Шичкина Ю.А.
Куприянов М.С.
Заводчикова М.Г.

Способы распараллеливания сложных запросов в клиент-серверных СУБД на примере MySQL

Аннотация. Все известные на сегодняшний день подходы к параллельной обработке запросов основываются на разделении нагрузки между узлами системы в формировании исполнения запросов от нескольких приложений-клиентов. Учет особенностей структуры запроса уже в начальной стадии компиляции позволяет достичь ряда преимуществ. В статье приводится пример распараллеливания сложного запроса и описываются возможности применения теории графов и методов параллельных вычислений как для распараллеливания запроса, так и для его оптимизации.

Введение.

На сегодняшний день использование баз данных и информационных систем являются неотъемлемой частью функционирования предприятий и организаций. В связи с этим большую актуальность приобретает адаптация существующих принципов построения и эффективного применения технологий и программного обеспечения для управления базами данных применительно к вычислительным системам различной архитектуры. Проблема повышения производительности СУБД в настоящее время продолжает оставаться актуальной из-за постоянного повышения требований к объемам обрабатываемых данных. Возникновение сверхбольших баз данных приводит к расширению видов и сфер применения СУБД.

Рынок программного обеспечения ПК располагает большим числом разнообразных по функционалу возможностям коммерческих систем управления базами данных для всех массовых моделей машин и для различных операционных систем.

К наиболее популярным клиент-серверным СУБД сегодня относятся: Microsoft SQL Server, Oracle, Firebird, PostgreSQL, MySQL.

Одни из этих СУБД обладают более мощным функциональным набором, другие менее укомплектованы различными функциями по обработке данных, но принцип работы всех перчисленных СУБД одинаков.

В клиент-серверных СУБД обработка данных ведется на сервере, в том же месте, где хранятся данные. Приложения-клиенты посылают запросы на обработку и получения данных из СУБД и получают ответы. Приложения-клиенты не имеют непосредственного доступа к файлам данных.

Сервер базы данных представляет собой мультипользовательскую версию СУБД, параллельно обрабатывающую запросы, поступившие со всех рабочих станций. В его задачу входит реализация логики обработки транзакций с применением необходимой техники синхронизации - поддержки протоколов блокирования ресурсов, обеспечение, предотвращение и/или устранения тупиковых ситуаций.

В ответ на пользовательский запрос рабочая станция получит не «сырье» для последующей обработки, а готовые результаты. Программное обеспечение рабочей станции при такой архитектуре играет роль только внешнего интерфейса (Front - end) централизованной системы управления данными. Это позволяет существенно уменьшить сетевой трафик, сократить время на ожидание блокированных ресурсов данных в мультипользовательском режиме, разгрузить рабочие станции и при достаточно мощной центральной машине использовать для них более дешевое оборудование. Но это не позволяет распределить части запроса между вычислительными ядрами для их параллельного исполнения.

Способы хранения таблиц в СУБД MySQL

СУБД MySQL является решением для малых и средних приложений и используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой двух типов таблиц: MyISAM, поддерживающие полнотекстовый поиск и InnoDB, поддерживающие транзакции на уровне отдельных записей.

Таблицы MyISAM прекрасно подходят для использования в интернете и других средах, где преобладают запросы на чтение.

MyISAM не поддерживает транзакции и с этим связаны его основные преимущества, такие как: скорость обработки запросов, объем таблиц, объем памяти при обновлениях таблиц и другие. Основными недостатками MyISAM являются, например, блокировка всей таблицы при обновлении только одного поля, слабая реализация сортировки и низкая скорость при высокой нагрузке (порядка 400 клиентов, исполняющих сложные запросы по базе данных размером 2-3 ГБ).

InnoDB - одна из выбираемых подсистем низкого уровня в СУБД MySQL, входит во все стандартные сборки для различных операционных систем. Основным отличием InnoDB от других подсистем низкого уровня MySQL является наличие механизма транзакций и внешних ключей. Достоинтвами InnoDB являются поддержка транзакций, построчная

блокировка при выполнении функции UPDATE и высокая скорость при сложных запросах. не блокирует всю таблицу;

Формат InnoDB был разработан для максимальной эффективности при обработке больших объемов данных. Следовательно, он обеспечивает MySQL транзакционно-безопасным драйвером таблицы с поддержкой обработки нескольких запросов сразу, обратной перемотки и возможности восстановления после аварийного отказа. InnoDB обеспечивает блокировку на уровне строки и также обеспечивает непротиворечивым чтением без блокировки в операторах SELECT, который увеличивает параллелизм транзакции.

Таким образом InnoDB для решения задач, требующих параллельной реализации подходит больше.

Способы распараллеливания запросов в клиент-серверных базах данных

Одним из способов достижения более высокой производительности является использование алгоритмов распараллеливания заданий. В СУБД существует три области применения таких алгоритмов:

- параллельный ввод/вывод,
- параллельные средства и утилиты администрирования,
- параллельная обработка запросов к базе данных.

Распараллеливание ввода/вывода в сочетании с оптимальным планированием заданий позволяет осуществить весьма эффективный одновременный доступ к фрагментированным таблицам и индексам, расположенным на нескольких физическим дисках, тем самым многократно ускоряя операции с относительно медленными внешними устройствами.

В отличие от параллельных ввода/вывода и администрирования, параллелизм в обработке запросов реализуется гораздо сложнее. Теоретическим обоснованием возможности распараллеливания запросов в реляционной СУБД является свойство реляционной замкнутости - результатом каждого реляционного оператора: SELECT - выборка подмножества строк отношения (таблицы); PROJECT - выборка подмножества полей (столбцов); JOIN - соединение двух таблиц, является новое отношение, а поскольку любой запрос можно разбить на иерархию элементарных операторов, то разумно попытаться выполнить их параллельно. Языку запросов SQL, несомненно, внутренне присущ параллелизм. Обработка запроса обычно состоит из множества атомарных операций, состав и последовательность которых, определяется оптимизатором после просмотра нескольких вариантов.

В MySQL запросы выполняются параллельно только от разных клиентов. Это значит, что MySQL не может распараллеливать выполнение одного запроса на нескольких процессорах. Поэтому, для увеличения

производительности запроса следует оптимизировать «тяжелые» запросы. Чтобы оценить временные затраты при выполнении запросов разных по структуре, но приводящих к одному и тому же набору данных, можно использовать в MySQL утилиту MySQLSlap, которая предназначена для тестирования нагрузки MySQL сервера.

Запрос вывода номера пользователя, электронной почты, номер темы, сообщение, название темы, номер темы и Ф.И.О пользователя с помощью MySQLSlap (листинг 1) занимает 32,77 секунд.

Листинг 1.

SELECT u.id_user, u.email, p.id_post, p.message, t.topic_name, t.id_topic, u.name FROM users u, topics t, posts p WHERE u.id_user BETWEEN 100 and 900 and t.id_topic=p.id_topic GROUP BY u.name.

После разбиения этого запроса на 2 запроса время сократилось до 30,6 секунд, что на 2,1 секунд быстрее первого запроса.

Большинство запросов с вложенными конструкциями SELECT можно во-первых реализовать по разному. И среди всех реализаций будут те котрые, выполняются быстрее и те, которые выполняются медленнее. Во-вторых, выше приведенный пример показывает, что разбиение одного сложного запроса на несколько взаимосвязанных простых запросов позволяет запускать часть выделенных подзапросов независимых друг от друга как от разных клиентов параллельно, что приводит к ускорениию обработки данных.

Деление запросов можно проводить вертикально, горизонтально и смешивая эти способы.

Под горизонтальной декомпозицией запроса подразумевается декомпозиция отношений, к которым применяется запрос. Сам запрос при этом остается неизменным (рис.1b). На рисунке 1a) приведен изначальный запрос.

Под вертикальной декомпозицией запроса понимается декомпозиция самого запроса на отдельные подзапросы, при этом сами отношения, к которым применялся изначальный запрос остаются неизменными (рис.1c).

В зависимости от структуры основного запроса, возможно проведение либо только горизонтльной декомпозиции, либо только вертикальной, а в лучшем случае смешенный вариант одновременного применения вертикальной и горизонтльной декомпозиций (рис.1d).

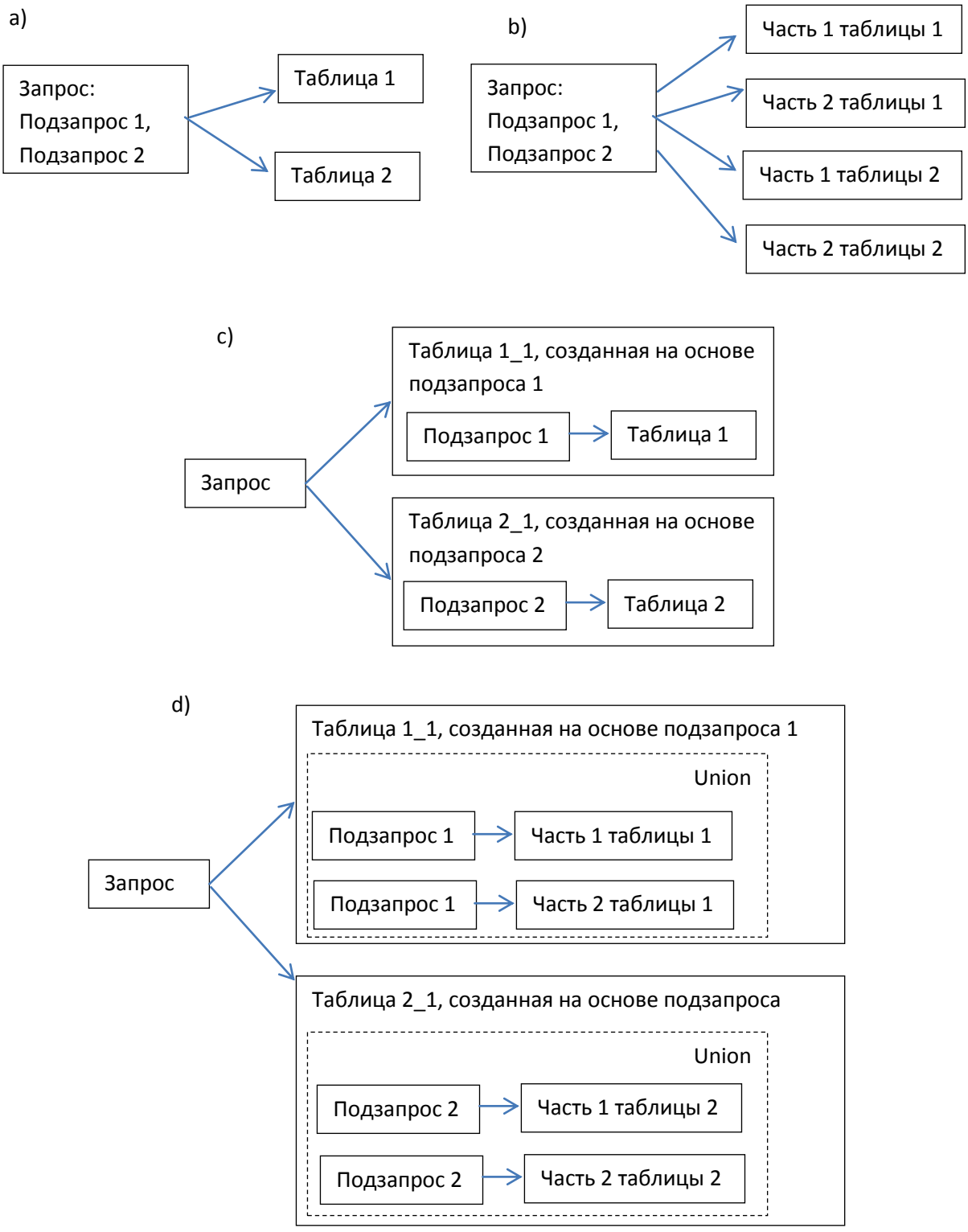

Рисунок 1. Модификация запросов

Более наглядное представление взаимосвязей частей запроса и отношений дает аппарат теории графов. На рис. 2a представлен граф, соответствующий вертикальному преобразованию запроса с рис. 1c. Рисунок 2b соответствует смешанной декомпозиции (рис.1d). Можно провести дальнейшую более глубокую декомпозицию отношений (рис.2c).

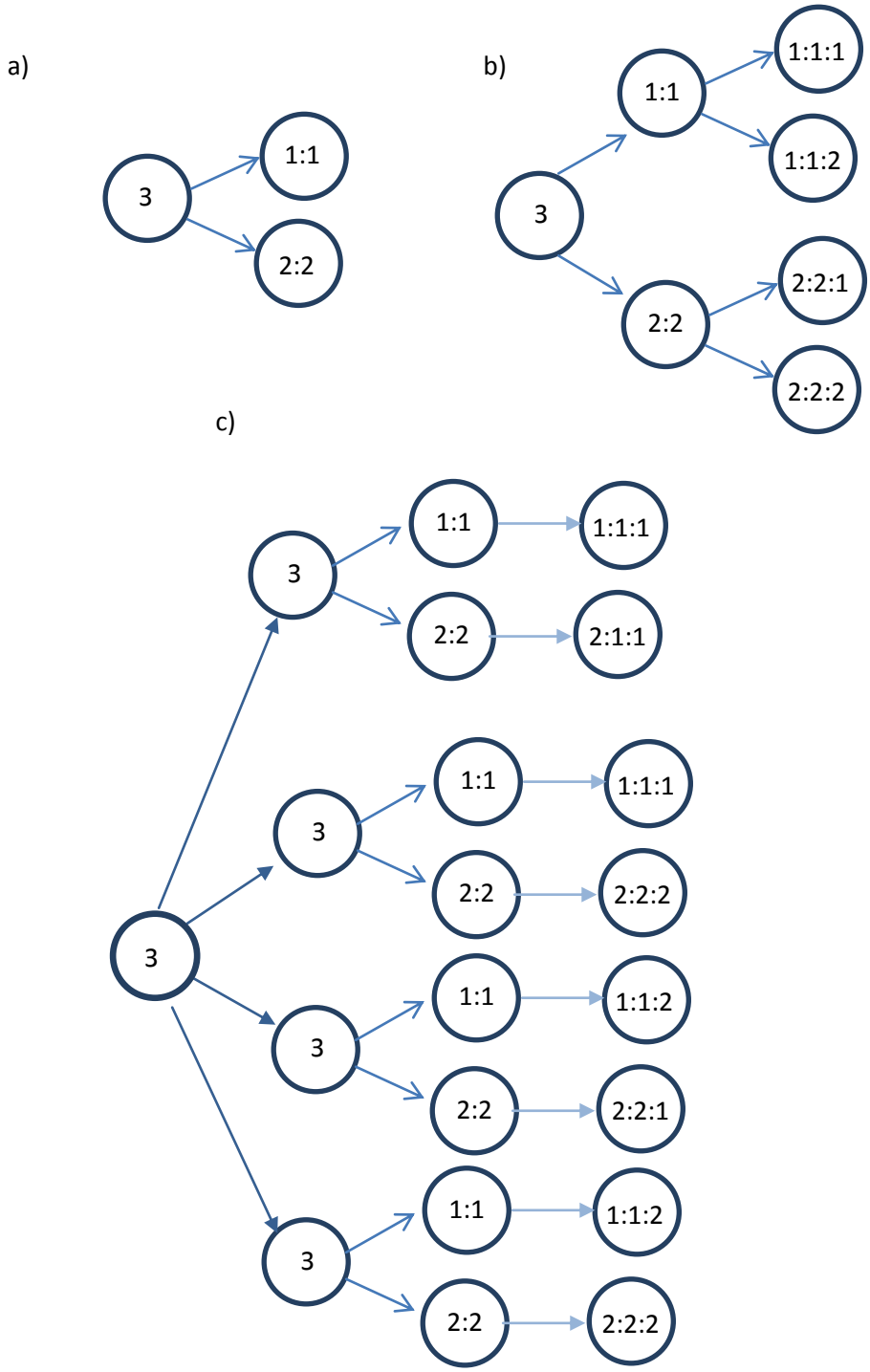

Рисунок 2. Представление запросов с помощью графов

Графы позволяют не только визуализировать сами запросы, но и с помощью матричных методов оптимизации параллельных алгоритмов определить степенб внутреннего параллелизма, необхожимый объем вычислительных ресурсов, построить новую модель паралллеьного запроса.

Заключение

Исследования показали, что если представлять части сложного запроса как отдельные клиентские запросы, то независимо от SQL-сервера средствами стандартного SQl расспараллеливание сложных запросов возможно.

К таким запросам также очень удачно могут быть применены методы оптимизации или моделирования параллельных алгоритмов, основанные на матричной алгебре и теории графов.

А с помощью специальной программной надстройки возможно проведение анализа структуры и преобразование любого пользовательского сложного запроса, записанного на стандартном SQL.

Работа тестовых запросов была проверена с помощью утилиты MySQLSlap.