ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 14 за 2015 г.

Коваленко В.Н., Коваленко Е.И.

Оптимизация запросов в системе массовой виртуальной интеграции баз данных

Рекомендуемая форма библиографической ссылки: Коваленко В.Н., Коваленко Е.И. Оптимизация запросов в системе массовой виртуальной интеграции баз данных // Препринты ИПМ им. М.В.Келдыша. 2015. № 14. 26 с. URL: http://library.keldysh.ru/preprint.asp?id=2015-14

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

В.Н. Коваленко, Е.И. Коваленко

Оптимизация запросов в системе массовой виртуальной интеграции баз данных

Москва — 2015

Коваленко В.Н., Коваленко Е.И.

Оптимизация запросов в системе массовой виртуальной интеграции баз данных

Рассматриваются вопросы оптимизации массовых поисковых запросов, которые получают данные из множества независимых баз реляционного типа. Даны экспериментальные оценки наиболее значимых для массовых запросов методов: преобразования логического плана и параллельного выполнения. Приведены условия получения выигрыша от применения двух серверов для параллельного выполнения оператора JOIN.

Ключевые слова: интеграция баз данных, оптимизация запросов, параллельное выполнение запросов, внутриоператорный параллелизм

Kovalenko Victor Nikolaevich, Kovalenko Evgeniia Ivanovna

Query optimization in a system of virtual database mass integration

The problem under consideration – optimization of massive search queries which obtain data from a set of independent relational databases. Experimental estimations of the most significant methods for massive queries: transformations of the logical plan and parallel execution are given. Also, we give conditions of gaining benefits from use of two servers for parallel execution of the JOIN operator.

Key words: database integration, query optimization, parallel query processing, intraoperator parallelism

Работа выполнена при поддержке программы фундаментальных исследований Президиума РАН.

1. Введение

В ИПМ им. М.В. Келдыша РАН разработана система виртуальной интеграции распределённых баз данных (MQ-DAI) [1], позволяющая создавать масштабные информационные инфраструктуры из 100-1000 баз. Поддерживаемые MQ-DAI запросы являются массовыми: они позволяют получать данные сразу из определённой совокупности баз (группы), причём способы представления данных в разных базах могут различаться. Испытания системы показали её практическую пригодность, на основе MQ-DAI создан программный комплекс [2] доступа к множеству распределённых архивов (PACS-серверов) медицинских изображений. Одним из актуальных вопросов развития имеющейся реализации MQ-DAI является оптимизация запросов, направленная на сокращение времени их выполнения, повышение масштабируемости системы интеграции и снятие ограничений по объёму данных в интегрируемых базах.

Оптимизации придаётся большое значение и в традиционных СУБД, но в системах виртуальной интеграции баз данных ситуация осложняется. Один запрос в таких системах получает данные из нескольких распределённых источников, вследствие чего возрастают издержки на сетевое взаимодействие и передачу данных. Эти факторы должны учитываться и компенсироваться специальными способами оптимизации [3]. Язык массовых запросов [4] системы MQ-DAI является расширением языка SQL-92 для поискового запроса SELECT и предназначен для работы в инфраструктурах с большим числом составляющих – баз данных, что привносит новый уровень вычислительной сложности: даже при небольшом объёме результирующих данных время выполнения может составлять десятки минут. Это даёт основание к рассмотрению не только традиционных методов оптимизации, но также, например, и методов, разработанных для параллельных СУБД.

Задача данной работы – определить круг наиболее значимых для массовых запросов методов оптимизации. Рассмотрены два типа методов: преобразования логического плана и параллельное выполнение. На основе анализа процесса выполнения запросов сформулированы условия получения выигрыша от параллельного выполнения на двух серверах. Приведены оценки эффективности рассмотренных методов оптимизации на экспериментальном стенде информационной инфраструктуры.

В качестве инструментального аппарата для проведения исследований использована система MQ-DAI, которая служит средством подготовки запросов и их запуска, а также комплекс OGSA-DAI/DQP [5, 6]. OGSA-DAI/DQP реализует основанные на Web-службах стандарты WS-DAI [7] удалённого доступа к базам данных и по этой причине широко используется как базовый аппарат при разработке приложений и системных средств. Система MQ-DAI представляет собой надстройку над этим комплексом, сохраняет его архитектуру и в части выполнения запросов практически полностью опирается

на его механизмы. Для целей работы такая организация системы интеграции оказалось весьма удачной, так как в части комплекса OGSA-DAI/DQP – системе OGSA-DQP реализованы в той или иной степени многие из известных методов оптимизации, а также предусмотрен способ расширения их состава.

2. Процесс обработки массового запроса

Возможности оптимизации во многом определяются способом обработки запросов в той или иной системе. Современные системы интеграции баз данных реализуются в архитектуре с промежуточным слоем – медиатором, роль которого заключается в приёме поступающих по сети от клиентских приложений запросов, выборке данных из указанных в запросе баз, выполнении над ними заданных в запросе операторов и возвращении результатов. Проблема интеграции гетерогенных баз, различающихся операционной средой и СУБД, нашла решение путём введения в архитектуру медиатора оболочек, которые обеспечивают унифицированный интерфейс доступа ко всему множеству интегрируемых баз [8].

Система OGSA-DQP вносит в медиаторную архитектуру существенные дополнения, в том числе касающихся способа выполнения запроса. Во-первых, если обычно вся обработка распределённых запросов происходит на единственном сервере, то OGSA-DQP позволяет создавать конфигурации, в которых имеется отдельный сервер приёма запросов (центральный сервер) и несколько серверов выполнения, на которых происходит их выполнение. В ситуации с несколькими серверами полный запрос делится на части, которые выполняются на разных серверах. Взаимодействие между всеми серверами, в том числе передача данных, осуществляется посредством Web-служб. В качестве протокола обмена сообщениями используется SOAP.

Во-вторых, OGSA-DQP обладает разнообразными механизмами расширения. В частности, это позволило реализовать обработку массовых запросов в форме препроцессора, который переводит их в распределённые запросы системы OGSA-DQP.

Процесс обработки массового запроса в описанной выше среде происходит следующим образом.

1. Поступивший на центральный сервер массовый запрос преобразуется в запрос системы OGSA-DQP. Так как в массовом запросе данные адресуются в терминах унифицированного представления – глобальной схемы, осуществляется интеграция данных, то есть вместо символических конструкций – групп подставляются идентификаторы конкретных баз, а также производится отображение имён таблиц и их атрибутов на схемы участвующих в запросе баз. Эти действия выполняет система MQ-DAI, и полученный в результате преобразованный запрос передаётся OGSA-DQP.

2. Производится синтаксический разбор запроса. Результатом является логический план запроса (LQP), который представляет собой бинарное дерево. В плане запроса терминальные узлы – листья – соответствуют операторам

получения данных из баз, участвующих в запросе, а нетерминальные узлы – реляционным операторам и операторам передачи данных.

3. Логический план запроса трансформируется последовательно применяемыми к нему несколькими оптимизаторами. Имеющийся в OGSA-DQP набор оптимизаторов включает преобразования, направленные на минимизацию объёма данных, выбираемых из баз, определения порядка выполнения оператора JOIN с учётом фактического размера таблиц и пр. Набор таких преобразований может расширяться путём добавления новых оптимизаторов в конфигурационный файл центрального сервера.

4. Строится план выполнения запроса (QEP). Оптимизированный логический план разбивается на разделы, узлам каждого раздела ставится в соответствие определённый сервер выполнения. Фактически сейчас OGSA-DQP поддерживает формирование плана выполнения лишь для одного сервера, однако имеются программные интерфейсы, использованные в данной работе, для размещения разделов на нескольких серверах.

5. Выполнение запроса. Каждый раздел плана преобразуется в цепочку действий базовой части – OGSA-DAI, причём в цепочку, содержащую головной узел плана, добавляется действие пересылки результатов запроса на центральный сервер. Далее цепочки рассылаются по серверам, на которых они выполняются.

В дальнейшем изложении рассматривается преобразования, выполняемые на шагах 3 и 4 процесса обработки.

3. Преобразования логического плана

Из имеющихся в системе OGSA-DQP оптимизаторов логического плана для массовых запросов наиболее существенны два оптимизатора, сокращающие объём данных, которые выбираются из баз и передаются на сервер выполнения:

- оптимизатор, исключающий выборку избыточных столбцов;
- оптимизатор, выталкивающий отбор строк в базы (Push Down Optimiser).

Покажем значимость этих оптимизаторов и оценим достигаемый с их помощью эффект.

3.1. Исключение выборки избыточных столбцов

Язык массовых запросов, являющийся расширением SQL, позволяет получать однотипные данные из многих баз, причём в разных базах представление данных (именование, формат) может быть различным. Возможность создания языка для работы в таких условиях обеспечивается подходом интеграции данных [9], который позволяет привести их к унифицированному представлению, в качестве которого используется глобальная схема информационного пространства. Интеграция данных производится путём определения отображений глобальной схемы на схемы

интегрируемых баз. В разработанном специально для массовых запросов методе интеграции [4] отображение задаётся отдельно для каждой базы в форме запроса SELECT. Каждый такой запрос определяет способ получения данных некоторой таблицы глобальной схемы из данных одной базы. С помощью оператора SELECT могут быть определены как простые унифицирующие преобразования, например переименование столбцов, так и сложные, в которых глобальная таблица порождается из нескольких таблиц базы путём выполнения реляционных операторов.

Отображающие запросы используются при преобразовании массового запроса в запрос системы OGSA-DQP: для каждой указанной в части FROM глобальной таблицы выполняется подстановка отображающего запроса SELECT. Так как отображающий запрос определяет глобальную таблицу целиком, в преобразованном запросе из базы данных будут извлекаться “лишние” столбцы, которые не требуется выдавать в качестве результата. Проиллюстрируем это на примере массового запроса

SELECT name, address FROM G1.persons

Запрос получает данные, соответствующие глобальной таблице persons, из баз, которые входят в состав группы G1 (в данном примере из двух баз DB1, DB2). Результаты запроса состоят из строк с полями name и address.

С целью упрощения мы приводим простейшие запросы отображения, в которых отсутствуют преобразование данных, а имена столбцов в базах совпадают со столбцами глобальной таблицы. После подстановки запросов отображения исходный запрос преобразуется в:

```
SELECT G1.G1_persons_name AS name, G1.G1_persons_ address AS address
FROM (
        (
            SELECT   persons.address AS G1_persons_address,          (1)
                     persons.name AS G1_persons_name,                (2)
                     persons.description AS G1_persons_description,  (3)
                     persons.nmb AS G1_persons_nmb,                  (4)
                     persons.passport AS G1_persons_passport,        (5)
            FROM                                                     (6)
                     (SELECT nmb, name, passport, address, description (7)
                     FROM DB1)                                       (8)
                     AS persons                                      (9)
        UNION ALL
            SELECT   persons.address AS G1_persons_address,          (10)
                     persons.name AS G1_persons_name,                (11)
                     persons.description AS G1_persons_description,  (12)
                     persons.nmb AS G1_persons_nmb,                  (13)
                     persons.passport AS G1_persons_passport         (14)
            FROM                                                     (15)
```

```
                (SELECT nmb, name, passport, address, description   (16)
                FROM DB2)                                           (17)
                AS persons                                          (18)
    )
) AS G1
```

Строки 1-9 и 10-18 соответствуют отображающим запросам к двум базам, в которых помимо столбцов результата name и address из баз выбираются также 3 других столбца глобальной таблицы persons.

Как показывается далее, один из наиболее значимых факторов, увеличивающих время выполнения запроса, – объём данных, выбираемых из баз. Проблему появления избыточных столбцов решает наличие в системе OGSA-DQP оптимизатора, который их устраняет на этапе трансформации логического плана запроса.

Эффект оптимизации такого рода демонстрирует эксперимент, в котором сравнивается время выполнения двух запросов SELECT. Данные выбираются из 6 баз, объединённых в группу, общий объём результатов – 18000 строк (по 3000 строк из каждой базы). В первом запросе выбирается один столбец с данными типа INTEGER (10), во втором – в дополнении к первому столбцу выбирается ещё один с данными типа CHARACTER.

Время выполнения первого запроса составляет 5.82 сек., второго – 7.48 сек. Ускорение существенное, особенно с учётом того, что затраты на выборку столбцов суммируются. Если выбираются все 5 столбцов таблицы, то время возрастает до 11.96 сек.

3.2. Проталкивание условий отбора строк на нижний уровень логического плана

Аналогично предыдущему виду оптимизации ускорение выполнения запросов может быть достигнуто путём уменьшения числа выбираемых из баз строк. Ограничения на выбор строк задаются в части WHERE запроса SELECT в виде условных выражений, применяемых к значениям полей выбираемых данных. Оптимизация запроса заключается в том, что такие условия выносятся в терминальные узлы логического плана – в оператор TABLE_SCAN, так чтобы отбор строк производился в самой базе и не передавался на сервер выполнения.

Для демонстрации эффективности такой оптимизации мы используем модельный эксперимент, в котором сопоставляются два массовых запроса:

```
SELECT G1.persons.nmb, G1.persons.name, G2.persons.nmb, G2.persons.name
FROM  G1.persons, G2.persons
WHERE G1.persons.nmb = G2.persons.nmb + 59950
```

и

```
SELECT G1.persons.nmb, G1.persons.name, G2.persons.nmb, G2.persons.name
FROM  G1.persons, G2.persons
WHERE G1.persons.nmb = G2.persons.nmb + 59950 AND G1.persons.nmb < 51
```

Постановка эксперимента следующая. В группе G1 две базы, каждая содержит 60000 строк. В группе G2 одна база с 15000 строк. Поля столбца persons.nmb во всех базах наполнены целыми числами от 1 до 60000.

Условие соединения (G1.persons.nmb = G2.persons.nmb + 59950) таково, что число строк результата в обоих запросах равно 50. Дополнительное условие во втором запросе G1.persons.nmb < 51 не меняет результата, однако время выполнения первого запроса – 8.9 сек., второго – 1.1 сек.

Хотя рассмотренный пример искусственный, оптимизация путём проталкивания выборки строк в базы весьма эффективна для любых запросов с бинарными операторами JOIN, UNION, также как и для запросов без бинарных операторов.

4. Методы параллельного выполнения запросов в среде распределённых серверов

Логический план запроса, строящийся в процессе его обработки, представляет собой бинарное дерево, в котором узлы соответствуют выполняемым операторам, а рёбра – потокам данных. Благодаря такой структуре выполнение запроса может быть ускорено путём параллельного выполнения частей плана.

Известны три классические формы параллельного выполнения запросов [10] (рис. 1):

– конвейерный параллелизм, когда в цепочке операторов, последовательно передающих друг другу данные, одновременно обрабатывается несколько блоков данных;

– межоператорный параллелизм, когда параллельно выполняются операторы, которые не зависят друг от друга по данным;

– внутриоператорный параллелизм, заключающийся в разбиении данных на фрагменты и выполнении нескольких клонов операторов над этими фрагментами параллельно.

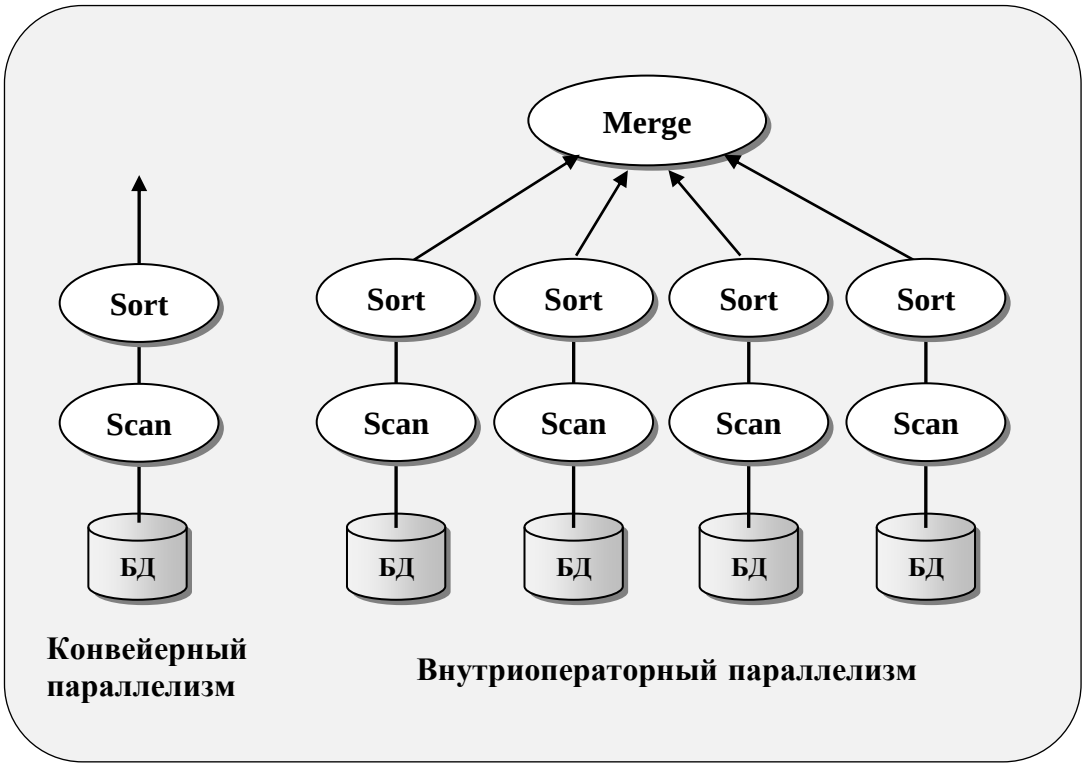

Рис. 1. Формы параллельного выполнения запросов [11].

Все эти формы ведут начало от параллельных баз данных, однако по сравнению с ними функционирование систем виртуальной интеграции происходит в более сложных условиях относительно медленных сетевых соединений и необходимости платформенно независимых протоколов обмена. Эти обстоятельства ставят вопрос о том, какие из методов параллельного выполнения могут быть полезны при виртуальной интеграции.

4.1. Схема выполнения запросов

Для анализа возможностей оптимизации путём распараллеливания рассмотрим, прежде всего, более детально этап выполнения запроса. Подготовка выполнения реализуется системой OGSA-DQP путём преобразования операторов логического плана LQP в действия базовой части комплекса – OGSA-DAI [5]. В ходе собственно выполнения каждое действие производит соответствующую обработку данных и передаёт свои результаты вышестоящему оператору. Если это бинарный реляционный оператор, он комбинирует два потока результатов, полученных в нижестоящих операторах. Оператор получения данных из баз (TABLE_SCAN) выполняет эту функцию посредством Web-сервиса доступа к данным (GDS), который установлен на всех серверах выполнения.

Следуя [12], приведём анализ операций в одной ветви плана выполнения QEP (рис. 2), который включает:

- получение данных из базы,
- передачу данных между операторами разного уровня,

– пересылку результатов на центральный сервер (ЦС).

Предполагается, что операторы распределены по двум серверам выполнения, обозначенным на рисунке СВ1 и СВ2.

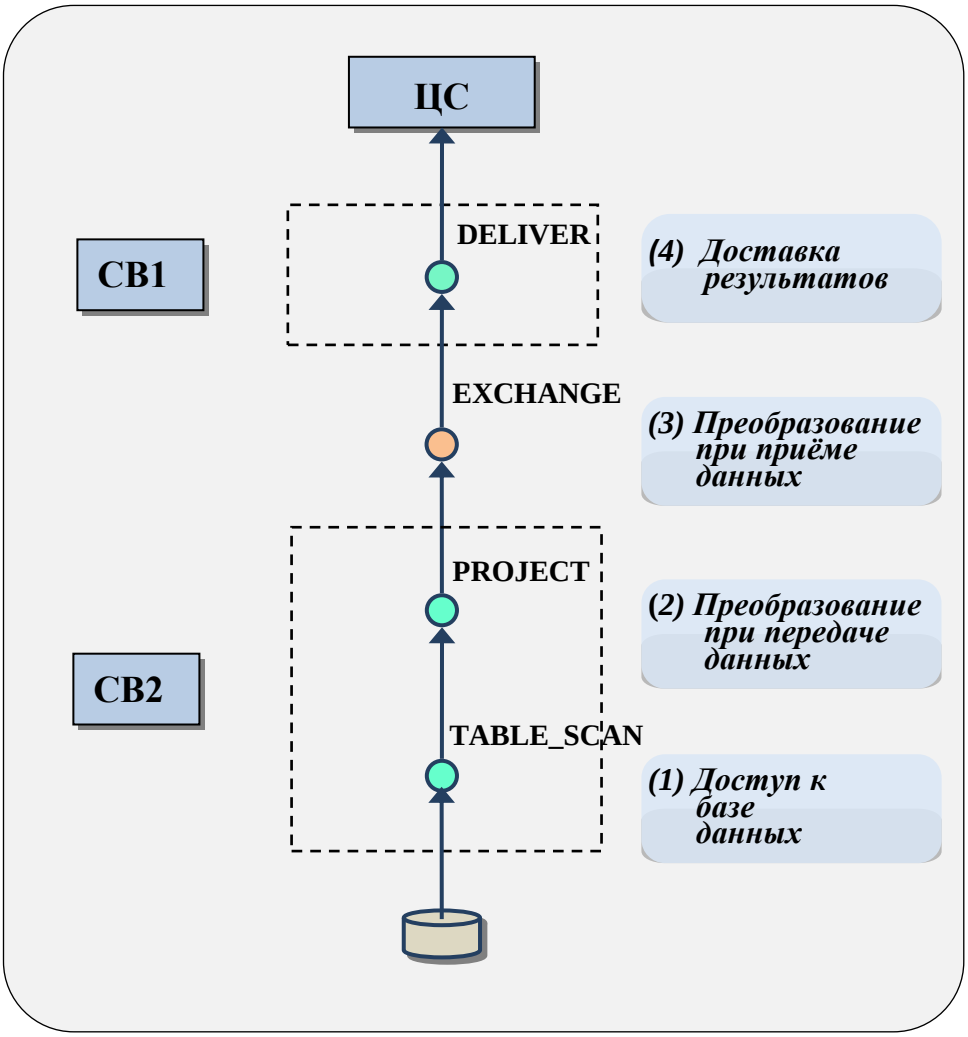

Рис. 2. Выполнение запроса в одной ветви QEP. Справа указаны наиболее значимые этапы выполнения, слева – серверы, на которых они выполняются

Наибольший вклад во время выполнения вносят следующие этапы.

(1) Доступ к базе данных через сервис GDS. На этом этапе GDS получает строки из базы по протоколу JDBC, преобразует их в текстовый формат WebRowSet и передает их в виде XML-сообщения оператору TABLE_SCAN.

(2) Преобразование при передаче данных. Этот этап реализуется оператором EXCHANGE. На посылающей стороне (сервер СВ2) выполняется упаковка строк в блоки и посылка блоков по сети.

(3) Преобразование при получении данных. Принимающая сторона оператора EXCHANGE на СВ1 распаковывает блоки обратно в набор строк.

(4) Доставка результатов. Передача данных от корневого оператора на центральный сервер ЦС, где он становится доступным клиенту.

На всех этапах данные передаются блоками по протоколу SOAP (Simple Object Access Protocol) посредством Web-сервисов.

4.2. Эффект конвейерного параллелизма

Благодаря тому, что действия системы OGSA-DAI реализованы в соответствии с итераторной моделью [13], процесс выполнения представляет собой конвейер, образованный цепочками операторов запроса, в которых выход одного оператора поступает на вход следующего. Параллельность достигается за счёт того, что разные операторы работают одновременно над несколькими порциями данных. В работе [12] приведены экспериментально полученные времена выполнения отдельных этапов запроса, соответствующего QEP на рис. 2. Суммарное время их выполнения составляет 50 сек., однако фактически время выполнения запроса равно 28.6 сек. Эффект почти двукратного ускорения достигается за счёт конвейерного параллелизма – этапы выполняются параллельно на разных машинах, обрабатывающих разные блоки данных.

В конвейере OGSA-DQP используются затратные протоколы передачи данных, поэтому большое значение приобретает метод блочного агрегирования. Блочное агрегирование позволяет уменьшить накладные расходы, которые связаны с пересылками данных между операторами, расположенными на разных серверах. Согласно этому методу посылающая сторона передаёт строки не поодиночке, а блоками определённого размера в одном сообщении. Выигрыш достигается за счёт того, что время передачи набора строк в одном блоке меньше, чем время пересылки каждой строки отдельно. Размер блока является параметром, и при выборе его значения принимаются во внимание характеристики сети.

Дополнительное преимущество поблочной пересылки проявляется в том, что этот метод в определённой степени компенсирует неравномерность поставки данных. Когда строки поставляются одна за другой, любая сетевая задержка сразу приводит к остановке выполнения запроса на принимающей стороне из-за нехватки строк. При передаче блоками принимающая сторона имеет запас строк, которыми она снабжает оператор обработки данных на своей стороне даже при задержке следующего блока.

Оценка выигрыша за счёт блочного агрегирования при получении данных от сервиса GDS получена в эксперименте, описанном в цитированной выше работе. Показано, что при чтении без блокирования (размер блока – одна строка) время получения данных – 218 секунд, а для блоков размера 110-130 – 10.5 секунд, то есть, блокирование даёт весьма значительный эффект. Более детально, время чтения падает с 50.5 до 10.5 секунд при увеличении числа строк в блоке от 5 до 30. Однако затем цена построения больших блоков начинает сказываться, и дальнейшего улучшения не происходит.

4.3. Сравнение производительности отдельных операторов

Для анализа конвейерного параллелизма, также как и рассматриваемого далее внутриоператорного параллелизма, важным является вопрос о соотношении производительности отдельных операторов. В плане выполнения могут присутствовать операторы SQL, оператор TABLE_SCAN получения данных из баз, а также оператор EXCHANGE обмена данными между операторами, размещёнными на разных серверах. Оценки производительности основных реляционных операторов (SELECT, UNION, JOIN) получены в работе [14] (таблица 1).

Таблица 1

Оценки производительности операторов LQP

Оператор	Производительность (мкс на строку)
SELECT, PROJECT, RENAME	10 – 22
UNION ALL	12.4 – 16
JOIN	30
TABLE_SCAN	900

Сравнение показывает, что оператор TABLE_SCAN работает более чем в 30 медленнее, чем реляционные операторы, так что последние могут не учитываться при оценке времени выполнения запросов (при равном числе обрабатываемых строк). Отметим, что из всего времени, затрачиваемого реализующим TABLE_SCAN сервисом GDS, до 80% занимает процессорная обработка: упаковка получаемых из базы данных в блоки и преобразование в формат WebRowSet. Для дистанционного доступа к базам GDS использует протокол JDBC, производительность которого оценивается примерно в 25 мкс на строку, так что затраты GDS в этом отношении незначительны.

Оценок производительности оператора обмена EXCHANGE нет. Однако задействованные в нём механизмы передачи имеют сложность, аналогичную механизмам TABLE_SCAN.

4.4. Внутриоператорный параллелизм

Из трёх известных форм параллельного выполнения запросов (конвейерного, межоператорного и внутриоператорного) в OGSA-DQP реализован и показал эффективность конвейерный параллелизм. Имеющиеся результаты по внутриоператорной форме в условиях, когда выполнение запросов происходит на нескольких серверах, немногочисленны [6, 14-17], и окончательного ответа на вопрос о его эффективности нет. Рассматриваемый далее способ параллельного выполнения ориентирован, прежде всего, на массовые запросы, для которых внутриоператорный параллелизм выглядит

естественным. В этой форме параллелизма операторы запроса клонируются и выполняются над различными фрагментами данных. Но в массовых запросах системы MQ-DAI операторы применяются к группам баз данных, так что фрагментация имеется изначально: в качестве фрагментов могут выступать отдельные базы или их объединения.

Далее даётся анализ эффективности параллельного выполнения для двух типов массовых запросов: для запроса, получающего данные из одной группы баз, и запроса, который содержит оператор соединения двух групп баз.

4.5. Параллельное выполнение оператора UNION

Рассмотрим массовый запрос, в котором участвует одна группа:

SELECT < столбцы > FROM G.T

Препроцессором MQ-DAI он преобразуется в запрос OGSA-DQP:

SELECT < столбцы > FROM (DB1_T ∪ DB2_T … ∪ DBn_T)

Здесь G обозначает группу, состоящую из баз {DB1,…, DBn}, T – таблицу глобальной схемы, {DB1_T,…, DBn_T} – таблицы, полученные из соответствующих баз, и преобразованные в схему T. Символ U обозначает оператор UNION ALL.

Дерево выполнения запроса приведено на рис. 3. В нём показаны наиболее значимые операторы:

- получение данных из баз (оператор TABLE_SCAN),
- операторы UNION ALL,
- доставка результатов на центральный сервер (DELIVER).

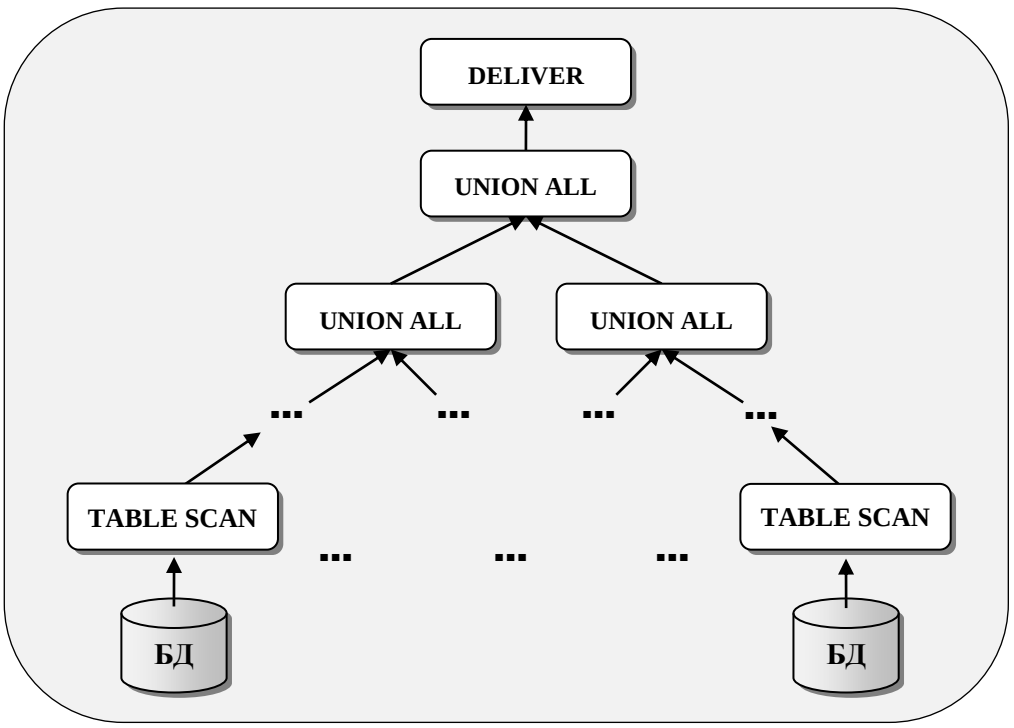

Рис. 3. Дерево выполнения запроса SELECT с одной группой на одном сервере.

Кроме того, в дереве имеются операторы для преобразования входных данных в формат глобальной схемы – PROJECT, RENAME. Они опущены, так как их вклад во время выполнения незначителен. Операторы UNION ALL могут образовывать многоуровневую иерархию, что показано многоточиями. Каждая ветвь дерева заканчивается оператором TABLE_SCAN получения данных из баз.

Предлагаемый способ оптимизации заключается в том, что дерево выполнения делится на две части так, чтобы они выполнялись на разных серверах (рис. 4). Ограничимся случаем такого разбиения, которое производится на некотором бинарном операторе (здесь оператор UNION ALL) и при котором получается лишь один поток данных между двумя серверами. В этом случае на втором сервере размещается некоторое поддерево целиком. Исходное дерево выполнения модифицируется: между двумя операторами, которые обмениваются данными, но расположены на разных серверах, вставляется оператор EXCHANGE. Он имеет две составляющие: одну на посылающей стороне, вторую на принимающей.

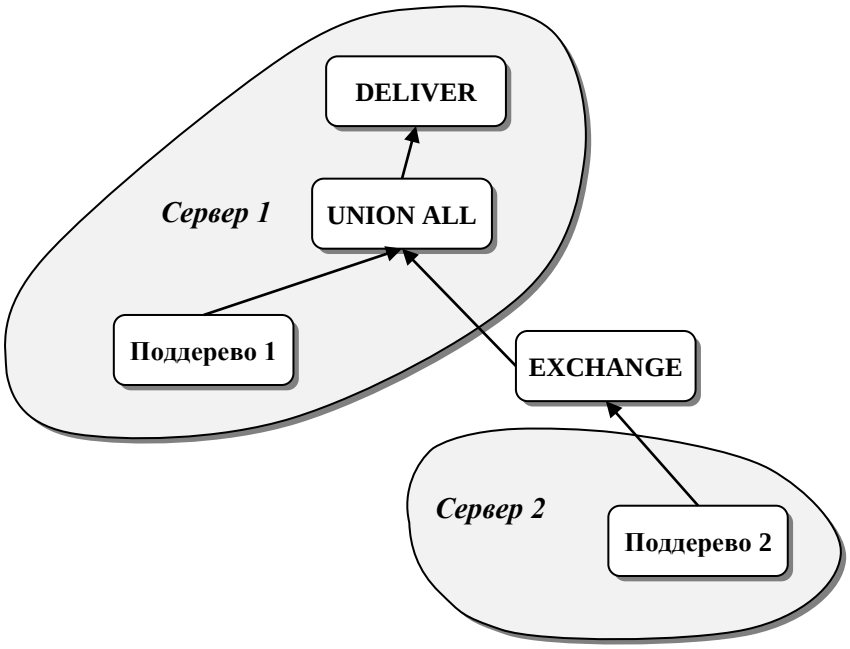

Рис. 4. Модифицированное дерево выполнения запроса для двух серверов.

Дадим аналитическую оценку выигрыша, исходя из разницы процессорных времён, которые затрачиваются на одном и двух серверах. Во время выполнения запроса вносят вклад также сетевые задержки, но они в определённой степени компенсируются конвейерным параллелизмом.

Обозначим процессорное время выполнения части запроса, которая содержит головной узел плана (левая часть) – T_L, остальной части дерева (правая часть) – T_R. Общее время выполнения запроса на одном сервере:

$$T_1 = T_L + T_R.$$

При выполнении запроса на двух серверах в левой части добавляется время выполнения принимающей части оператора EXCHANGE – E_L. В правой части – посылающей части EXCHANGE – E_R. Здесь обе части выполняются параллельно на разных процессорах, и время выполнения запроса составляет:

$$T_2 = \max (T_L + E_L, T_R + E_R).$$

Выигрыш от распараллеливания $\Delta = T_1 - T_2 > 0$ достигается при одновременном выполнении двух условий:

$$\begin{cases} T_L + T_R > T_L + E_L \\ T_L + T_R > T_R + E_R \end{cases} \text{ или } \begin{cases} T_R > E_L \\ T_L > E_R \end{cases}$$

Приведённые в п. 4.3 характеристики производительности операторов позволяют считать, что время выполнения обеих частей практически совпадает со временем этапа получения данных из баз посредством сервиса GDS. В дереве выполнения операторы UNION ALL объединяют полученные строки, так что объём передаваемых и принимаемых оператором EXCHANGE данных равен объёму данных, получаемых из баз. Заметим, что это обеспечивается оптимизирующими преобразованиями OGSA-DQP, за счёт чего из баз получается только необходимая часть таблиц. Также есть основания предполагать, что показатели производительности оператора EXCHANGE не лучше, чем у сервиса GDS. Из всего сказанного можно заключить, что T_R примерно равно E_L, так что выполнение запросов, не содержащих иных, кроме UNION ALL, бинарных операторов, не даёт выигрыша на двух серверах.

Этот вывод подтверждают полученные нами экспериментальные результаты, представленные в таблице 2. В ней приведены времена выполнения запроса с одной группой в зависимости от числа строк, получаемых из баз (всего в группе 4 базы). В первом столбце указано общее число получаемых строк, во втором – время выполнения запроса на одном сервере, в третьем – на двух серверах.

Таблица 2

Время выполнения запроса с одной группой на одном и двух серверах

Число входных строк (тысячи)	1 сервер (сек.)	2 сервера (сек.)
0	0,8	1,18
2	1,78	2,19
4	2,77	3,22
6	3,95	4,35
8	5,00	5,35
10	6,27	6,80
12	7,29	7,60
14	8,25	8,57
16	9,21	9,80
18	11,29	11,66

При всех значениях числа входных строк время выполнения на одном сервере оказывается меньшим.

4.6. Параллельное выполнение оператора JOIN

Распределение оператора UNION ALL по нескольким серверам не даёт выигрыша по причине того, что количество строк, вводимых из баз, совпадает с количеством строк результата. Для оператора JOIN это, как правило, не так. Над данными, введёнными из баз, на сервере системы выполняется операция соединения, которая сокращает число строк, и это уменьшает затраты на их передачу между серверами.

Для оптимизации JOIN применяется известный способ. Если JOIN выполняется над таблицами A и B, и одна из таблиц может быть представлена в виде объединения, например $A = A_1 \cup A_2$, то имеется два варианта вычислить соединение $A \bowtie B$ (знак $\bowtie$ обозначает оператор JOIN):

$$(A_1 \cup A_2) \bowtie B \qquad \text{или} \qquad (A_1 \bowtie B) \cup (A_2 \bowtie B)$$

Второй вариант может быть использован для внутриоператорной оптимизации.

В языке массовых запросов JOIN выполняется над глобальными таблицами, которые являются объединением данных из нескольких баз, образующих группу:

$$(A_1 \cup A_2 \cup \ldots \cup A_N) \bowtie (B_1 \cup B_2 \cup \ldots \cup B_M)$$

Предлагаемая оптимизация заключается в разбиении первой группы на две равные по объёму поставляемых данных части:

$$A = (A_1 \cup \ldots \cup A_K) \cup (A_{K+1} \cup \ldots \cup A_N)$$

Исходный оператор JOIN преобразуется в два, которые выполняются над меньшим числом баз и результаты которых объединяются оператором UNION ALL:

$$((A_1 \cup \ldots \cup A_K) \bowtie (B_1 \cup B_2 \cup \ldots \cup B_M)) \cup ((A_{K+1} \cup \ldots \cup A_N) \bowtie (B_1 \cup B_2 \cup \ldots \cup B_M))$$

Деревья выполнения такого запроса на одном и двух серверах аналогичны деревьям для рассмотренного выше оператора UNION ALL. Отличие лишь в том, что и в левой и в правой части дерева выполняются операторы JOIN.

Укрупнённое дерево выполнения на двух серверах показано на рис. 5. Узлы $A_L \bowtie B$ и $A_R \bowtie B$ обозначают поддеревья, которые вычисляют левую и правую части оператора UNION ALL. Дерево распределено по серверам также как в п. 4.5: поддеревья $A_L \bowtie B$ и $A_R \bowtie B$ размещены на разных серверах, а оператор UNION ALL получает один из входных потоков данных $A_R \bowtie B$ с помощью оператора EXCHANGE. Потенциальный выигрыш может быть получен за счёт сокращения числа строк при выполнении каждого из поддеревьев.

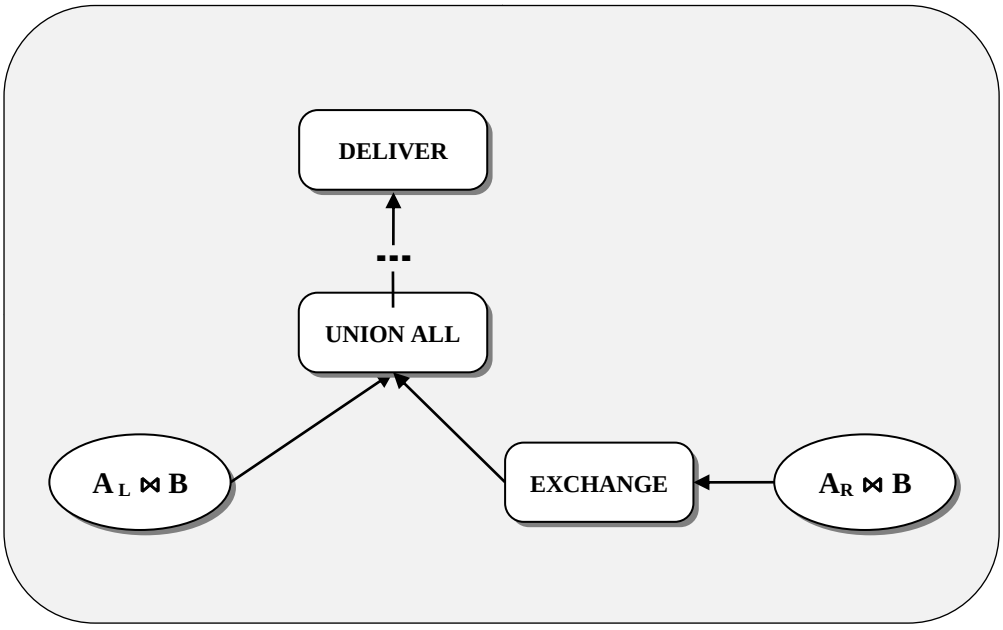

Рис. 5. Дерево выполнения JOIN на двух серверах.

4.7. Аналитическая оценка выигрыша для JOIN

Условия выигрыша $\Delta = T_1 - T_2 > 0$:

$$\begin{cases} T_R > E_L \\ T_L > E_R \end{cases}$$

остаются справедливыми и для дерева выполнения с JOIN.

Чтобы получить оценку выигрыша в зависимости от соотношения между числом входных строк и строк результата JOIN, рассмотрим участвующие в этих неравенствах величины. Оценка даётся в предположениях:

– рассматривается линейное приближение работы операторов от числа строк в группе A;

– в результате соединения $A \bowtie B$ количество строк уменьшается в γ раз, причём результирующие строки распределены равномерно, так что каждое из поддеревьев $A_L \bowtie B$ и $A_R \bowtie B$ выдаёт количество строк, пропорциональное числу входных строк;

– группа B фиксирована.

T_L определяется: получением данных в поддереве $A_L \bowtie B$, выполнением соединения $A_L \bowtie B$, передачей результатов всего запроса сервису управления выполнением – DELIVER ($A \bowtie B$). В соответствии с этим:

$$T_L=\alpha_L*N_L+ \alpha_D*\gamma*(N_L+ N_R)$$

Здесь N_L – число строк в A_L, N_R – в A_R. Первое слагаемое соответствует поддереву $A_L \bowtie B$, второе – оператору DELIVER ($A \bowtie B$).

Основной вклад в T_R вносят: получение данных в поддереве $A_R \bowtie B$ и выполнение этого соединения: $T_R=\alpha_R*N_R$. Оператор EXCHANGE выполняется в обоих поддеревьях:

$$E_{R=} \beta_R*\gamma*N_R$$

$$E_{L=} \beta_L*\gamma*N_R$$

Обе части обрабатывают одно и то же число строк – результатов правой части $\gamma*N_R$.

В этих обозначениях условия выигрыша:

$$\begin{cases} \alpha_R * N_R > \beta_L * \gamma * N_R \\ \alpha_L * N_L + \alpha_D * \gamma * (N_L + N_R) > \beta_R * \gamma * N_R \end{cases}$$

Первое условие даёт ограничение сверху для γ:

$$\gamma < \frac{\alpha_R}{\beta_L}$$

Второе условие, сводящееся к:

$$\gamma * \left[\beta_R * \frac{N_R}{N_L} - \alpha_D * \left(1 + \frac{N_R}{N_L}\right)\right] < \alpha_L$$

определяет зависимость между γ и $\frac{N_R}{N_L}$ – разбиением строк из группы A между A_L и A_R.

5. Экспериментальное исследование выполнения запроса на двух серверах

В полученных соотношениях учитывается лишь необходимое для выполнения запросов процессорное время. Однако время выполнения зависит

также от сетевых задержек. Для учёта их влияния, а также в связи с отсутствием точных значений производительности операторов, проведено экспериментальное исследование запросов с оператором JOIN.

В экспериментах использовались массовые запросы следующего общего вида:

SELECT
    <столбцы из БД групп G1, G2>
FROM
    G1, G2
WHERE <условия выборки данных и соединения >

Условие выборки данных позволяет регулировать число строк, получаемых из баз группы G1, а условие соединения – число строк результата.

Рассмотренный в п. 4.6 способ оптимизации JOIN может быть применён для любого состава групп. В данном исследовании группа G1 состоит из двух баз: G1=DB1 ∪ DB2, а группа G2 – из одной, DB3. Массовый запрос преобразуется в следующий запрос OGSA-DQP:

SELECT
    <выбираемые столбцы глобальной схемы>
FROM
    <запрос с оператором DB1 ⋈ DB3>
    UNION ALL
    <запрос с оператором DB2 ⋈ DB3>
WHERE < условие выборки данных>

Оператор JOIN реализован как THETA JOIN. Дерево выполнения этого запроса соответствует приведённому на рис. 5.

Постановка экспериментов потребовала доработки аппарата выполнения запросов OGSA-DQP. В существующих версиях этой системы нет средств распределения запроса по нескольким серверам. Компонента, выполняющая эту функцию, реализована в виде нового оптимизатора, который подключается путём добавления дополнительного описателя в конфигурационный файл центрального сервера. Оптимизатор размещения выполняется на заключительном этапе построения дерева выполнения и производит разметку узлов этого дерева информацией о том, на каком сервере будет выполняться соответствующий оператор. Исходя из цели эксперимента, используется следующий алгоритм размещения.

1. Обход дерева начинается с корневого узла. Запрашивается первый сервер выполнения. Узел размещается на полученном сервере.
2. Проходятся все нижележащие узлы до первого бинарного. Все эти узлы размещаются на тот же сервер.
3. На этот же сервер распределяется левое поддерево первого бинарного узла.

4. Запрашивается ещё один сервер выполнения. На него распределяется правое поддерево.
5. Между первым бинарным узлом и его правым поддеревом вставляется оператор EXCHANGE.

Для рассматриваемых запросов первым от корня бинарным оператором является UNION ALL, и, таким образом, соединения DB1⋈ DB3 и DB2⋈ DB3 выполняются на разных серверах.

Эксперименты проведены в локальной сети с пропускной способностью 100 Мбит/сек. Среду обработки запросов образуют центральный сервер и два сервера выполнения. На всех серверах установлен комплекс OGSA-DAI/DQP (ogsadai-4.0-gt-4.2.0), на центральном сервере установлена, кроме того, система MQ-DAI. Характеристики серверов примерно одинаковые: Intel Core CPU 3.2 Ггц, 1 Гб, Scientific Linux 4.5.

Три используемые базы данных (MySQL 5.5.15) размещены на отдельных серверах (Intel Core CPU 2,9 Ггц, 1 Гб, Scientific Linux 4.5). Базы DB1 и DB2, входящие в группу G1 содержат по 60000 строк, база G3 – 15000 строк.

5.1. Результаты экспериментов

В экспериментах исследована зависимость выигрыша Δ=T1-T2, получаемого при выполнении запроса на двух серверах, от двух переменных Δ= Δ(I, O), где I – число строк, считываемых из каждой базы DB1 и DB2 (из левой части JOIN), O – число строк результата запроса. Число строк, выбираемых правой частью JOIN из базы DB3 фиксировано и равно 15000.

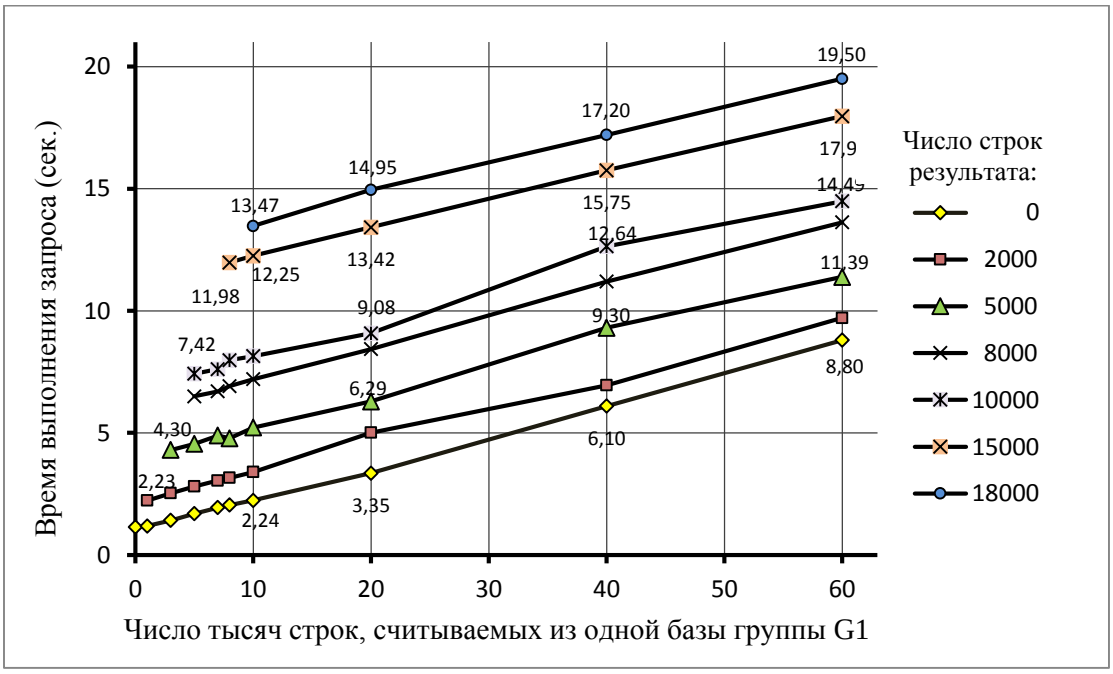

Рис. 6. Время выполнения JOIN на одном сервере в зависимости от параметра I.

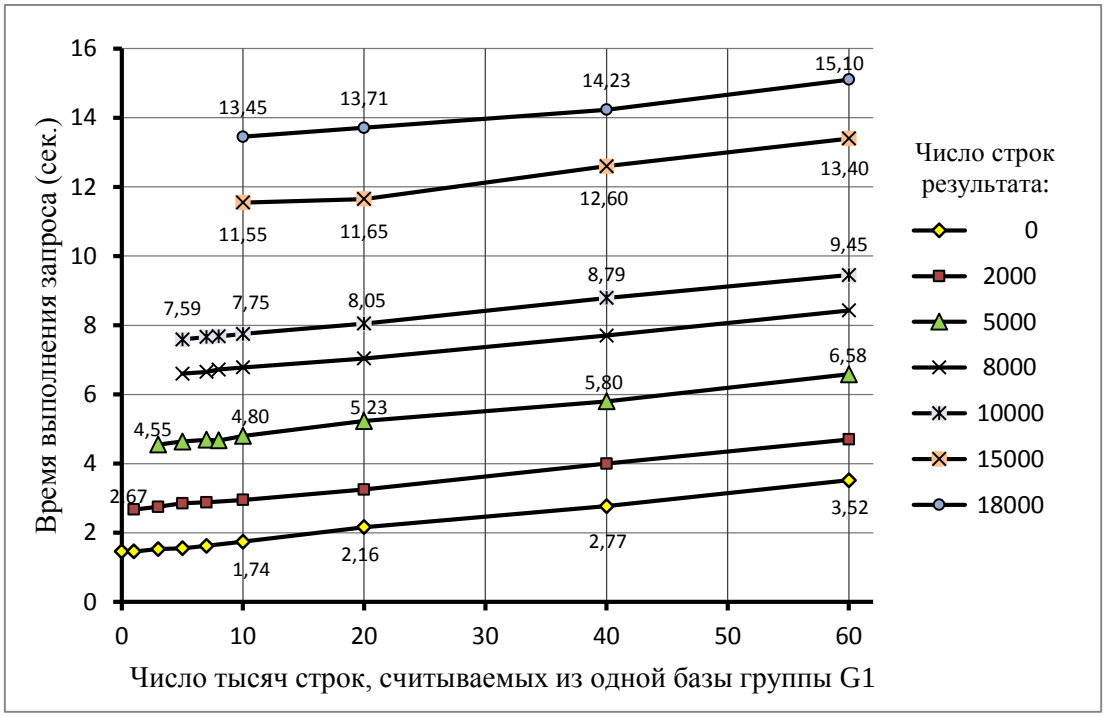

Рис. 7. Время выполнения JOIN на двух серверах в зависимости от параметра I.

На рис. 6 и 7 приведены зависимости времени выполнения запроса от I при разном числе строк результата O на одном и двух серверах соответственно. Значения времён выполнения в каждой точке получены путём 10-кратного выполнения запроса и вычисления среднего. Отклонения от среднего не превышают 10%.

Из графиков видно, что время выполнения T и на одном, и на двух серверах увеличивается от числа входных строк I и строк результата O. Зависимость T от I с хорошим приближением линейная, при этом значения T в точках пересечения линий тренда с вертикальной осью на графике для одного сервера меньше, чем соответствующие значения для двух серверов. Однако коэффициент линейной зависимости T(I) меньше на двух серверах, так что при I > 10000 выигрыш достигается. Такое поведение можно объяснить сетевыми задержками: при малом объёме входных данных они перевешивают процессорные затраты, а при большом – лимитирующим фактором становится процессор. Экспериментальные данные о выигрыше $\Delta = T1-T2$ приведены в таблице 3.

Во всей исследованной области выигрыш является возрастающей функцией от I и убывающей от O, хотя зависимость от O нарушается в ряде точек. При I<10000 выигрыша нет – Δ отрицательна или близка к нулю. При I>10000 выигрыш имеет место для всего исследованного диапазона O. Максимальный выигрыш достигается в точке I=60000, O=0 и равен 5.28 сек.

Таблица 3

Разность времён выполнения (сек.) на одном и двух серверах.
Единица параметров I, O – тысяча строк.

O / I	0	2	5	8	10	15	18
0	-0,32						
1	-0,27	-0,44					
3	-0,11	-0,22	-0,25				
5	0,14	-0,04	-0,09	-0,10	-0,17		
7	0,32	0,17	0,20	0,05	-0,05		
10	0,50	0,45	0,42	0,42	0,39	0,70	0,02
20	1,19	1,76	1,06	1,39	1,03	1,77	1,24
40	3,33	2,95	3,50	3,50	3,85	3,15	2,97
60	5,28	5,01	4,81	5,19	5,04	4,57	4,40

Представляет интерес величина относительного выигрыша

$\theta = \frac{T_1 - T_2}{T_1}$ (рис. 8).

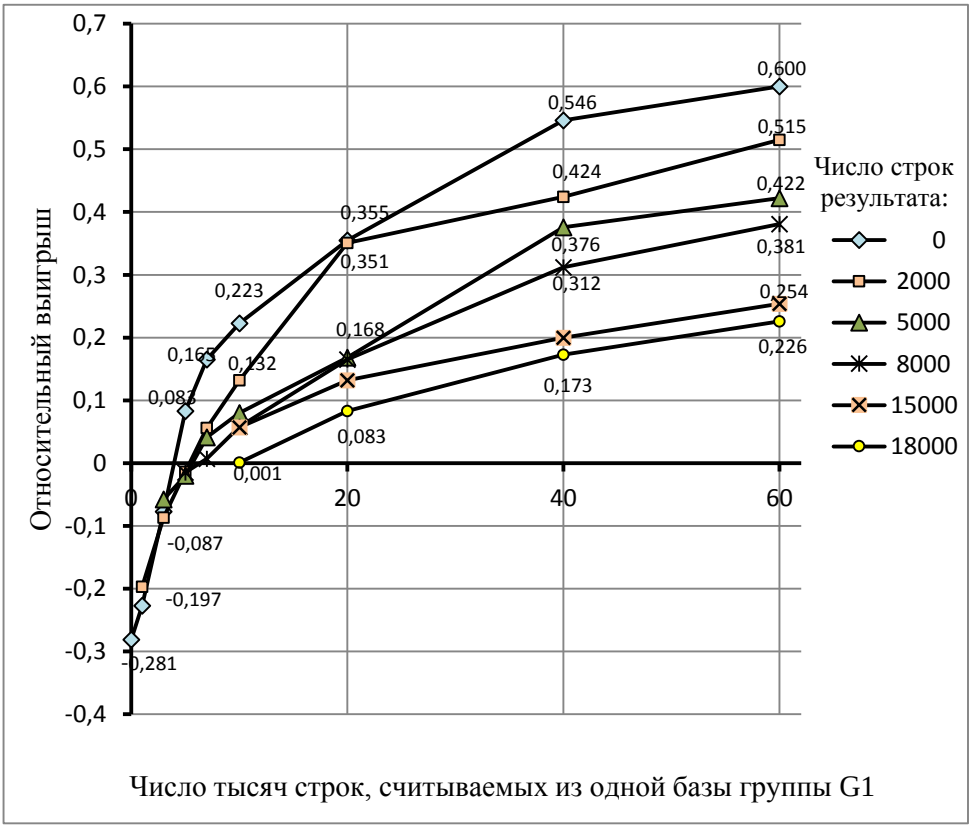

Рис. 8. Относительный выигрыш при выполнении JOIN.

Функция Θ(I,O) убывает при возрастании O, а при фиксированном O ведёт себя как дробно-линейная функция: нелинейно возрастает, а затем при больших

значениях I выходит на предельное значение, которое зависит от O и меняется от 0.6 (при O=0) до 0.23 (при O=18000).

6. Заключение

Улучшение технических характеристик (скорости обработки запросов, надёжности, масштабируемости) является актуальной проблемой для современного этапа развития систем виртуальной интеграции. Проблема усложняется в условиях крупных информационных инфраструктур, на которые ориентирована система MQ-DAI и используемый в ней язык массовых запросов. Одно из возможных направлений решений этой проблемы – оптимизация выполнения запросов.

Как наиболее существенные для массовых запросов рассмотрены два типа методов оптимизации: преобразования логического плана и параллельное выполнение. Среди методов первого типа выделены: исключение выборки избыточных столбцов и выталкивание условий отбора строк в базы данных. Экспериментально показано, что оба метода многократно (до 10 и более раз) сокращают время выполнения запроса в результате уменьшения количества обрабатываемых данных, причём эффекты применения этих методов суммируются.

Среди методов параллельного выполнения наиболее известен и получил реализацию конвейерный параллелизм. По имеющимся данным он даёт двукратное ускорение обработки и сокращает влияние сетевых задержек. Как перспективный оценивается также внутриоператорный параллелизм, в особенности для массовых запросов, в которых могут параллельно выполняться операции над фрагментами данных из разных баз.

Анализ процесса выполнения запросов в комплексе OGSA-DAI/DQP, являющегося базовым для MQ-DAI, показал, что для массовых запросов, в которых объём данных результата совпадает с объёмом данных, полученных из баз, внутриоператорный параллелизм не даёт выигрыша. Это объясняется реализацией операторов обмена данными между серверами выполнения посредством протокола SOAP, который затратен по нагрузке на процессор, но необходим для обеспечения независимости от серверных платформ.

Тем не менее внутриоператорный параллелизм способен дать эффект для массовых запросов, которые содержат оператор JOIN, благодаря тому, что этот оператор уменьшает объём обрабатываемых данных. Как показали эксперименты на стенде информационной инфраструктуры, внутриоператорный параллелизм даёт выигрыш в массовых запросах с оператором JOIN при объёме выбираемых из баз данных, большем 10000 строк. Выигрыш от распараллеливания является возрастающей функцией от объёма входных данных и убывающей от объёма результатов. Максимальное ускорение при выполнении на двух серверах достигает двух раз.

Таким образом, можно сделать следующие выводы. Наиболее значимыми методами оптимизации массовых запросов являются преобразования

логического плана. В имеющихся реализациях систем виртуальной интеграции набор таких преобразований далеко не исчерпывающий. Перспективной и практически важной для массовых запросов представляется, например, оптимизация агрегирующих функций SQL.

Внутриоператорный параллелизм способен обеспечить возможность работы системы интеграции с таблицами баз данных большого размера, не только ускоряя выполнение, но и уменьшая требования к оперативной памяти серверов. Но реализация такой оптимизации требует развития средств сбора статистики для оценки альтернативных вариантов выполнения.

7. Литература

[1]. Коваленко В.Н., Коваленко Е.И., Куликов А.Ю. Система массовой интеграции баз данных: функциональные возможности и способ реализации. // Труды 5-й международной конференции «Распределенные вычисления и Грид-технологии в науке и образовании», Дубна, 2012 г., с. 337-342.

[2]. Технология интеграции архивов медицинских изображений / С.В. Кирсанов [и др.] // Врач и информационные технологии, №2, 2013, с. 59-70.

[3]. Donald Kossmann. The State of the Art in Distributed Query Processing // ACM Computing Surveys, 32(4), December 2000, pp. 422-469, URL: http://ece.ut.ac.ir/dbrg/seminars/AdvancedDB/2006/Khalili-Amiri/Report1/p422-kossmann.pdf

[4]. Коваленко В.Н., Куликов А.Ю. Интеграция данных и язык запросов в масштабных информационных инфраструктурах // Программные продукты и системы. № 3, 2012, с. 124-130.

[5]. Distributed Data Management with OGSA–DAI / Michael J. Jackson [etc.] // Grid and Cloud Database Management (eds. Fiore, S. and Aloisio, G.), Springer-Verlag, July 2011, pp. 63-86.

[6]. Integrating distributed data sources with OGSA–DAI DQP and Views / Dobrzelecki, B. [etc.] // Phil. Trans. R. Soc. A. Vol. 368, no. 1926, 13 September 2010.

[7]. Web Services Data Access and Integration - The Core WS-DAI Specification / Antonioletti M. [etc.] // Open Grid Forum, 2012.

[8]. Wiederhold, G., Genesereth, M. The conceptual basis for mediation services // IEEE Expert. 12, 1997, pp. 38–47.

[9]. Lenzerini M. Data Integration: A Theoretical Perspective // PODS 2002, pp. 233–246.

[10]. A Novel Approach to Resource Scheduling for Parallel Query Processing on Computational Grids / Anastasios Gounaris [etc.] // Distributed and Parallel Databases Journal 19(2-3), 2006, pp. 87-106.

[11]. David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems // Commun. ACM 35, 6 (June 1992), pp. 85-98.

[12]. Experience on Performance Evaluation with OGSA-DQP / Nedim Alpdemir [etc.] // Proc. of Fourth UK e-Science All Hands Meeting, September 2005. URL: http://www.cs.man.ac.uk/~rizos/papers/ahm05b.pdf
[13]. Graefe, G. Iterators, schedulers, and distributed-memory parallelism // Software-Practice and Experience 26(4), 1996, pp. 427–452.
[14]. Fan Zhu. Parallel Processing of JOIN Queries in OGSA-DAI // MS Thesis, Dept. EPCC, The University Of Edinburgh, Edinburgh, Aug 21, 2009.
[15]. Resource Scheduling for Parallel Query Processing on Computational Grids / Anastasios Gounaris [etc.] // 5th IEEE/ACM International Workshop on Grid Computing, GRID'04, 2004.
[16]. Fan ZHU, Mingqiang ZHOU. Parallelization for Complex Query Based on OGSA-DAI // Journal of Computational Information Systems Vol.8, no. 12, 2012, pp. 4843-4853, URL: http://www.Jofcis.com.
[17]. Joshua Eke. OGSA-DAI parallel hash joins using Bonfire // Master's thesis, The University of Edinburgh, 2011.

Оглавление