Математические структуры и моделирование
2011, вып. 24, с. 85–90

УДК 004.032.24:004.657

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ В ИСПОЛНИТЕЛЕ ЗАПРОСОВ СОВРЕМЕННЫХ СУБД НА ПРИМЕРЕ MYSQL

Д.А. Лыфарь

В статье исследованы алгоритмы обработки данных (выборки и соединения) в современных СУБД. Оценена теоретическая возможность исполнения их на графических процессорах

Введение

С целью портирования исполнителя (процессора) запросов СУБД MySQL на GPU был сделан обзор части его архитектуры и алгоритмов. Рассмотрена реализация современной СУБД MySQL (операции выборки, сортировки и соединения, а также индексирование). Предложены пути реализации этих операций на GPU.

1. Парсер

Парсер MySQL, как и многие другие, состоит из двух основных компонент: сканера и набора правил грамматики. Однако, в отличие от парсеров других СУБД [3], MySQL преобразовывает запрос не в байт-код, а в набор объектов классов C++ в памяти. Конечная цель парсера — это получение дерева разбора, которое затем будет преобразовано в логический план запроса. Дерево разбора в MySQL представлено объектом класса LEX. Среди прочих членов этого класса можно выделить: st_select_lex и sql_command. Первый отвечает за хранение различных параметров запроса, как-то: список таблиц, список полей, ссылки на объекты этого же типа, отвечающих за представление подзапросов и прочее. Второй определяет, какой тип запроса должен быть выполнен (выборка, вставка, удаление и др.).

Так как в нашей работе наибольший интерес представляет разбор и исполнение части WHERE оператора SELECT, рассмотрим, как MySQL реализует эту функциональность. Как и по многим другим частям разбираемого запроса, MySQL строит дерево разбора по WHERE, указатель на корень которого хранится как член уже упомянутого класса st_select_lex. Любое подобное дерево

ООО «Кристаллникс»
E-mail: dlyfar@gmail.com

является деревом разбора выражения, узлами и листьями которого являются наследники класса Item. Семейство наследников этого класса покрывает собой различные части синтаксиса SQL: арифметические операции, различные SQL функции, логические операторы, ссылки на столбцы таблицы и др.

Любой элемент дерева WHERE, GROUP BY, HAVING, ORDER BY представляет собой экземпляр класса наследника Item. Класс Item имеет набор виртуальных функций, начинающихся с val_. Нам интересна функция val_int (возвращающее целочисленное значение), так как именно её вызывает исполнитель запросов у корневого узла, чтобы определить, удовлетворяет ли запись заданному условию или нет. В случае если оптимизатор не смог переписать выражение более оптимально, то исполняется оригинальное выражение. В нашем случае это важно, так как оптимизатор может упростить лишние условия, операции и т.п., поэтому мы должны будем загрузить дерево выражений на GPU после фазы оптимизации.

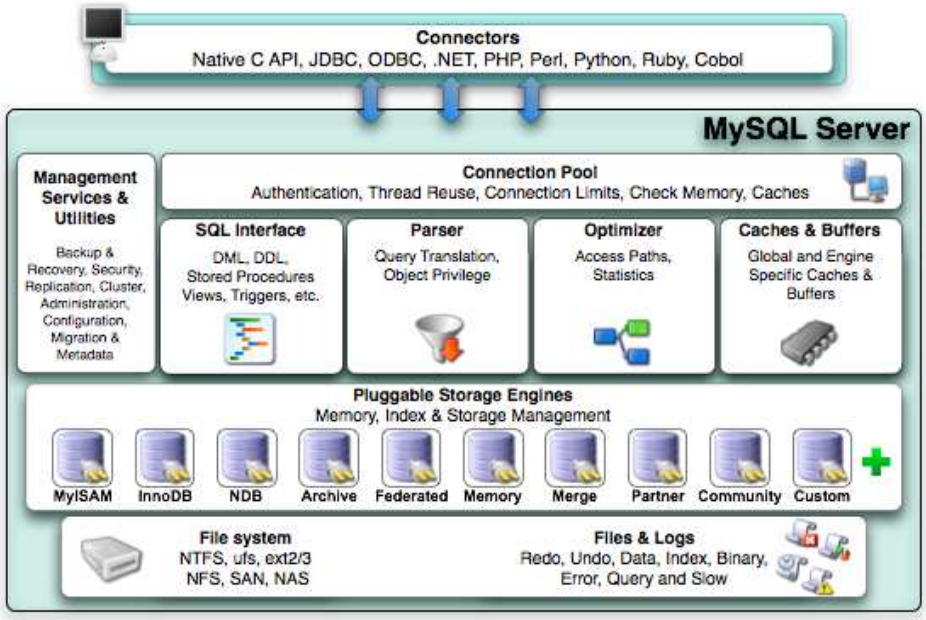

Рис. 1. Архитектура MySQL.

2. Хранилища данных

Для того чтобы понять, как MySQL исполняет запросы над данными, необходимо получить представление о способах доступа к этим данным и, в зависимости от выбранного способа для конкретного запроса, принимать решение о целесообразности исполнения запроса на графическом процессоре. СУБД MySQL имеет абстрактный унифицированный интерфейс, который позволяет разработать собственное хранилище данных. Из существующих и широко известных можно выделить MyISAM, InnoDB, XtraDB. Абстрактный интерфейс,

который реализует каждое из этих хранилищ, предоставляет оптимизатору и исполнителю запросов методы для простых операций: открытие/закрытие дескриптора таблицы, последовательное сканирование записей, поиск по ключу, добавление и обновление записей. Каждое из хранилищ обладает своими особенностями, в силу которых оптимизатор не всегда может подобрать унифицированный оптимальный план запроса. В качестве примера рассмотрим несколько различий между двумя хранилищами MyISAM и InnoDB.

2.1. Размещение данных в MyISAM

MyISAM сохраняет кортежи на диске в порядке их вставки, поскольку строки имеют фиксированный размер, MyISAM может найти любую из них путём смещения на требуемое число байт от начала таблицы. Что касается индексов, то в этом хранилище не делается структурных различий между первичным и остальными индексами. В MyISAM индексы представляют собой B-дерево, листьями которого являются пары ключ — номер строки в таблице. MyISAM не поддерживает кластерные индексы [1] и хранит каждый индекс отдельно от файла с данными, это говорит о том, что эффективнее всех с жёсткого диска будут считываться данные, сохранённые в порядке первичного индекса (чтение будет последовательно). Чтение же с помощью других индексов приводит к случайным дисковым запросам.

2.2. Размещение данных в InnoDB

InnoDB, в свою очередь, имеет кластерную организацию. Индекс хранится вместе с данными, отдельного хранилища для данных, как MyISAM, нет. Каждый листовой узел в кластерном индексе содержит значение первичного ключа, а также прочие столбцы, идентификатор транзакции, отката и другие служебные данные. Вторичные индексы в InnoDB имеют другую организацию. Здесь листьями вторичных индексов являются значения первичного ключа, а не номера строк, как в MyISAM. Это позволяет уменьшить количество работы при перемещении строки (не нужно обновлять номера строк в индексах, как в MyISAM).

Оптимизатор не всегда может использовать индексы (условия WHERE, где индексируемый столбец является частью более сложного условия), использовать покрывающие индексы (что приводит к нежелательному падению производительности на примере InnoDB). Если данные в таблице следуют не в порядке следования ключей, то это приведёт к тому, что данные считываются с жёсткого диска в произвольном порядке, а это значительно медленнее последовательного чтения.

2.3. Сортировка в MySQL

Сортировка в MySQL может быть с использованием индексов и файловая. Просмотр самого индекса происходит относительно быстро, однако здесь может

возникнуть упомянутая ранее проблема: данные следуют не в порядке первичного ключа (либо индекс не покрывает запрос и приходится считывать полную строку в случае с MyISAM из файла данных), что приводит к операциям чтения с произвольным доступом. В случае, если рабочая нагрузка характеризуется большим объёмом ввода/вывода, это будет значительно медленнее простого последовательного чтения данных.

Сортировка результатов по индексу работает только в тех случаях, когда порядок элементов в точности соответствует порядку, указанному в ORDER BY, а все столбцы отсортированы в одном направлении (по возрастанию или по убыванию). Если в запросе объединяется несколько таблиц (JOIN), то нужно, чтобы в ORDER BY упоминались столбцы только из первой таблицы. Ещё одно ограничение в ORDER BY заключается в том, что должен быть указан самый левый префикс ключа. В остальных случаях MySQL использует файловую сортировку.

Сортировка с использованием индекса работает очень быстро, т.к. данные считываются в уже отсортированном порядке. Что касается файловой сортировки, то это обычная быстрая сортировка над массивами данных, которые помещаются в оперативную память с последующим использованием сортировки слиянием. Объем оперативной памяти, доступный MySQL для файловой сортировки, определяется специальной переменной (sort_buffer_size). Если данные не помещаются полностью, то сортировка использует временный файл. Файловая сортировка может работать в двух режимах.

- Сортируемые элементы содержат все необходимые колонки исходной таблицы, результатом является отсортированный список кортежей, и нет необходимости обращаться к исходным данным после сортировки.

- Сортируются пары <ключ, row_id>, и затем доступ к данным осуществляется с помощью row_id, что приводит к чтению таблицы с произвольным доступом и сказывается на производительности.

Практически всегда оптимизатор старается использовать первый подход, исключение составляют столбцы типа BLOB или с переменной длиной. В силу указанных причин (не всегда возможно использовать индекс для сортировки) можно предположить, что целесообразно будет заменить обычную быструю сортировку на параллельную сортировку на GPU вместе с параллельной сортировкой на CPU одновременно. Таким образом будет сортироваться больший объем данных одновременно. Затем применить сортировку слиянием.

Операция соединения

В настоящее время можно выделить несколько подходов к реализации операции соединения в современных СУБД. К простейшему способу реализации относиться метод соединения посредством вложенных циклов (tuple-based nested loop join). Существует несколько способов оптимизировать реализацию:

разбиение данных на блоки для уменьшения количества операций чтения, использование индексов. Алгоритм читает кортежи из первой таблицы операции соединения (таблицы после разбора запроса не обязательно будут следовать начальному порядку из-за работы оптимизатора) и передаёт их во внутренний цикл, который обрабатывает следующую таблицу соединения и так далее до тех пор, пока все таблицы из запроса не будут обработаны. Псевдокод операции соединения приведён ниже:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

Очевидно, что таблицы из внутренних циклов приходится читать несколько раз. MySQL использует упомянутую выше буферную оптимизацию. Алгоритм использует буферы, чтобы уменьшит количество чтений кортежей из таблиц во внутренних циклах. Например, если десять кортежей считываются в буфер, и он передаётся в следующий внутренний цикл. Каждый кортеж таблицы из внутреннего цикла можно сравнить со всеми десятью из буфера. Работа алгоритма регулируется константой, задающей размер буфера, в котором хранятся только данные, которые имеют отношение к соединению, а не все данные. Буфер создаётся для каждой пары из соединения, т.е. каждое соединение может быть обработано с использованием нескольких буферов. Этот способ пока что единственный, используемый в MySQL в данный момент. Рассматривая соединение как операцию для возможного портирования на GPU, можно отметить лёгкость распараллеливания. Параллельным реализациям операций соединения посвящено несколько исследований. [2]. Предположим, что наш алгоритм обрабатывает соединение двух отношений: R — внешнее и S — внутреннее. В общем случае данные не помещаются полностью в оперативную память, поэтому алгоритм выполняется над порциями данных из отношений (обозначим их как R' и S'). Принимая во внимание ограничения архитектуры современных GPU, можно выделить несколько основных проблем в обработке соединений.

- Обращение к общей видеопамяти из блоков обладает высокой латентностью. Необходимо локализовать данные, по которым вычисляется соединение.

- Процесс записи результата. Ввиду отсутствия механизма, позволяющего эффективно выделить память на устройстве, результат соединения R' и S' должен сохранятся в два прохода: вычисление места, необходимого под результат, и запись данных. Тут возможно использование атомарных операций для единого глобального счётчика. Необходимо рассмотреть случай, когда результат соединения не помещается в доступную видеопамять. В этом случае придётся использовать несколько проходов.

Каждая группа потоков будет отвечать за соединение одного кортежа из R' с остальными из S'. Для того чтобы избежать частых обращений к глобальной видеопамяти, мы попытаемся загрузить S' полностью в локальную память, соответственно размер S' перед операцией соединения выбирается исходя из размеров доступной локальной (shared) памяти блока. В свою очередь число блоков будет равно числу кортежей из R'.

Заключение

Проведённое исследование показывает возможность использования графических процессоров для работы с базами данных. Мы рассмотрели только операции выборки, сортировки и соединения, что далеко от полного SQL. Следует иметь в виду ещё и многопользовательскую природу СУБД, работающих по архитектуре клиент-сервер. В таком окружении, возможно, необходим диспетчер задач, который будет распределять нагрузку между центральным процессором и графическим, а также использовать несколько устройств GPU. Основным препятствием для использования GPU в реальных проектах будет являться медленная шина между CPU и GPU, а также ограниченные возможности по менеджменту памяти непосредственно на GPU и её малый размер.

Литература

1. Шварц Б., Зайцев П., Ткаченко В. и др. MySQL. Оптимизация производительности, 2-е издание. М., СПб. : Символ-Плюс, 2010. 832 с.
2. He B., Yang K., Fang R., Lu M., Govindaraju M.K., Luo Q., and Sander P.V. Relational Joins on Graphics Processors. ACM SIGMOD 2008. URL: `http://www.cse.ust.hk/catalac/papers/gpujoin_sigmod08.pdf` (дата обращения: 1.09.2011)
3. The Architecture of SQLite. URL: `http://www.sqlite.org/arch.html` (дата обращения: 1.09.2011)