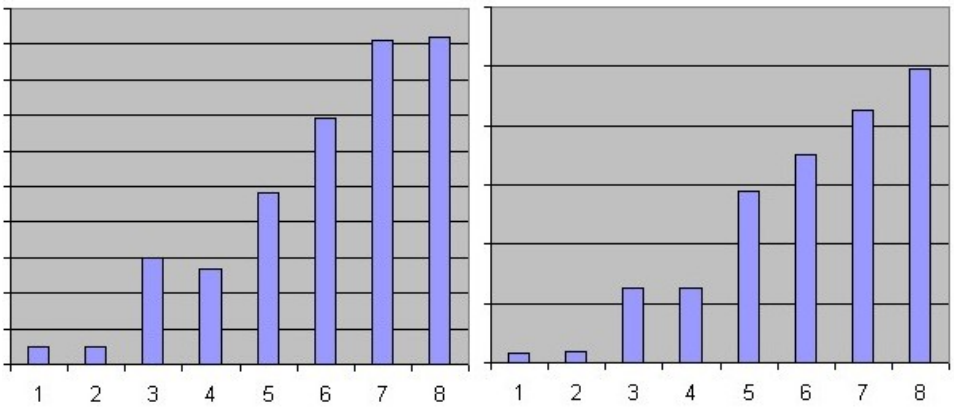

Рис. 13: Большой файл – MarkovTable, PathTree

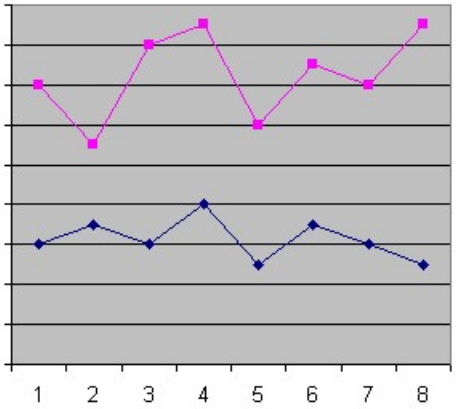

Рис. 14: Маленький файл – скорость вычисления стоимости

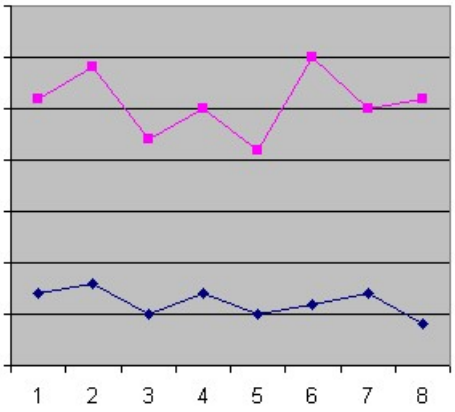

Рис. 15: Маленький файл – скорость вычисления стоимости

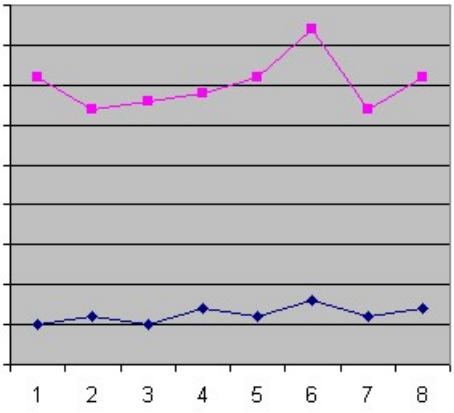

Рис. 16: Большой файл – скорость вычисления стоимости

5.4 Другие эксперименты

Проводились и иные эксперименты. Они, в частности, показали, что:

1. В формуле для $S(P_1/P_2)$ (MarkovTable) предпочтительно брать «первый порядок приближения», поскольку она дает значительно более точный результат, чем простое перемножение, а формулы выше порядком на рассматривавшихся файлах заметного улучшения качества оценки не дали.

2. При суммировании для оси AD (также MarkovTable) хороший результат дает уже значение n, равное двум. Впрочем, это объясняется структурой тестовых файлов, и мы не обобщаем это утверждение на любые XML-данные.

6 Заключение

В данной работе были достигнуты следующие результаты. В рамках программного комплекса XAnswer реализован ряд статистических структур и физических операций. Разработаны методы оценки селективности операций на основе статистических структур PathTree и MarkovTable. Разработана и реализована стоимостная модель для оценки стоимости выполнения навигационных выражений с осями child и descendant-or-self. Проведенные эксперименты показали приемлемую точность стоимостных оценок, выдаваемых функцией, а также позволили сравнить скорость вычисления стоимости плана при использовании оценок селективности, построенных на основе разных статистических структур.

Список литературы

[1] *Aboulnaga, A.* Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. / A. Aboulnaga, A. R. Alameldeen, J. F. Naughton // VLDB / Ed. by P. M. G. Apers, P. Atzeni, S. Ceri et al. — Morgan Kaufmann, 2001. — Pp. 591–600.

[2] *de Aguiar Moraes Filho, J.* Statistics for Cost-Based XML Query Optimization. / J. de Aguiar Moraes Filho, T. Härder // Grundlagen von Datenbanken / Ed. by S. Brass, A. Hinneburg. — Institute of Computer Science, Martin-Luther-University, 2006. — Pp. 110–114.

[3] *Dietz, P. F.* Maintaining order in a linked list / P. F. Dietz // STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing. — New York, NY, USA: ACM Press, 1982. — Pp. 122–127.

[4] Extensible Markup Language (XML) 1.0 (second edition). — 2000. — 6 October. http://www.w3.org/TR/REC-xml/.

[5] *Goldman, R.* DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. / R. Goldman, J. Widom // VLDB / Ed. by M. Jarke, M. J. Carey, K. R. Dittrich et al. — Morgan Kaufmann, 1997. — Pp. 436–445.

[6] *Grust, T.* Accelerating XPath location steps / T. Grust // SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data. — New York, NY, USA: ACM Press, 2002. — Pp. 109–120.

[7] *Ioannidis, Y. E.* Query optimization / Y. E. Ioannidis // *ACM Comput. Surv.* — 1996. — Vol. 28, no. 1. — Pp. 121–123.

[8] *Lim, L.* CXHist : An On-line Classification-Based Histogram for XML String Selectivity Estimation. / L. Lim, M. Wang, J. S. Vitter // VLDB / Ed. by K. Böhm, C. S. Jensen, L. M. Haas et al. — ACM, 2005. — Pp. 1187–1198.

[9] *Mathis, C.* Hash-Based Structural Join Algorithms. / C. Mathis, T. Härder // EDBT Workshops / Ed. by T. Grust, H. Höpfner, A. Illarramendi et al. — Vol. 4254 of *Lecture Notes in Computer Science*. — Springer, 2006. — Pp. 136–149.

[10] *McHugh, J.* Query Optimization for XML. / J. McHugh, J. Widom // VLDB / Ed. by M. P. Atkinson, M. E. Orlowska, P. Valduriez et al. — Morgan Kaufmann, 1999. — Pp. 315–326.

[11] MonetDB. http://monetdb.cwi.nl/.

[12] *Rode, H.* — Methods and Cost Models for XPath Query Processing in Main Memory Databases. — Master's thesis, Universität Konstanz / Fachbereich für Informatik und Informationswissenschaft, 1900. — jan 01. http://www.ub.uni-konstanz.de/kops/volltexte/2004/1195.

[13] *Sartiani, C.* A Framework for Estimating XML Query Cardinality / C. Sartiani. citeseer.ist.psu.edu/sartiani03framework.html.

[14] *Sartiani, C.* Yet Another Query Algebra For XML Data. / C. Sartiani, A. Albano // IDEAS / Ed. by M. A. Nascimento, M. T. Özsu, O. R. Zaïane. — IEEE Computer Society, 2002. — Pp. 106–115.

[15] Sedna XML Database. http://modis.ispras.ru/sedna/index.htm.

[16] System R: an architectural overview / M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin et al. // *IBM Syst. J.* — 1999. — Vol. 38, no. 2-3. — Pp. 375–396.

[17] System R: Relational Approach to Database Management / M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin et al. // *ACM Trans. Database Syst.* — 1976. — Vol. 1, no. 2. — Pp. 97–137.

[18] Very Large DataBase Endowment Inc. http://vldb.org/.

[19] *Wang, S.* Optimization of nested XQuery expressions with orderby clauses / S. Wang, E. A. Rundensteiner, M. Mani // *Data Knowl. Eng.* — 2007. — Vol. 60, no. 2. — Pp. 303–325.

[20] *Wu, Y.* Estimating Answer Sizes for XML Queries. / Y. Wu, J. M. Patel, H. V. Jagadish // EDBT / Ed. by C. S. Jensen, K. G. Jeffery, J. Pokorný et al. — Vol. 2287 of *Lecture Notes in Computer Science*. — Springer, 2002. — Pp. 590–608.

[21] *Wu, Y.* Structural Join Order Selection for XML Query Optimization. / Y. Wu, J. M. Patel, H. V. Jagadish // ICDE / Ed. by U. Dayal, K. Ramamritham, T. M. Vijayaraman. — IEEE Computer Society, 2003. — Pp. 443–454.

[22] XAnswer. http://sourceforge.net/projects/xanswer/.

[23] XML Path Language (XPath) 2.0. — 2002. — 15 November. http://www.w3.org/TR/xpath20/.

[24] XML Schema Part 2: Datatypes. — 2001. — 2 May. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[25] XPathLearner: An On-line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. / L. Lim, M. Wang, S. Padmanabhan et al. // VLDB. — Morgan Kaufmann, 2002. — Pp. 442–453.

[26] XQuery 1.0: An XML Query Language. — 2002. — 15 November. http://www.w3.org/TR/xquery/.

[27] XQuery 1.0 and XPath 2.0 Data Model. — 2002. — 16 August. http://www.w3.org/TR/2002/WD-query-datamodel-20020816/.