Исследование эффективности алгоритмов выполнения алгебраических операций к XML-данным в распределенных запросах

Борис Хвостиченко

Санкт-Петербургский Государственный Университет
boriskhv@acm.org

Аннотация

В данном исследовании производится оценка алгоритмов выполнения запросов к XML-данным и предлагаются модель стоимости и оптимизатор для выполнения XQuery-запросов в сетевой среде с разнородными СУБД.

Выполнение и оптимизация XQuery-запросов внутри отдельной СУБД достаточно изучена, однако проблема выполнения распределенных запросов к разнородным СУБД стала привлекать внимание исследователей сравнительно недавно. При этом основная работа связана с самим выполнением запросов, но не его оптимизацией (например, [6]). Можно обратиться к оптимизации распределенных запросов к реляционным или объектным БД, хороший обзор методик приведен Kossman'ом в [2].
В данной работе предлагается использовать парадигму гибридного переноса (как данных, так и самого запроса) для распределенного выполнения запроса, а также приводится пример модели стоимости выполнения запроса.
Эксперименты показали преимущество гибридного метода переноса над методом переноса запроса, обусловленное распределением нагрузки по узлам сети и выбором самой производительной сети.

1. Введение и обзор ключевых новейших работ по исследуемой тематике

Формат XML представляется наиболее подходящим для хранения и обмена данными между приложениями в интернет-среде на сегодняшний день. Популярность этого формата вызвана по большей части гибкостью в представлении различной информации. Этим же объясняется широкое использование XML для интеграции различных источников данных, причем источники данных совершенно необязательно являются исконно XML СУБД. При этом для осуществления запросов к XML используется предложенный World Wide Web Consortium язык XQuery [9].

Так как часто XML используется для объединения разнородных данных, то возникает необходимость выполнения запросов к различным удаленным источникам данных. В традиционных базах данных достаточно широко известны такие методы выполнения распределенных запросов как перемещение данных (data shipping), перенос запроса (query shipping) и гибридный метод (hybrid shipping). В настоящий момент язык XQuery поддерживает только один метод выполнения удаленного запроса – перемещение данных, который реализуется с помощью функции document(). Некоторая оптимизация этого способа предложена Юрием Солдаком в [5]. Она заключается в предварительной подготовке и отправке той части документа, которая впоследствии будет использоваться в запросе. Фактически это метод переноса запроса, только переносится исключительно операция проекции. Даже с учетом этого способа запрос может выполняться не самым оптимальным способом, например, когда в документе ищется всего один элемент по значению идентификатора. Более того, когда источником данных выступает не XML СУДБ, а виртуальное XML-представление над другой логической моделью данных, этот способ может оказаться вообще неприменимым.

В ходе этого исследования предполагается сделать оценку стоимости выполнения операций в различных реализациях XML-хранилищ данных, а также предложить модель стоимости, правила эквивалентности планов выполнения запроса и алгоритм оптимизатора распределенных XQuery-запросов.

Цели данной работы формулируются следующим образом:

- изучить существующие подходы к реализации запросов к XML-данным и их сильные и слабые стороны,
- получить оценки стоимости выполнения алгебраических операций для нескольких СУБД,
- на основе полученных оценок построить модель стоимости для оптимизатора XQuery-запросов к распределенным источникам данных.

Предположения и допущения

Предположения, которые использовались при дальнейшей работе:

1. Оптимизатор может не знать, какие СУБД стоят на узлах, которые выполняют подзапросы (т.е. узлы – «черные ящики»), но есть доступная статистика по серверам.
2. Считается, что сетевая среда обеспечивает устойчивое (может быть, медленное) соединение, при этом любой узел может связаться с любым другим узлом.
3. Предполагается, что результаты выполнения промежуточных подзапросов не кэшируются на узлах и не могут быть использованы во второй раз.
4. В сети нет копий одних и тех же данных (т.н. «зеркал»), хотя при дальнейшем расширении оптимизатора можно рассматривать и такую возможность.

2. Идея исследования

Идея исследования состоит в применении метода гибридного переноса для выполнения распределенных XQuery-запросов к разнородным источникам данных и оценки возможностей переноса операций распределенного запроса в другие узлы сети при помощи заранее вычисленной статистики. Дальнейшая структура работы построена следующим образом:

1. Описана методология выполнения распределенных запросов.
2. Определена модель стоимости распределенного запроса.
3. Предложены правила эквивалентности.
4. Описан алгоритм оптимизации запросов.
5. Приведены результаты экспериментов, подтверждающих эффективность предложенного метода.

Выполнение распределенных XQuery-запросов

Здесь предполагается, что источники данных ведут себя как узлы в peer-to-peer сети, т.е. каждый из них может выступать как источником запроса, так и его исполнителем. Схема выполнения запроса выглядит следующим образом:

1. Один из узлов, далее называемый «Инициатор», получает XQuery-запрос от клиента.
2. Этот же узел выполняет первичную обработку и оптимизацию запроса и рассылает соответствующие подзапросы в другие узлы сети, которые выступают как серверы БД.
3. Результаты выполнения подзапросов напрямую пересылаются следующему потребителю этих результатов, таким образом нагрузка распределяется по всей сети.
4. Окончательный результат запроса пересылается к Инициатору.

Все взаимодействие с клиентом осуществляет один и тот же узел в сети, а выполнение запроса можно представить в виде сетевого графа, работы в котором – это выполнение подзапросов и пересылка данных по сети.

Подобный подход отличается от предложенного Sucio [6] тем, что, во-первых, могут пересылаться не только запросы, но и данные, и во-вторых, не используется центральный узел для объединения всех результатов подзапросов.

Отличие от метода Солдака [5] заключается в том, что запрос может выполняться не в узле-инициаторе, а передаваться в более производительные СУБД. При этом даже несмотря на потери времени на передачу данных

В свою очередь, стандартный метод выполнения удаленных запросов языка XQuery использует только метод пересылки данных, но не может передавать запросы на другие узлы сети.

Стоимость выполнения распределенного запроса

Стоимость выполнения запроса определяется просто – это «время ответа», т.е. время, по прошествии которого Инициатор получает полный ответ на запрос. Задача оптимизатора – выполнить запрос за минимальное время. Загрузка узлов во внимание не принимается.

Так как запрос можно представить в виде сетевого графа, то стоимость выполнения запроса будет равна длине критического пути в этом графе. Для ее определения необходимо знать время выполнения всех работ в графе, для чего используется упрощенная алгебра XQuery.

Оптимизатор представляет запрос в упрощенной алгебре и оценивает стоимость выполнения запроса на основе известной статистики. Затем он оптимизирует выполнение запроса (т.е. уменьшает время ответа) при помощи переноса выполнения некоторых операций на другие узлы. Таким образом, оптимизатор использует смешанную стратегию переноса данных и запроса (hybrid shipping).

3. Описание методов, алгоритмов и экспериментов

В этом разделе подробно описаны упрощенная алгебра выполнения распределенных запросов, модель стоимости и алгоритм оптимизации выполнения запросов и приведены результаты эксперимента над тремя различными СУДБ.

Упрощенная алгебра для распределенного выполнения запроса

Все операции при сетевом выполнении запроса можно разделить на три основных вида:

1. Операции загрузки полученных узлом данных для их обработки (может отсутствовать для некоторых СУБД).
2. Операции упрощенной алгебры XQuery:
    a. Выборка, как по строго определенному пути, так и по не полностью специфицированному.
    b. Операции конструирования результата.
    c. Упорядочивание и агрегация результата.
    d. Соединение, итерация.
3. Пересылка данных по сети:
    a. Установление соединения.
    b. Собственно пересылка данных.

Скорость выполнения операций типов 1 и 2 зависит от производительности самой СУБД, в то время как выполнение операций типа 3 зависит от пропускной способности сети. Более

того, предполагается, что скорость выполнения той или иной алгебраической операции зависит только от количества элементов в том множестве, над которым производится операция. Поэтому количество элементов на выходе каждой операции оценивается при помощи правил из табл. 1.

Навигация по пути	Среднее количество элементов, доступных по одинаковому пути
Выборка по нестрого специфицированному пути	Среднее количество элементов, доступных по одинаковой метке дуги
Упорядочивание	Число элементов на входе равно числу элементов на выходе
Соединение	Количество разных ключей, умноженное на среднюю повторяемость одного ключа

Табл. 1: правила для вычисления выходного размера множества элементов.

Стоимость операций упрощенной алгебры

Время выполнения сетевых операций определяется любым способом (например, пересылкой некоторого количества данных).

Время выполнения несетевых операций определяется с помощью модификации теста XMark [3, 8]. Каждый узел при вступлении в сеть обязан предоставить всем узлам время выполнения совершенно определенного набора запросов над определенным набором данных (одинаковым для всех) – четырем тестовым файлам размера 0.1, 1, 5 и 10 мегабайт. Табл. 2 содержит статистическую информацию по тестовым документам.

	100Kb	**1Mb**	**5Mb**	**10Mb**
Узлов	2932	29136	142271	286008
Элементов	1729	17132	83533	167865
Текстовых узлов	1203	12004	58738	118143
Средний размер элемента	34	34	35	35
site/people/person	25	255	1275	2550
site/open_auctions/open_auction	12	120	600	1200
site/closed_auctions/closed_auction	10	97	487	975
site/regions//item	22	217	1087	2175

site//description	45	444	2224	4450
site/regions/europe/item	6	60	300	600

Табл. 2. Статистическая информация по тестовым документам.

В приложении А приведены восемь запросов, на основе которых определяется производительность системы. Для каждой операции упрощенной алгебры скорость выполнения определяется как суммарное время, потраченное на все запросы к четырем файлам, поделенное на количество элементов, участвовавших в запросе. То есть производительность рассчитывается следующим образом:

Навигация по пути: $\left(\sum_{size=100kb}^{10mb} Q1 + Q2 + Q3 \right) / 810777$

Навигация по нестрогому пути: $\left(\sum_{size=100kb}^{10mb} Q4 + Q5 \right) / 540518$

Соединение: $\left(\sum_{size=100kb}^{10mb} Q6 - Q1 - Q3 \right) / 3132160$

Конструирование результата: $\left(\sum_{size=100kb}^{10mb} (Q7 - Q1) / 15 \right) / 4105$

Упорядочивание результата: $\left(\sum_{size=100kb}^{10mb} Q8 - Q4 \right) / 3501$

Правило эквивалентности

Уместно предположить, что запрос уже преобразован в дерево выполнения и итерации преобразованы в соединения там, где это требуется. Можно также считать, что проведена начальная оптимизация – например, выборки по условию поставлены как можно ближе к листьям дерева, поэтому оптимизатор не рассматривает возможность изменения порядка следования алгебраических операций. Отчасти это диктуется заданным порядком XML-элементов, из-за которого, например, операция соединения не является коммутативной.

Поэтому для оптимизации используется всего одно правило эквивалентности: запрос можно выполнять на другом узле, предварительно переслав туда данные и подгрузив их в СУБД.

Иначе говоря, если N1 – исходный узел сети, N2 – получатель результата, а N3 – кандидат на перенос запроса, то правило выглядит так:

$$Send(N1 \to N2) \circ Q = Send(N3 \to N2) \circ Q' \circ LoadData(N3) \circ Send(N1 \to N3) \circ GetData(N1)$$

Здесь Q’ – это запрос Q, переписанный для СУБД узла N3 (возможно, он будет вообще не на языке XQuery). В модели не учитывается время, которое будет потрачено на переписывание запросов из одного языка в другой.

Алгоритм оптимизации

Алгоритм оптимизации пытается перенести некоторые операции на другие узлы. Он оценивает возможности переноса операций на другие узлы сети, учитывая при этом стоимость передачи данных и загрузки их в другую СУДБ.

Детальное описание шагов алгоритма выглядит следующим образом:

1. Вычислить критический путь в графе выполнения запроса.
2. По одной просмотреть пары операций (алгебраическая – передача данных), находящиеся на этом пути.
3. Для каждой операции алгебры оценить результат ее переноса на другой узел. Используются заранее вычисленные оценки стоимости операций.
4. Если перенос оказывается выгоднее, то заменить соответствующую операцию согласно правилу эквивалентности и повторить алгоритм с шага 1.
5. Если ни одну операцию нельзя перенести, то завершить работу алгоритма и использовать получившийся план для выполнения запроса.

Алгоритм всегда сходится, потому что при каждом переносе время выполнения запроса уменьшается, но остается положительным.

Интересно, что при выполнении запроса количество алгебраических операций не изменяется, и одну и ту же операцию не имеет смысла просматривать несколько раз. Поэтому трудоемкость алгоритма равна количеству алгебраических операций, умноженному на количество узлов в сети.

Экспериментальные результаты

Для определения производительности алгоритма распределения запроса использовалась часть теста XMark 1.1 [3, 8]. Все эксперименты производились на Pentium-III 498MHz с оперативной памятью 256Мб, с установленной операционной системой Standard Enterprise Windows Platform версии 5 build 97. Для чистоты эксперимента системный кэш был выключен, а СУБД использовались с параметрами, выставленными «по умолчанию». В ходе эксперимента использовались следующие СУБД: Berkeley DB XML 2.1.8 [1], Sedna [4], X-Hive/DB 6.1 [7].

Тест имитирует базу аукционов, в которой отдельными элементами отражены лоты, заявки на покупку лотов, а также отдельные люди, которые сделали эти заявки. В файле содержится детальная информация как о лотах и о людях, так и об открытых и уже завершившихся аукционах. Тест достаточно широко используется для тестирования различных XML СУДБ. Для эксперимента использовался автоматически сгенерированный файл размера 10Мб.

Производительность СУБД, вычисленная согласно вышеописанной методике, приведена в табл. 3. Berkeley оказалась быстрейшей СУБД практически по всем операциям, и вполне логично ожидать, что узел с ней будет использован для выполнения самых трудоемких работ. Еще оказалось, что Sedna не поддерживает упорядочивание результата, поэтому можно считать, что ее производительность при выполнении этой операции равна положительной бесконечности.

	Sedna	X-Hive	Berkeley
Загрузка данных	0.001726	0.001833	0.001114
Поиск по пути	0.002766	0.001505	0.001389
Поиск по нестрогому пути	0.002346	0.019380	0.006340
Соединение	0.025220	0.032715	0.019167
Конструирование результата	0.036898	0.116866	0.028961
Упорядочивание	N/A	1.383890	0.964897

Табл. 3: производительность различных СУБД (ms/элемент).

Тестовый запрос

В качестве тестового примера был взят следующий запрос из стандартного набора запросов XMark: вывести имена всех людей, а

также лоты, которые они купили на аукционе в Европе (см. рис. 1). Этот запрос разбивается в упрощенной алгебре на несколько операций выборки по пути и две операции соединения по ключу.

```
for $p in document("Sedna")/site/people/person
let $a := for $t in
   document("XHive")/site/closed_auctions/closed_auction
   let $n := for $t2 in
        document("Berkeley")/site/regions/europe/item
        where  $t/itemref/@item = $t2/@id
        return $t2
   where $p/@id = $t/buyer/@person
   return <item> $n/name/text() </item>
return <person name=$p/name/text()> $a </person>
```

Рис. 1: тестовый запрос.

Здесь имя документа обозначает название узла, в котором находится документ и считается, что первый узел является инициатором. Запрос выполнялся к 10-мегабайтному тестовому файлу XMark, скорость соединения между узлами показана на рис. 2. Было проведено шесть экспериментов с различными сочетаниями узлов, оказалось, что время выполнения запроса в основном зависит от узла-инициатора. Поэтому в результатах приведено усредненное значение для одного и того же инициатора запроса.

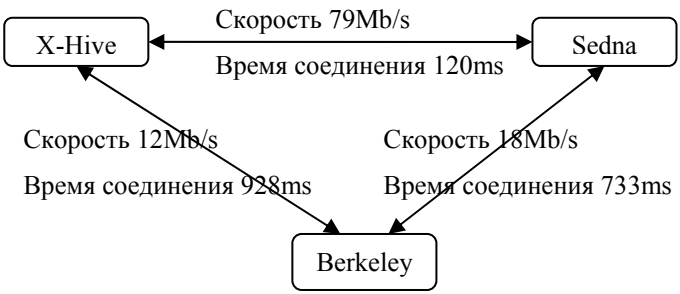

Рис. 2: схема соединений между узлами сети.

Оптимизатор выработал план выполнения запроса, показанный на рис. 3. Все операторы соединения выполняются на узле Berkeley, даже несмотря на медленное соединение между ним и остальными узлами. В предложенном плане выполнения запроса действительно переносятся как данные, так и подзапросы, т.е. используется парадигма гибридного переноса. Интересно, что план выполнения

запроса не зависит от того, какой узел является инициатором запроса, а время выполнения фактически отличается только на время пересылки данных в узел-инициатор запроса.

В предложенном плане все «дорогие» операции соединения перенесены в наиболее производительную СУБД Berkeley, так как стоимость пересылки данных значительно меньше выигрыша, получаемого при выполнении запроса. Данный план не учитывает дополнительную нагрузку на СУБД Berkeley, которая возникает при одновременном получении данных и выполнении запроса.

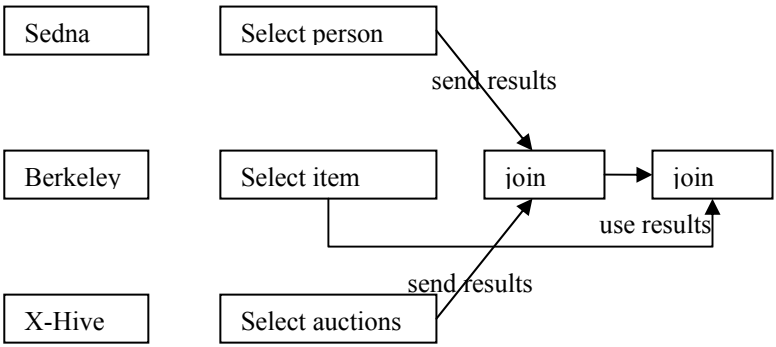

Рис. 3: План выполнения запроса, предложенный оптимизатором.

Время выполнения запросов показано в табл. 4. Метод гибридного переноса действительно позволяет выполнять запросы в несколько раз быстрее, чем метод переноса данных, причем чем больше размер данных, тем больше будет выигрыш во времени выполнения запроса (как абсолютный, так и относительный). Метод, предложенный Солдаком, позволяет достичь той же скорости выполнения запроса, если инициатор запроса – самая производительная СУБД. Надо полагать, что при выполнении достаточно сложного запроса метод Солдака будет проигрывать, потому что не сможет распараллелить трудоемкие операции.

Инициатор	Data shipping	Hybrid shipping	Smart data shipping (Soldak)
Sedna	126554	60786	78346
X-Hive	135599	61154	101560
Berkeley	97465	60709	60820

Табл. 4. Время выполнения тестового запроса (в ms).

Понятно, что наибольший выигрыш достигается при переносе самых трудоемких операций – соединения, упорядочивания, устранения дубликатов, потому что время их выполнения нелинейно зависит от размера входных данных.

4. Выводы и обсуждение результатов

В данной работе исследовано применение метода гибридного переноса для оптимизации распределенных запросов XQuery. Представлены упрощенная алгебра выполнения распределенных запросов, а также модель стоимости и алгоритм оптимизации. Были проведены эксперименты, результаты которых показали работоспособность данной модели оптимизации распределенных вычислений и ее преимущества над методом переноса данных и методом Солдака.

У предложенной модели есть два недостатка:

1. Высокая трудоемкость оптимизации (фактически полный перебор по всем узлам сети).
2. Так как на самом деле работоспособность сети меняется (узлы могут быть загружены пользователями, каналы связи перегружаются или вообще выходят из строя), то модель оптимизации в реальной ситуации вырабатывает субоптимальное решение.

Поэтому можно провести дальнейшую модификацию метода: каждый узел получает свой подзапрос и еще раз оптимизирует его с учетом текущего состояния системы, а для упрощения перебора использовать динамические хэш-таблицы. Также можно учитывать нелинейный характер стоимости выполнения некоторых операций (например, стоимость сортировки или соединения) и более точно оценивать стоимость выполнения подзапросов на удаленных узлах.

Одно из предположений, которое было сделано – это отсутствие узлов-«зеркал» в сети. На самом деле, можно модифицировать модель, чтобы учесть эту возможность. Для этого надо позволить оптимизатору переносить с узла на узел не только операции высокого уровня, но и операции физического доступа к данным. Правда, придется расширить алгебру соответствующим образом. Можно расширить алгебру и еще больше, предложив более точный набор операций из алгебры XQuery [9].

Кроме того, в модели никак не учитывается возможность перестановки порядка выполнения операторов (например, если соединяются три множества), потому что XML задает порядок следования элементов. Можно ввести еще одну операцию – разупорядочивание множества и принудительно использовать ее перед операцией сортировки. Также можно использовать пару операций «сохранить порядок»-«восстановить порядок» чтобы увеличить пространство для оптимизации.

5. Литература

1. Berkeley XML DB Homepage, 2005. http://www.sleepycat.com/products/xml.shtml
2. D. Kossmann. The state of the art in distributed query processing. ACM Comp. Surv., 32(4):422-469. New York: ACM Press, 200.
3. A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.
4. Sedna DB Homepage, 2005. http://modis.ispras.ru/Development/sedna.htm
5. Y. Soldak. Estimating Influence of Subquery Granularity on Evaluating a Query to Remote XML Stores. // Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems. /ed. by S.D Kuznetsov, B.A. Novikov. – SPb.: NIIMM, 2004. – P. 22-25.
6. D. Suciu. Distributed query evaluation on semistructured data. ACM Trans. Database Syst., 27(1):1-62. New York: ACM Press, 2002.
7. X-Hive/DB Homepage, 2005. http://www.x-hive.com/products/db/index.html
8. XMark Homepage, 2005. http://www.xml-benchmark.com
9. XQuery Specification Working Draft, 2005. http://www.w3.org/TR/xquery/

Приложение А. Набор запросов для определения производительности СУБД.

```
Q1. Return the name of the person with ID `person0'
for $b in
document("test.xml")/site/people/person[@id="person0"]
return $b/name/text()

Q2. Return the initial increases of all open auctions.
for $b in
document("test.xml")/site/open_auctions/open_auction
return <increase> $b/bidder[1]/increase/text() </increase>

Q3. How many sold items cost more than 40?
count(for $i in
  document("test.xml")/site/closed_auctions/closed_auction
where $i/price/text() >= 40
return $i/price)

Q4. How many items are listed on all continents?
for $b in document("test.xml")/site/regions
return count ($b//item)

Q5. How many pieces of prose are in our database?}
for $p in document("test.xml")/site
return count($p//description)

Q6. List the names of persons and the number of items they
bought
for $p in document("test.xml")/site/people/person
let $a := for $t in
    document("test.xml")/site/closed_auctions/closed_auction
  where $t/buyer/@person = $p/@id
  return $t
return <item person=$p/name/text()> count ($a) </item>

Q7. List all persons use French markup in the result.
for $t in document("test.xml")/site/people/person
  return <personne>
    <statistiques>
    <sexe> $t/gender/text() </sexe>,
    <age> $t/age/text() </age>,
    <education> $t/education/text()</education>,
    <revenu> $t/income/text() </revenu>
    </statistiques>,
    <coordonnees>
    <nom> $t/name/text() </nom>,
    <rue> $t/street/text() </rue>,
    <ville> $t/city/text() </ville>,
    <pays> $t/country/text() </pays>,
    <reseau>
    <courrier> $t/email/text() </courrier>,
```

```
    <pagePerso> $t/homepage/text()</pagePerso>
    </reseau>,
    </coordonnees>
    <cartePaiement> $t/creditcard/text()</cartePaiement>
  </personne>
```

Q8. Give an alphabetically ordered list of all items along with their location.

```
for $b in document("test.xml")/site/regions//item
let $k := $b/name/text()
return <item name=$k> $b/location/text() </item>
sortby (.)
```

Performance Research of Algebraic Operations Evaluation Algorythms in Distributed XML Queries

Boris Khvostichenko
State University of St. Petersburg

Behind rising interest to XML Query languages and optimizing XML queries evaluation, there is particular interest in optimizing distributed XML queries. At the moment, there is only one broad-spread method of distributed XML query evaluation – data shipping. This doesn't allow nodes other than query initiator to perform subqueries and do parallel processing.

This paper introduces lightweight algebra for distributed XML query evaluation and describes optimization algorithm that uses this algebra and hybrid shipping method to optimize distributed query. Cost model and performance measurement techniques are also present in this paper.