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

Препринты ИПМ • Препринт № 25 за 2016 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Коваленко В. Н., Коваленко Е. И.

Оптимизация поисковых запросов с агрегирующими функциями в условиях массовой виртуальной интеграции баз данных

Рекомендуемая форма библиографической ссылки: Коваленко В. Н., Коваленко Е. И. Оптимизация поисковых запросов с агрегирующими функциями в условиях массовой виртуальной интеграции баз данных // Препринты ИПМ им. М.В.Келдыша. 2016. № 25. 18 с. doi:10.20948/prepr-2016-25
URL: http://library.keldysh.ru/preprint.asp?id=2016-25

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 поисковых запросов с агрегирующими функциями в условиях массовой виртуальной интеграции баз данных

Москва — 2016

Коваленко В.Н., Коваленко Е.И.

Оптимизация поисковых запросов с агрегирующими функциями в условиях массовой виртуальной интеграции баз данных

Принцип виртуальной интеграция предполагает, что при выполнении запроса данные из нескольких баз передаются по сети на сервер системы интеграции, где происходит их совместная обработка. Одно из возможных направлений оптимизации запросов – сокращение объёма передаваемых данных. В данной работе предлагается способ оптимизации для запросов, вычисляющих агрегирующие функции языка SQL – сумму, среднее значение и прочее. Представлены алгоритмы оптимизирующего преобразования над планом выполнения запроса и обеспечения вычисления функций непосредственно в базах данных. Результаты экспериментов показали высокую эффективность оптимизации такого вида.

Ключевые слова: интеграция баз данных, оптимизация запросов, агрегирующие функции

Kovalenko Victor Nikolaevich, Kovalenko Evgeniia Ivanovna

Optimization of search queries with aggregation functions for the conditions of mass virtual integration of databases

The concept of virtual integration assumes that during a query execution the data from several databases are transferred via network to the integration system server where they are processed jointly. One of the possible directions of query optimization is reduction of the transmitted data amount. In this paper a method of optimization for the queries which calculate the aggregation functions of the SQL language (the sum, average et al.) is offered. Algorithms of optimizing transformation over the query execution plan and those ensuring the function evaluation directly in databases are presented. The experimental results have shown high efficiency of the proposed method.

Key words: database integration, query optimization, aggregation functions

1. Введение

Работа продолжает исследования [1] по оптимизации поисковых запросов в системах, с помощью которых осуществляется массовая интеграция автономных реляционных баз данных. В результате интеграции образуется общее информационное пространство, а рассматриваемые запросы дают возможность получать данные сразу из множества баз, хотя они могут отличаться СУБД и представлением данных. Интеграция является виртуальной: данные не помещаются в общее хранилище (физическая интеграция), при выполнении запросов они извлекаются непосредственно из каждого источника и приводятся к единой форме.

Реализованная в ИПМ им. М.В. Келдыша РАН система MQ-DAI [2] поддерживает массовую виртуальную интеграцию, позволяя создавать масштабные информационные инфраструктуры из большого числа (порядка 100) автономных баз. Инфраструктурный подход поддерживается в MQ-DAI в трёх аспектах.

В языке массовых поисковых запросов [3], который является расширением SQL, введена новая конструкция – группа. С помощью этой конструкции в массовом запросе определяется множество опрашиваемых баз. Состав группы формируется отбором баз по приписанным им содержательным атрибутам (местоположению, организационной принадлежности, тематике и т.д.).

Реализован механизм интеграции данных, то есть унификации представлений однотипных данных в разных базах. Используется известный подход на основе глобальной схемы [4], но разработан вариант, в котором унификация представлений определяется независимо для каждой базы. Благодаря этому изменения в составе баз и в их организации не требуют модификации приложений, работающих с инфраструктурой.

Управление инфраструктурой в большой степени децентрализовано: в MQ-DAI имеются средства управления, которые позволяют их владельцам включать в инфраструктуру новые базы (а также отключать их) без нарушения её нормального функционирования.

Работа в основном связана с первым аспектом и касается способа интерпретации языка массовых запросов. Виртуальная интеграция обладает рядом достоинств по сравнению с физической [5,6], но время выполнения запросов при виртуальной интеграции становится существенно большим, чем при работе с единственной базой – хранилищем, и это становится критическим фактором её практической применимости. В связи с этим для виртуальной интеграции разработан репертуар методов оптимизации запросов [7,8], причём они отличны от методов, используемых в традиционных СУБД.

В этом ряду и тема работы – оптимизация массовых запросов с агрегирующими функциями. Агрегирующие функции – сумма элементов данных, среднее значение, стандартное отклонение и пр. – полезные и широко используемые в приложениях конструкции языка SQL. Помимо того, наличие у них свойства дистрибутивности делает их перспективными с точки зрения

оптимизации: дистрибутивность означает, что вычисление функции над массивом данных может быть разбито на независимые части, выполняющиеся над фрагментами исходного массива. Это свойство прямо соотносится с массовыми запросами, в которых имеется естественное разбиение данных: по базам, в которых они содержатся.

В работе рассматривается вопрос, каким образом свойство дистрибутивности может быть использовано для ускорения выполнения массовых запросов. В варианте решения, представленном далее, излагаются метод и алгоритмы преобразования запроса на основе дистрибутивности, а также способ выталкивания функций в базы данных. Приведены данные экспериментов, показывающих высокую эффективность рассматриваемого подхода.

2. Свойство дистрибутивности агрегирующих функций

Как показано в ряде работ [9,10], в системах виртуальной интеграции основные затраты времени на обработку запросов связаны с передачей данных по сети. Хотя современные СУБД выполняют запросы быстро, при использовании платформенно независимых протоколов типа SOAP в таких системах большое количество процессорного времени тратится на преобразования получаемых из СУБД данных. Уменьшить потери помогают методы блокирования строк и различные методы параллельного выполнения [11].

Другой класс методов использует для оптимизации сокращение объёма передаваемых по сети данных. К таким методам относится, например, исключение выборки из баз избыточных, то есть не нужных для вычисления результата запроса, строк и столбцов. Эффективность этого вида оптимизации весьма значительна [1]. В данной работе рассматривается метод, основанный на сокращении передач данных, для оптимизации запросов, которые содержат агрегирующие функции.

Как известно, агрегирующие функции SQL – это функции, получающие на вход столбец данных, по которому вычисляется единственная выходная величина. Состав агрегирующих функций в разных реализациях SQL отличается, здесь мы ограничимся следующими: MAX (максимальное значение), MIN (минимальное значение), COUNT (количество элементов), SUM (сумма), AVG (среднее значение), STDDEV (стандартное отклонение), VARIANCE (дисперсия).

Предлагаемый способ основан на том, что часть агрегирующих функций обладает свойством дистрибутивности. Под дистрибутивностью функции F понимается возможность её вычисления следующим образом:

$$F(X) = F1(F2(x_1), F2(x_2), \dots, F2(x_n)) \qquad (1)$$

F1 и F2 это также агрегирующие функции, $\{x_1, x_2, \dots, x_n\}$ – любое разбиение входного массива данных $X= x_1 \cup x_2 \dots \cup x_n$, $x_i \cap x_j = \emptyset$ при $i \neq j$. Функция

F является дистрибутивной, если равенство выполняется при любых разбиениях X и значениях данных.

Из рассматриваемого набора свойством дистрибутивности обладают первые 4 функции:

SUM (X) = SUM (SUM (x_1), SUM (x_2), … , SUM (x_n))
MAX (X) = MAX (MAX (x_1), MAX (x_2), … , MAX (x_n))
MIN (X) = MIN (MIN (x_1), MIN (x_2), … , MIN (x_n))
COUNT (X) = SUM (COUNT (x_1), COUNT (x_2), …, COUNT (x_n))

Свойство дистрибутивности даёт возможность вычислять функцию по частям. При реализации агрегирующих функций в традиционных СУБД это свойство не столь важно: входной массив непосредственно доступен целиком. В системах виртуальной интеграции один запрос выполняется над данными из множества распределённых баз, и ситуация осложняется необходимостью пересылок данных. В этом случае последовательность действий для вычисления функции включает:

- выборку данных, требуемых для вычисления функции, из каждой базы и передачу их на сервер выполнения запросов;
- объединение поступающих фрагментов в общий массив на сервере выполнения;
- вычисление функции.

Передача данных из баз на сервер обусловливает основные временные потери. При вычислении дистрибутивных функций эти потери можно исключить практически полностью. Рассматриваемый способ оптимизации заключается в том, чтобы вместо извлечения из всех баз частей массива аргументов функции F вычислять непосредственно в каждой базе скалярные величины F2 (x_i) и только их передавать на сервер. Значение исходной функции F тогда вычисляется функцией F1 согласно правой части формулы (1). Эффект такой оптимизации значительный – из каждой базы передаётся лишь одна величина.

Свойством дистрибутивности обладают не все агрегирующие функции. Для остальных из рассматриваемых (AVG, STDDEV, VARIANCE) известны разные способы вычислений, однако существуют и такие, которые позволяют их вычислять, исходя из дистрибутивных функций:

AVG (X) = SUM (X)/ COUNT (X)
VARIANCE = AVG ((X – AVG(X)) **2)
STDDEV = SQRT (VARIANCE)

Оптимизация агрегирующих функций, которые входят в правые части этих формул, даёт нужный эффект и для вычисляемых таким образом функций.

3. Дистрибутивное преобразование запросов с агрегирующими функциями

Агрегирующие функции вычисляются в контексте запроса SELECT. Рассматриваемый далее способ оптимизации (далее дистрибутивная оптимизация) предполагает преобразование исходного запроса в соответствии с описанным выше математическим аппаратом.

Приведём пример массового запроса с агрегирующей функцией:

SELECT F(Expression) FROM Group.Table WHERE Condition (2)

Синтаксис языка массовых запросов в основном совпадает с синтаксисом SQL 92. Основные отличия языка массовых запросов – адресация элементов данных в терминах глобальной схемы и выполнение запроса над множеством баз. Запрос в примере выдаёт значение функции F, аргументом которой является выражение (Expression). Выражение вычисляется по значениям атрибутов глобальной таблицы Table, которые извлекаются из группы баз Group и удовлетворяют условиям в части WHERE. В примере указана единственная функция в части SELECT, однако подход применим и к более общей ситуации, когда в ней содержится несколько функций.

Для определения дистрибутивного преобразования существенен вопрос его встраивания в общую схему обработки запросов в системе массовой интеграции. Как показано на рис. 1, она состоит из трёх компонентов: упомянутой выше системы MQ-DAI, которая надстроена над системами OGSA-DQP и OGSA-DAI [12,13]. Каждая из этих систем имеет собственный язык запросов, и массовый запрос в процессе обработки подвергается последовательности преобразований.

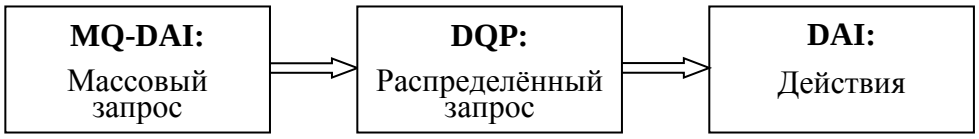

Рис. 1. Последовательность преобразований массового запроса.

Система MQ-DAI, принимающая массовый запрос, преобразует его в распределённый запрос системы DQP. Так как в массовом запросе данные адресуются в терминах унифицированного представления – глобальной схемы, осуществляется локализация данных, то есть вместо символических конструкций – групп подставляется совокупность подзапросов, получающих соответствующие данные из всех баз группы (при этом используются отображения глобальной схемы на схемы баз). Полученный в результате преобразованный запрос передаётся системе DQP, которая обеспечивает его выполнение, опираясь на язык действий системы DAI.

Вопрос встраивания в эту схему решён применением дистрибутивной оптимизации к распределённому запросу на языке системы DQP, который получен из массового запроса. Соответствующий массовому запросу (2) запрос DQP имеет вид:

```
SELECT F (Expression) FROM
 (SELECT Attr FROM Request_DB1
  UNION ALL
  …                                                    (3)
  SELECT Attr FROM Request_DBn
 ) AS G
 WHERE Condition
```

- F обозначает вычисляемую в массовом запросе функцию.
- Expression – аргумент функции F. В простейшем случае это столбец, а в общем случае это может быть арифметическим выражением от нескольких столбцов.
- Attr – получаемые из баз DB1, …, DBn столбцы. Они выбираются запросами SELECT, которые обозначены Request_DBi. Каждый такой запрос, во-первых, получает данные из одной базы и, во-вторых, преобразует их из фактического представления в ней в представление глобальной схемы.
- G – имя таблицы, которая получается в результате объединения выбранных данных. Аргумент Expression функции F вычисляется на столбцах этой таблицы.

Заметим, что оптимизация может применяться и непосредственно к запросам системы DQP такого вида.

Дистрибутивное преобразование запроса DQP основывается на описанном выше способе вычисления функций с помощью разбиения данных на части (в запросе (3) это разбиение уже получено). Целью оптимизации является построение запроса (4), в котором агрегируются значения вспомогательных функций, полученные из отдельных баз:

```
SELECT F1 (G.f2) FROM
      (SELECT F2 (Expression^T) AS f2 FROM
            (SELECT Attr FROM Request_DB1)
       WHERE Condition^T
       UNION ALL                                       (4)
       …
       SELECT F2 (Expression^T) AS f2 FROM
            (SELECT Attr FROM Request_DBn)
       WHERE Condition^T
      ) AS G
```

Соответствие между исходной функцией F и функциями F_1, F_2, которые используются для её вычисления, представлено в таблице 1.

Таблица 1

Соответствие исходной агрегирующей функции и функций в оптимизированном запросе

Исходная функция (F)	**F_1**	**F_2**
SUM	SUM	SUM
MAX	MAX	MAX
MIN	MIN	MIN
COUNT	SUM	COUNT

Сравнение запросов (3) и (4) показывает следующее. Части, отвечающие за получение данных из баз, совпадают. Имя f_2 – любой уникальный идентификатор. Требуется преобразование имен элементов данных в аргументе $Expression^T$ функции F_2 и в условии выбора строк $Condition^T$. Такая необходимость связана с тем, что в выражении $Expression^T$ запроса (4) строки заданы в терминах глобальной схемы, в то время как в Expression исходного запроса (3) им соответствуют переименованные в результате объединения имена. Аналогичная ситуация с переходом от условия выборки строк Condition к $Condition^T$, здесь переименование выполняется в обратную сторону.

Содержание дистрибутивной оптимизации составляет преобразование текста исходного запроса (3) в оптимизированный запрос (4). Преобразование производится с использованием дерева синтаксического разбора AST (Abstract Syntax Tree) [14], в котором в явном виде представлены все структурные составляющие запросов. Реализация преобразования выполнена с помощью инструментальных средств системы DQP, в числе которых построение AST из текста запроса (а также его частей: выражений и условий) и наоборот, получение запроса по его AST, а также выделение и замена отдельных частей запроса

Алгоритм дистрибутивного преобразования в целом выглядит следующим образом:

- верификация исходного запроса на соответствие паттерну запроса (3). Ключевые конструкции проверки: наличие агрегирующей функции, наличие одного или нескольких операторов UNION ALL;
- построение AST исходного запроса;
- выбор в зависимости от имени функции в исходном запросе соответствующих функций оптимизированного запроса.
- построение выражений и условий выборки данных для оптимизированного запроса;
- построение текста оптимизированного запроса в соответствии с

шаблоном.

Хотя дистрибутивное преобразование определено для достаточно простого примера, оно допускает обобщение, в том числе для случаев, когда в исходном запросе вычисляется список функций, когда функции входят в состав выражений. Это, в частности, позволяет вычислять агрегирующие функции, которые не обладают свойством дистрибутивности. Приведём, например, оптимизированный запрос для функции AVG (X) = SUM (X)/ COUNT (X).

```
SELECT sum/count FROM
(SELECT SUM (G.f2) AS sum, SUM (G.f3) AS count FROM
    (SELECT SUM (Expression^T) AS f2, COUNT (Expression^T) AS f3
        FROM (SELECT Attr FROM Request_DB1)
    WHERE Condition^T                                          (5)
    UNION ALL
        …
    SELECT SUM (Expression^T) AS f2, COUNT (Expression^T) AS f3
        FROM (SELECT Attr FROM Request_DBn)
    WHERE Condition^T
    ) AS G
)
```

4. Выталкивание функций в базы данных

Программная реализация модуля дистрибутивной оптимизации агрегирующих функций позволила получить количественную оценку её эффективности. Результаты экспериментов, представленные далее в разделе 7, показывают, что этот вид оптимизация даёт существенный эффект: время выполнения уменьшается в 2-3 раза. Однако имеется сильная зависимость времени выполнения от количества данных, используемых при вычислении функции. Наличие такой зависимости говорит о том, что они передаются на сервер выполнения в полном объёме. Как показал анализ базовой системы DQP, это объясняется тем, что функции выполняются на сервере системы даже в том случае, когда их аргументы содержатся в единственной базе. Наблюдаемый, тем не менее, эффект оптимизации даёт конвейерный параллелизм, задействованный в DQP.

Для полной реализации потенциала дистрибутивной оптимизации требуется “выталкивание” функций в базы данных. Под выталкиванием имеется в виду следующее. Рассмотренный выше способ дистрибутивной оптимизации вычисляет результирующее значение функции путём разбиения исходного запроса на несколько подзапросов, которые получают данные только из одной базы:

```
SELECT F2 (Expression^T) AS f2 FROM
    (SELECT Attr FROM Request_DBi)                    (6)
WHERE Condition^T
```

Так же как и исходный запрос, каждый из подзапросов содержит функцию, но поскольку её аргументы содержатся в одной базе, появляется возможность исключить передачу “сырых” данных на сервер, выполняя каждый подзапрос (6) непосредственно в соответствующей базе. В этом случае время выполнения будет минимальным и равным времени вычисления и передачи значения функции.

Реализация такого “выталкивания” функций не может основываться на преобразовании текста запроса как в случае дистрибутивной оптимизации: требуется изменение способа выполнения агрегирующих функций в базовом для MQ-DAI комплексе OGSA-DAI/DQP. В этой связи приведём наиболее существенные аспекты процесса подготовки выполнения запросов в этом комплексе.

Начальный этап подготовки – построение плана выполнения (Logical Query Plan – LQP) [15], выполняет система DQP. План строится по синтаксическому дереву запроса AST и также представляет собой дерево, однако его узлам соответствуют операторы нескольких типов, задающие определённые действия над данными. Большую их часть составляют операторы реляционной алгебры. Из дополнительных операторов наибольший интерес для нашей задачи представляют операторы SCALAR_GROUP_BY, вычисляющий агрегирующие функции, и TABLE_SCAN, который содержит запрос, посылаемый в базу данных.

Структуру дерева выполнения задают двунаправленные связи между узлами предок ↔ потомок. Связи соответствуют потокам данных, которые передаются от потомка к предку. Сами узлы аннотированы информацией, необходимой для их интерпретации. Прежде всего, это тип оператора, схемы (имена столбцов) входных и выходных данных. Кроме того, для каждого типа операторов имеются специфические аннотации, в том числе:

- для оператора RENAME – отображение имён схемы входных данных на схему выходных данных;
- для оператора PROJECT – список имён выбираемых столбцов;
- для оператора SELECT – условие отбора строк;
- для оператора SCALAR_GROUP_BY – вычисляемая функция вместе с её аргументом (может быть арифметическим выражением);
- для оператора TABLE_SCAN – SQL-запрос к базе данных.

Важной особенностью подготовки выполнения является то, что вначале по синтаксическому дереву строится исходный вариант LQP, который затем улучшается последовательно применяемыми оптимизаторами. Примером оптимизации может служить план, содержащий оператор TABLE_SCAN, которому предшествует оператор SELECT. Если SQL-запрос к базе данных в операторе TABLE_SCAN имеет вид:

```
SELECT * FROM table
```

а предшествующий оператор SELECT содержит логическое условие ("age < 20"), оператор SELECT удаляется, и условие переносится в TABLE_SCAN. SQL-запросом к базе данных становится:

SELECT * FROM table WHERE age < 20

Похожее оптимизирующее преобразование применяется, когда перед TABLE_SCAN стоит оператор PROJECT. В этом случае модифицируется список SELECT SQL-запроса – из него удаляются лишние столбцы. Отметим, что в архитектуре DQP предусмотрен специальный механизм подключения новых оптимизаторов, а в имеющейся версии их 8.

План LQP содержит всю необходимую для выполнения информацию, но представлена она в абстрактной форме, которая оставляет свободу в выборе способа выполнения. В этой части система DQP опирается на систему DAI [13], преобразуя план LQP в язык действий DAI. Действия DAI реализуют реляционные операторы, передачу данных и их преобразование, а цепочки связанных по данным действий позволяют определить обработку данных из нескольких баз. Система DAI представляет собой программную среду интерпретации цепочек действий, которая основана на многопоточной реализации итераторной конвейерной модели [16]. Достоинство этой модели заключается в том, что программные интерфейсы действий построены на унифицированных принципах, благодаря чему выходные данные одного действия могут быть переданы любому другому. Модель способствует параллельному выполнению действий в форме конвейерного параллелизма: каждое действие порождает одну строку за такт работы и передаёт его следующим действиям в цепочке.

Преобразование плана LQP в действия DAI составляет заключительный этап подготовки выполнения запроса. Преобразование осуществляется путём обхода узлов плана, при этом, во-первых, каждый оператор плана отображается в соответствующую цепочку действий. Во-вторых, для каждого действия задаются входные и выходные данные, и тем самым определяются цепочки действий, связанных по передаваемым данным.

В этом контексте задача выталкивания функций в базы данных решается путём модификации двух этапов: построения плана выполнения запроса и преобразования плана выполнения в действия.

5. Преобразование плана выполнения

Рассмотрим преобразование плана LQP для выталкивания функций в базы, используя в качестве иллюстрации рис. 2. Изображенный на нём пример LQP соответствует (с некоторыми упрощениями) массовому запросу:

SELECT SUM (nmb) AS sum FROM G1.persons WHERE nmb<10

после того, как выполнена дистрибутивная оптимизация.

Вычислению функций в LQP соответствуют узлы с оператором SCALAR_GROUP_BY, которые аннотированы агрегирующими функциями. Функции представлены именем и аргументом – в общем случае арифметическим выражением. Используемые в арифметическом выражении атрибуты (имена столбцов таблиц) определяют входные данные оператора.

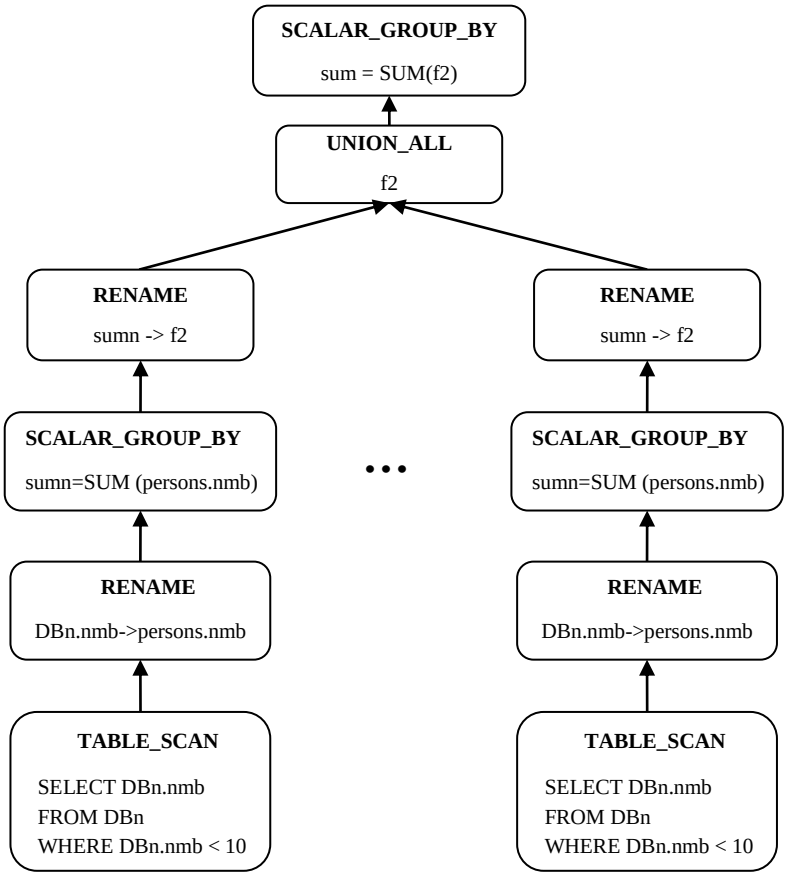

Рис. 2. План выполнения, построенный
после дистрибутивной оптимизации.

Операторы SCALAR_GROUP_BY, как и все другие операторы LQP, выполняются на сервере системы интеграции. Однако если все входные данные получены из единственной базы, функция может быть вычислена непосредственно в ней. Каждый оператор SCALAR_GROUP_BY выделяет в дереве выполнения поддерево, вершиной которого он является. Концевые узлы поддерева (листья) представляют собой операторы TABLE_SCAN, определяющие множество источников данных для оператора вычисления функции. Критерий возможности выталкивания – единственный источник в поддереве от SCALAR_GROUP_BY (на рисунке это операторы в ветвях, лежащих ниже оператора UNION_ALL).

Цель преобразования LQP для выталкивания функций – замена оператора SCALAR_GROUP_BY на оператор TABLE_SCAN, выходными данными которого является значение функции, а не исходные данные для её вычисления.

Преобразование состоит, во-первых, в модификации SQL-запроса к базе, и, во-вторых, в согласовании выходных данных модифицированного TABLE_SCAN с входными данными предка оператора SCALAR_GROUP_BY. Задача модификации SQL-запроса не простая по причине того, что цепочка от SCALAR_GROUP_BY до TABLE_SCAN может содержать любые операторы, с помощью которых получаются атрибуты, используемые в функции. Для массовых запросов, например, цепочка реализует отображение глобальной схемы в схему базы данных.

Алгоритм преобразования состоит из следующих основных шагов.

- Обнаружение в LQP множества $\{s_j\}$ операторов SCALAR_GROUP_BY, которые удовлетворяют сформулированному критерию, а также множества $\{t_j\}$ соответствующих каждому s_j операторов TABLE_SCAN. Для выполнения этого шага в системе DQP имеются программные методы навигации по дереву LQP, замены его узлов и поиска операторов определённого типа. Далее для каждого s_j выполняются следующие два шага.
- По цепочке операторов от s_j до t_j строится модифицированный SQL-запрос. Построение производится последовательно, начиная с s_j. Каждый очередной оператор либо переименовывает атрибуты списка SELECT, либо включает в часть FROM запроса некоторый подзапрос.
- Цепочка $[s_j \ldots t_j]$ заменяется на новый оператор TABLE_SCAN с модифицированным SQL-запросом, который вычисляет значение функции, и выходными данными, соответствующими s_j.

Применительно к рассматриваемому примеру, каждая из ветвей от оператора UNION_ALL заменяется на цепочку (рис. 3):

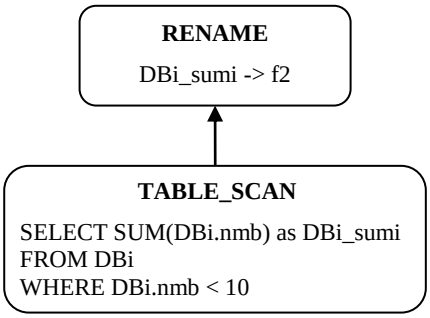

Рис. 3. Преобразованная ветвь LQP с вычислением функции в базе данных.

6. Реализация выталкивания функций в базы данных

В аспекте реализации рассматриваемого преобразования LQP возникает проблема представления модифицированного оператора TABLE_SCAN, который содержит вычисление функции. Проблема возникает из-за того, что формат узла TABLE_SCAN в имеющейся версии DQP не предполагает хранения SQL-запроса в явном виде. В узле отдельно хранятся список извлекаемых столбцов, имена таблиц, которым они принадлежат, и предикаты

отбора строк. При преобразовании TABLE_SCAN в действия системы DAI штатный конструктор строит запрос к базе данных, компонуя его из отдельных частей, но представление функций в формате TABLE_SCAN не предусмотрено.

Решается эта проблема следующим образом. При построении узла TABLE_SCAN, который содержит обращение к функции, в этот узел добавляется аннотация специального типа (с помощью имеющегося в DQP метода addAnnotation). В качестве значения в аннотацию записывается в явном виде построенный модифицированный SQL-запрос.

Кроме того, соответственно произведена модификация конструктора действий. Она заключается в том, что при построении действий для каждого оператора TABLE_SCAN проверяется наличие дополнительной аннотации. Если она есть, в действие передаётся явно заданный запрос, в противном случае вызывается штатный конструктор.

Таким образом, реализация выталкивания функций состоит из двух компонентов.

- Компонента преобразования LQP представляет собой новый оптимизатор, который построен в соответствии с интерфейсами системы DQP и подключается к процессу обработки запросов стандартным для оптимизаторов способом. Его назначение – обнаружение паттерна, удовлетворяющего условиям выталкивания, построение модифицированного запроса к базе данных в операторе TABLE_SCAN и модификация плана выполнения.
- Модифицированный конструктор действий для оператора TABLE_SCAN меняет способ преобразования его в действия при наличии в SQL-запросе функции. В системе DQP предусмотрен способ подключения конструкторов в последовательность обработки запросов, аналогичный подключению оптимизаторов, – путём задания имени конструктора в конфигурационном файле.

7. Оценка эффективности оптимизации агрегирующих функций

Описанные компоненты оптимизации запросов с агрегирующими функциями (дистрибутивный оптимизатор, оптимизатор модификации оператора TABLE_SCAN и конструктор для TABLE_SCAN) реализованы на уровне прототипа. Реализация имеет ограничения на сложность обрабатываемых запросов, однако она позволила получить экспериментальную оценку эффективности разработанного метода.

Эксперименты проведены в программной среде, включающей систему MQ-DAI, комплекс OGSA-DAI/DQP со встроенными в них компонентами оптимизации. В эксперименте измерялось время выполнения запроса вида (2) на конфигурации из двух баз данных (СУБД MySQL). Три графика на рис. 4 демонстрируют зависимость времени выполнения от основного фактора – количества данных, используемых при вычислении функции.

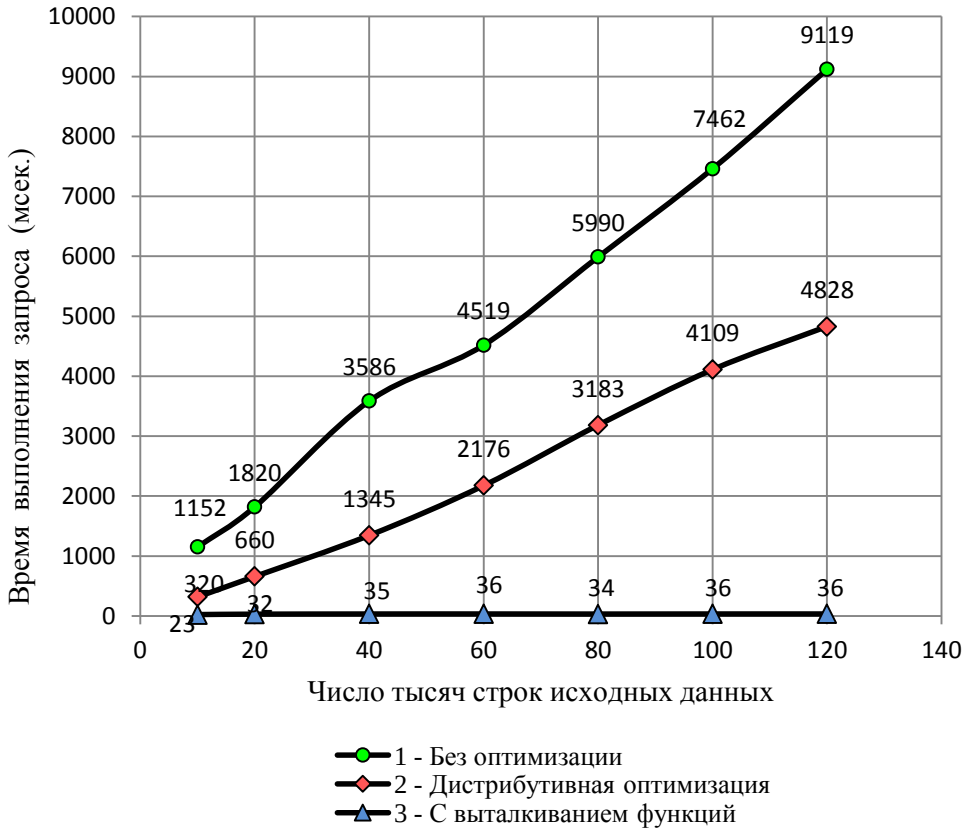

Рис. 4. Зависимость времени выполнения запросов с агрегирующей функцией от объёма исходных данных, необходимых для её вычисления.

Первый график соответствует выполнению запроса без оптимизации агрегирующих функций. Два других – выполнению запроса с разной степенью оптимизации. На графике 2 представлены результаты для запроса, оптимизация которого ограничена только дистрибутивной, а на графике 3 дистрибутивная оптимизация дополнена выталкиванием функции в базу данных.

Сравнение графиков 1 и 2 показывает, что дистрибутивная оптимизация даёт существенный эффект: время выполнения сокращается в 2-3 раза при разных объёмах исходных данных. В то же время наличие на графике 2 сильной зависимости от количества данных говорит о том, что они передаются на сервер выполнения. Имеющееся, тем не менее, ускорение можно отнести на счёт параллельного выполнения, которое поддерживается в DQP.

Время выполнения на графике 3 радикально снижается, меняясь от 23 до 36 мсек, а зависимость от количества данных становится не столь значительной, как на графиках 1 и 2.

8. Заключение

В работе описан способ оптимизации поисковых запросов с агрегирующими функциями. Задача решается в контексте систем виртуальной интеграции и в условиях, когда запросы являются массовыми, то есть

выполняются над данными из множества баз. Предлагаемое решение направлено на то, чтобы исключить сетевую передачу большого объёма данных из баз на сервер выполнения запросов, что является основным фактором, определяющим время их выполнения.

Показано, что этой цели можно достичь за счёт сочетания двух приёмов: преобразования запроса с использованием свойства дистрибутивности агрегирующих функций и поддержки вычисления функций непосредственно в каждой базе, с тем чтобы на сервер передавались лишь вычисленные значения.

Соответствующие алгоритмы разработаны с ориентацией на конкретную среду обработки запросов, состоящую из системы массовой интеграции MQ-DAI и комплекса OGSA-DAI/DQP. Однако процесс обработки в этой среде построен на основе общепринятых принципов, и, тем самым, полученные результаты могут найти более широкое применение.

Программная реализация компонентов оптимизации позволила получить экспериментальную оценку эффективности подхода, подтвердив предположение, что в результате оптимизации время выполнения запроса с агрегирующими функциями становится примерно таким же, как у запроса, получающего из каждой базы одну строку минимального объёма.

9. Литература

[1]. 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

[2]. 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.

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

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

[5]. Enterprise Information Integration: Successes, Challenges and Controversies / Alon Y. Halevy [etc.] // SIGMOD '05 Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ACM New York, NY, USA ©2005, ISBN:1-59593-060-4.

[6]. Jens Bleiholder, Felix Naumann. Data fusion // ACM Computing Surveys (CSUR), v.41 n.1, p.1-41, December 2008.

[7]. 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

[8]. Grinev M., Kuznetsov S. Towards an Exhaustive Set of Rewriting Rules for XQuery Optimization: BizQuery Experience // 6th East-European Conference on Advances in Databases and Information Systems (ADBIS) LNCS 2435, 2002, p. 340-345.
[9]. 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/~gounaris/dqp-ahm2005.pdf
[10]. The design and implementation of OGSA-DQP: A service-based distributed query processor / Steven Lynden [etc.] // Future Generation Computer Systems, Volume 25, Issue 3, March 2009, P. 224-236,
URL: http://www.cs.man.ac.uk/~norm/papers/fgcs09.pdf
[11]. A Novel Approach to Resource Scheduling for Parallel Query Processing on Computational Grids / Anastasios Gounaris [etc.] // Distributed and Parallel Databases Journal, 19(2-3): 87-106 (2006).
[12]. OGSA-DAI. URL: https://sourceforge.net/projects/ogsa-dai/
[13]. 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.
[14]. Абстрактное синтаксическое дерево.
URL: https://ru.wikipedia.org/wiki/Абстрактное_синтаксическое_дерево
[15]. Logical Query Plan. URL: http://en.wikipedia.org/wiki/Query_plan.
[16]. Graefe, G. Iterators, schedulers, and distributed-memory parallelism // Software-Practice and Experience 26(4), 427–452 (1996).

Оглавление