Способ совмещения межоперационной и внутриоперационной форм параллелизма запросов в базах данных

Ю. А. Шичкина[1], М. С. Куприянов[2]
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)
[1]strange.y@mail.ru, [2]mikhail.kupriyanov@gmail.com

Armando de Jesús Plasencia Salgueiro[3]
Instituto de Cibernética, Matemática y Física, La Habana, Cuba
[3]ymauri@gmail.com, armando@icimaf.cu

Аннотация. Развитие многоядерных и многопроцессорных вычислительных систем казалось бы должно значительно ускорить процесс обработки данных из различных типов баз данных. В данной статье описаны проблемы, возникающие при попытке распараллелить обработку данных в реляционных базах данных. Для решения одной из этих проблем представлен подход к совмещению межоперационной и внутриоперационной форм параллелизма запросов, который основан на теории ориентированных графов. Суть его состоит в построении графа по межоперационной форме параллелизма запроса, а потом его преобразовании с учетом фрагментации отношений и получения внутриоперационной формы.

Ключевые слова: база данных; запрос; параллельные вычисления; оптимизация; межоперационная и внутриоперационная формы параллелизма; фрагментация отношений

I. Введение

Современные реляционные СУБД имеют как правило высокоразвитые оптимизаторы запросов. Так SQL Server обеспечивает параллельную обработку запросов, оптимизирующую выполнение запросов и операции с индексами на компьютерах, где установлено несколько микропроцессоров (ЦП). Во время оптимизации запроса SQL Server пытается обнаружить запросы и операции с индексами, которые можно ускорить за счет параллельного выполнения. Для таких запросов SQL Server вставляет в план выполнения операторы обмена, чтобы подготовить запрос к параллельной обработке. После вставки операторов обмена получается план параллельного выполнения запроса [1].

PostgreSQL может вырабатывать планы запросов, которые будут задействовать несколько CPU, чтобы получить ответ на запросы быстрее. Но, разработчики утверждают, что для многих запросов параллельное выполнение не даёт никакого выигрыша, либо из-за ограничений текущей реализации, либо из-за принципиальной невозможности построить параллельный план, который был бы быстрее последовательного [2].

В MySql с параллелизмом запросов дела обстоят хуже. Сами разработчики утверждают, что параллелизм – это одна из сложнейших проблем для СУБД [3]. Параллелизм запросов в MySql осуществляется за счет разбиения запроса по разным клиентам и подключениям.

Таким образом, складывается ситуация, когда увеличение объемов реляционных баз данных и требований к скорости обработки данных происходит намного быстрее, чем развитие методов ускорения процесса обработки данных, в том числе методов автоматического построения параллельных запросов.

Исследования в данной области проводятся непрерывно. Так, в [8] приводится анализ того, какие виды параллелизма существуют в запросах, правда, не дается каких-либо рекомендаций по их реализации. Авторы [4] рассматривают узкий класс SQL-запросов, включающих операции объединения и группировки. Эти исследования опираются на теорему о достаточных условиях для предгруппового преобразования [9]. При этом остаются в стороне вопросы, касающиеся вложенности подзапросов. Схожие исследования, но с добавлением операторов сортировки проводятся авторами [7].

Проблема минимизации времени отклика на запрос при многопоточном соединении с использованием межоператорного параллелизма в параллельной и распределенной средах рассматривается в [6]. Задача планирования запросов в этой среде представляется как «проблема максимизации потока», а ее решение дает алгоритм, который статически находят оптимальный набор планов для запросов, запускаемых в многопоточных соединениях. Если же запрос один, но большой, то вопрос его разложения по потокам авторами не затрагивается.

Ускорению обработки запросов с применением кэша посвящена работа [5], в которой представлен алгоритм выбора кэша, оценки стоимости его применения в определенных условиях, адаптации кэша при изменении условий. Сам подход к оптимизации с применением кэша основывается на последовательных планах в форме деревьев.

Публикация выполнена в рамках международного гранта РФФИ (18-57-34001 Куба_т)

В целом, несмотря на разнообразие и количество проводимых исследований в области усовершенствования процесса получения данных из различных баз данных, на текущий момент методик и программного обеспечения по оптимизации схем хранения данных и эквивалентному преобразованию запросов к форме, наиболее эффективно реализуемой на вычислительных системах, допускающих параллельное исполнение запросов, очень мало и они далеки от совершенства. Что касается оптимизации запросов к реляционным базам данных, то в этой области исследования превалирует модель для преобразования SQL-запросов в структуру дерева запросов [10].

В данной статье рассматривается ряд проблем, с которыми приходится сталкиваться любому исследователю при построении плана параллельного запроса и общий подход к решению этих проблем.

II. Некоторые проблемы при построении параллельного плана запроса

Проблема 1. Выбор формы параллелизма. Как известно [11] существует несколько форм параллелизма, применяемых в параллельных СУБД.

Рис. 1. Классификация форм параллельной обработки транзакци

Когда у пользователя один, но сложный по составу запрос и применяемый к большому объему данных, то речь может идти только о внутризапросном параллелизме. При этом пользователю приходится решать, что будет лучше: применить межоперационный или внутриоперационный параллелизм. Межоперационный параллелизм предполагает параллельное выполнение реляционных операций, принадлежащих одному и тому же плану запроса. Горизонтальный параллелизм предполагает параллельное выполнение независимых поддеревьев дерева, представляющего план запроса. Основная проблема, связанная с горизонтальным параллелизмом, заключается в том, что очень трудно обеспечить, чтобы два подплана одного плана начали генерировать выходные данные в правильное время и в правильном темпе. Вертикальный (конвейерный) параллелизм предполагает организацию параллельного выполнения различных операций плана запроса на базе механизма конвейеризации. Основным недостатком синхронного конвейера является блокирующий характер операций конвейерной обработки отдельных порций данных.

Внутриоперационный параллелизм реализуется в основном в форме фрагментного параллелизма [12], который предполагает разбиение на непересекающиеся части отношения, являющегося аргументом реляционной операции [13].

Проблема 2. Определение независимых операций в запросе. Традиционно для решения этой задачи применяется дерево запроса. Актуальной задачей в рамках этой проблемы является выбор равных по времени выполнения подзапросов. Особенно, это важно, если результаты этих подзапросов являются аргументами общего запроса.

Проблема 3. Выбор формы фрагментации отношения. В реляционных системах баз данных фрагментация подразделяется на вертикальную и горизонтальную [14]. В этом свете проблема в свою очередь делится на две: выбор формы фрагментации и выбор способа фрагментации. После чего при вертикальной фрагментации появляется задача слияния результатов, а при горизонтальной фрагментации более остро стоит проблема деления на равные фрагменты.

Проблема 4. Совмещение межоперационного и внутриоперационного параллелизма. От того, в какой степени и в каких точках запроса будет применяться та или иная форма параллелизма, зависит ускорение выполнения запроса. Причем практика показывает, что ускорение может быть и отрицательным.

Проблема 5. Оценка накладных расходов. В отличии от классических алгоритмов величина накладных расходов может быть в запросах очень большой за счет обрабатываемых объемов данных и специфики работы СУБД. Так, например, запрос:

```
Select атрибуты
From Table1                                   (1)
Where (условие 1) and (атрибут<=значения);
```

выполняется значительно медленнее запроса

```
Select атрибуты
From Table1                                   (2)
Where (условие 1).
```

Поэтому разделение запроса (2) на два запроса (1) при горизонтальной фрагментации будет эффективным только при объеме данных больше некоторой константы, которую вычислить можно статистически.

Проблема 6. Статическая оценка ускорения запроса при различных формах параллелизма, формы кэширования, индексации, действий встроенного оптимизатора СУБД. Для идеального плана параллельного выполнения запроса необходимо проводить полноценное исследование с оценкой степени влияния различных параметров на скорость выполнения запроса и размер накладных расходов. Делать это для произвольного единоразового запроса не имеет смысла, т.к. этот процесс является слишком трудоемким.

Проблема 7. Выбор оптимального плана с учетом решения проблемы 6.

Методов, позволяющих решить эти проблемы полностью, однозначно и универсально для любого запроса к любой реляционной базе данных на произвольной вычислительной системе, не существует. Но в теории параллельных вычислений есть аппарат и методы [15, 16], которые позволяет свести эти проблемы к минимуму.

III. Применение теории графов для совмещения меж- и внутриоперационного параллелизма

Для построения параллельного плана запросов в первом приближении можно использовать метод распараллеливания классических алгоритмов, основанный на списках смежности информационного графа алгоритма [15]. Под информационным графом запроса мы будем далее подразумевать ориентированный граф, в котором вершинами являются подзапросы, и направленное ребро соединяет вершины a и b, если подзапрос, соответствующий вершине b в качестве входных данных использует результат запроса, соответствующего вершине a. Т.к. в большинстве своем запросы к реляционным базам данных очень редко достигают степени вложенности больше трех, редко содержат более 10 ярусов, то построить начальный информационный граф можно вручную. При этом, все методы оптимизации информационного графа по различным параметрам (времени, вычислительным узлам, объему межпроцессорных передач и т.д.), разработанные для классических алгоритмов, полностью подходят для запросов. Но, при всех качествах и возможностях этих методов, запросы к реляционным базам данных имеют свою специфику и поэтому непосредственное применение перечисленных методов в лучшем случае даст некоторое ускорение, а в худшем случае не даст никакого эффекта. В первую очередь это происходит из-за того, что входными данными для запросов являются целые таблицы, и они содержат очень большой объем данных. Во вторых информационный граф дает информацию только о параллелизме по задачам, а запросы, которые бы содержали очень много независимых подзапросов, на практике встречаются редко. Поэтому часто по информационному графу в первом приближении запас внутреннего параллелизма равен нулю.

Рассмотрим несколько вариантов запроса, состоящего из предложений select, from, where.

Вариант 1. Запрос S осуществляется только к одной таблице T. Запрос S является подзапросом к другим запросам $S_1..S_k$, $k \in N$. Информационный граф такого запроса представлен на рис. 2а.

Для начала рассмотрим случай, когда запрос S не содержит функции агрегирования avg.

Распараллеливание запроса будем производить путем деления таблицы T по некоторому атрибуту на две таблицы T_1 и T_2: $T=T_1 \cup T_2$, где символом «$\cup$» обозначается операция union в SQL. Тогда информационный граф запроса изменится и примет вид графа с рис. 2б.

Рис. 2. Графы с целыми и разделенными таблицами запроса

Варианты запроса S' в зависимости от выражения в предложении Select представлены ниже в таблице 1.

ТАБЛИЦА I Соответствие начальной и конечной форм запроса

№	Предложение Select	S'
1.	Select max(d1)…	Select max(@m1,@m2);
2.	Select min(d1)…	Select min(@m1,@m2);
3.	Select sum(d1)…	Select sum(@m1,@m2);
4.	Select count(d1)…	Select sum(@m1,@m2);

Теперь, будем полагать, что в запросе S нет функции агрегирования. Тогда запросу S в соответствии с графом с рис. 1б) будет соответствовать запрос S':

Select @m1 union @m2;

Рассмотрим последний вариант простого запроса – случай, когда запрос S содержит функцию агрегирования avg. Такие запросы также попадают под схему с рис. 2б. Им будет соответствовать S':

Select (@m1+@m2)/(@c1+@c2);

Таким образом, независимо от вида функции агрегирования и ее наличия в предложении Select, все запросы вида «select … from T where…» попадают при горизонтальной фрагментации под схему с рис.2б.

Вариант 2. Запрос S осуществляется только к нескольким таблицам T_1 и T_2:

Select атрибуты
From T_1, T_2
Where (условие) and (T_1.key=T_2.key);

Запрос S является подзапросом к другим запросам $S_1..S_k$, $k \in N$. Информационный граф такого запроса также будет соответствовать рис.1а.

Пусть каждая из таблиц T_1 и T_2 разбита каким-то способом (неважно в настоящем контексте каким именно) на две части: $T_1= T_{11} \cup T_{12}$, $T_2= T_{21} \cup T_{22}$. Тогда по свойству естественного соединения получаем:

$$T_1 \times T_2 = (T_{11} \cup T_{12}) \times (T_{21} \cup T_{22}) = T_{11} \times T_{21} \cup T_{12} \times T_{21} \cup T_{11} \times T_{22} \cup T_{12} \times T_{22}.$$

В соответствии с этим правилом запрос S будет преобразован к форме S':

Select @m_{11} union Select @m_{12} union Select @m_{21} union Select @m_{22};

Соответственно граф запроса тогда примет вид:

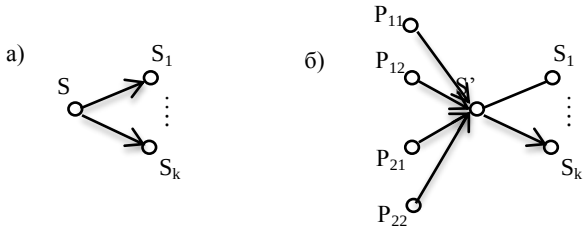

Рис. 3. Графы с целыми и разделенными таблицами запроса

Пусть k – число делимых таблиц, t – число ярусов в информационном графе запроса, r – текущий ярус графа, n_r – число запросов на ярусе r, m_r – число запросов варианта 1, k_i – число частей в i-й таблице, k_g – число вершин, добавляемых к исходному графу. Тогда:

$$k_g = \sum_{r=1}^{t}\left(\sum_{i=1}^{m_r} k_{ri} + \sum_{i=m_r+1}^{n_r}\sum_{j=m_r+1}^{n_r} k_{ri}k_{rj}\right)$$

На рис. 4 представлены графики роста общего числа вершин в зависимости от сложности начального графа и начального числа вершин в графе при делении каждой таблицы всего лишь на две части. Под сложностью понимается произведение значений: числа ярусов, среднего числа вершин на ярусе и среднего числа таблиц в запросах.

Рис. 4. График роста сложности графа запроса

По рис. 4. видно, что с ростом сложности графа число вершин в модифицированном графе с учетом разделенных на части таблиц очень сильно возрастает. Вручную построить такой граф очень трудоемко, а проанализировать его на предмет построения оптимального параллельного плана выполнения запроса еще сложнее. В этом случае на помощь приходят методы параллельных вычислений [15, 16].

IV. Заключение

Проведенный анализ существующих решений в области распараллеливания запросов показал, что сегодня существует множество частных решений как полученных отдельными исследователями, так и организациями и реализованные в различных программных продуктах. Эти частные решения не позволяют большинству пользователей с легкостью решать свои задачи по ускорению запросов к реляционным базам данных. В статье выделены наиболее значимые проблемы, для решения которых требуется универсальный аппарат. Одним из таких аппаратов может быть теория ориентированных графов. В этом случае методы, разработанные для распараллеливания классических алгоритмов и предназначенные для их оптимизации по числу узлов, по времени выполнения, по числу межпроцессорных объемов и т.д., подходят и для оптимизации запросов. Но для их применения граф исходного запроса должен быть модифицирован. В результате граф запроса позволяет совместить межоперационный и внутриоперационный параллелизм. А добавление к графу весов, позволяет учесть такие параметры, как индексы, накладные расходы, объем данных и прочее.

Список литературы

[1] https://technet.microsoft.com/ ru-ru/ library/ ms178065 (v=sql. 105).aspx

[2] https://postgrespro.ru/docs/postgresql/9.6/parallel-query

[3] https://www.seagate.com/ru/ru/our-story/data-age-2025/

[4] M.Al. Hajj Hassian and M. Bamba, Parallel processing of Group by join queries on nothing shared machines // ICSOFT 2006, Software and Data Technologies, pp 230-241.

[5] S. Babu, K. Munagala, J. Widom, and R. Motwani. Adaptive caching for continuous queries. In ICDE, 2005, pp.118-129.

[6] A.Dasphande, L.Hellester, Flow algorithm for parallel query optimization. // In:24th International Conference on Data Ingineering , pp.754-763. IEEE,Cancun, 2008.

[7] R.Acker, C.Roth, R.Bayer, Parallel query processing in databases on multicore architecture ICA3PP 2008: Algorithms and Architectures for Parallel Processing pp 2-13

[8] P.Mohankumar, P.Kumaresan and Dr.J.Vaideeswaran, Optimism analysis of parallel queries in databases through multicores // International Journal of Database Management Systems (IJDMS), Vol.3, No.1, February 2011, pp.156-164

[9] Tsois, A., Sellis, T.K.: The generalized pre-grouping transformation: Aggregate-query optimization in the presence of dependencies. In: VLDB, pp. 644–655 (2003)

[10] Spiliopoulou M., Hatzopoulos M., Ttanslation of SQL queries into a graph structure: query transformations and pre-optimization issues in a pipeline multiprocessor environment // Informafion Sysfems 1992, Vol. 17, No. 2, pp. 161-170.

[11] Л.Б. Соколинский. Параллельные системы баз данных. Издательство МГУ, 2013, 184 с.

[12] DeWitt D.J., Gray J. Parallel Database Systems: The Future of HighPerformance Database Systems // Communications of the ACM. 1992. Vol. 35, No. 6. P. 85-98.

[13] Graefe G. Query evaluation techniques for large databases // ACM Computing Surveys. 1993. Vol. 25, No. 2. P. 73–169.

[14] Williams M.H., Zhou S. Data Placement in Parallel Database Systems // Parallel database techniques. // IEEE Computer society, 1998. P. 203–218.

[15] Shichkina, Y.A., Kupriyanov, M.S. Applying the list method to the transformation of parallel algorithms into account temporal characteristics of operations // SCM 2016, 7519759, c. 292-295

[16] Shichkina, J., Degtyarev, A., Kulik, B., Fridman, A. Optimization of relational databases schemas by means of n-tuple algebra // AIP Conference Proceedings, 1863, 110008