Модель затрат для оптимизации аналитических запросов в гетерогенных системах

П.А, Курапов, Д.В. Куликов и А.Ф. Мелик-Адамян

***Аннотация*—Одним из методов повышения производительности аналитических запросов является использование гетерогенных систем и эффективное распределение вычислений между устройствами. Оптимальность этого распределения, а следовательно, и итоговый прирост производительности зависят от способности оптимизатора запросов сравнивать эффективность планов исполнения. Одним из способов управления сложностью гетерогенного исполнения является разработка моделей затрат, которые бы поддерживали множество устройств. Поскольку прямая адаптация существующих моделей затруднительна, в этой статье мы предлагаем методологию оценки времени исполнения аналитического запроса в гетерогенной системе с помощью поиска вычислительных шаблонов с известными характеристиками производительности. Мы выделяем аналитические шаблоны и показываем, как план запроса представляется с помощью набора паттернов. Мы даем оценку эффективности модели, построенной на их переносимой реализации. Преимущество подхода заключается в простоте миграции на новые устройства и возможности автоматической калибровки при изменении параметров системы.**

***Ключевые слова*—гетерогенные вычисления, графический процессор, модель затрат, оптимизация аналитических запросов, обработка аналитических запросов, СУБД;**

I. Введение

Оценка вычислительной стоимости исполнения аналитического запроса играет важную роль в его оптимизации [1]. С переходом к использованию гетерогенных систем с целью повышения производительности СУБД [2] задача оценки затрат усложняется, а с ней и сама NP-трудная задача оптимизации запроса [3]. Классические модели, предсказывающие время исполнения запроса, основываются на составлении экспертной аналитической модели, которая учитывает особенности микроархитектуры центрального процессора. Поскольку появляющиеся ускорители отличаются от ЦПУ архитектурно и микроархитектурно [4], существующие модели затрат не могут быть легко адаптированы и переиспользованы [5]. Кроме того, с повышением многообразия исполнительных устройств [6] и с расширением иерархии системы памяти [7], сложность поддержки аналитических моделей растет экспоненциально. В этой статье мы предлагаем новый подход к оценке стоимости исполнения, который основан на выделении характеристических шаблонов аналитических запросов и составлении профиля производительности запроса на основе этих базовых блоков функциональности.

Статья организована следующим образом. Глава 2 дает описание классического подхода к модели затрат. В главе 3 мы выделяем основные вычислительные паттерны аналитических задач. Глава 4 разбирает пример того, как наши паттерны описывают реальный аналитический запрос, а также представляет общий алгоритм поиска паттернов в запросе. Общие результаты предсказательной силы модели приведены в главе 5.

II. Классический подход

Традиционно, поиск оптимального плана исполнения запроса строится на оценке его стоимости [8]. Стоимость может определяться по-разному, в зависимости от целей и требований к СУБД. Для простоты, без ограничения общности, мы будем рассматривать стоимость C как меру времени исполнения запроса $Texec$. На время исполнения запроса влияют следующие факторы:

- Размеры входных таблиц и типы используемых данных.
- Селективность предикатов — влияет на количество обрабатываемых записей.
- Реализаций реляционных операторов, составляющих запрос.
- Размещение данных в иерархии памяти.
- Устройства исполнения операторов.

Поскольку классические СУБД использовали гомогенные системы, последние два пункта не оказывали влияния на построение механизмов оценки стоимости. Именно это не позволяет переиспользовать существующий модели для оценки в гетерогенной системе.

Технически, вычисление стоимости запроса реализуется за счет двух компонент: оценки количества обрабатываемых записей (cardinality estimation) и

Статья получена 21 марта 2022.
Курапов П.А. аспирант МФТИ, 141701, Московская область, г. Долгопрудный, Институтский пер., 9. (email:kurapov@phystech.edu).
Куликов Д.В., студент СПбГУ, 199178, г. Санкт-Петербург, 14-ая линия Васильевского острова, 29. (email:st077260@student.spbu.ru).
Мелик-Адамян А.Ф., к. т. н., Intel (2200 Mission College Blvd. Santa Clara, CA 95054-1549 USA (+1) 408-765-8080), Остин, Техас (email:areg.melik-adamyan@intel.com).

применение аналитической модели (cost model), учитывающей особенности алгоритма и аппаратуры, на которой он исполняется [9]. Стоит, однако, отметить, что применение аналитической модели — не единственный способ управления процессом оптимизации. Возможно использование обучаемых целиком [10] или частично [11] моделей. Расширение пространства оптимизации является дополнительным препятствием для такого вида подходов, а управление оптимизацией на основе существующей модели и вовсе невозможно ввиду ее отсутствия.

При переходе к гетерогенным системам оценка количества обрабатываемых записей остается неизменной — она зависит только от статистических характеристик данных и вида запроса. Модель затрат же должна измениться.

III. Шаблоны аналитических вычислений

В основе нашего подхода лежит работа по выделению ключевых паттернов параллельных вычислений [12]. Мы расширяем эту классификацию на специфичные для аналитических запросов вычисления. Основой шаблона являются его характеристики с точки зрения уровня используемого параллелизма, взаимодействия между потоками и синхронизации и способа обращения в память. Мы выделяем следующие аналитические паттерны:

(1) Scan-filter (strided) — обращение к последовательной памяти столбца (столбцов) таблицы и вычисление предиката. Затраты на материализацию результата в общем случае ограничены созданием таблицы индексов записей, удовлетворяющим предикату.
(2) Hash build — вычисление хеша и построение хеш таблицы для набора записей. Значениями, обычно, выступают некоторого вида дескрипторы записей для экономии памяти. Требование к использованию дополнительной памяти существенно, поскольку в параллельных реализациях сохранение линейности представления данных (для эффективной передачи между устройствами и типами памяти).
(3) Hash probe — поиск в существующей хеш таблице по ключу, зачастую, без изменения состояния таблицы. Случайный характер доступа к памяти.
(4) Hash-join — объединение построения хеш-таблицы и поиска в ней.
(5) Group-by (hash partitioning) — разделение данных на группы по предикату. Размещение групп в памяти нетривиально: для устройств без динамической памяти или поддержки единого линейного буфера на все группы необходимо оценить и заранее выделить достаточно слотов. В частном случае разделение данных по хешу.
(6) Group-by-aggregate — разделение данных на группы с последующим применением агрегации к каждой. Помимо подсчета агрегации, обычно, присутствует дополнительная логика, например детектирование переполнения.
(7) Radix sort — перестановка записей в таблице согласно произвольному компаратору. Для ограниченных типов типично использование поразрядной сортировки в силу ее хорошей асимптотической сложности [13].
(8) Reduction — применение агрегационной функции к буферу. Уровень параллелизма зависит от размера буфера.

A. *Реализация основных паттернов в СУБД*

В реальной СУБД, использующей графический и центральный процессоры для исполнения запросов, генерация кода и способ исполнения различаются по причинам производительности [14]. В их основе лежит несколько фундаментальных различий в аппаратуре:

(1) Исполнение кода запроса на графическом процессоре подразумевает перемещение памяти таблиц в доступное адресное пространство. На практике, обычно, это означает использование относительно медленной шины PCIe. Поэтому, для эффективного исполнения предпочтительно совместное исполнение операторов без промежуточной материализации и обработка данных больших размеров [15]. Важно, что размеры данных не могут превышать размер графической памяти по той же причине.
(2) Модели программирования, такие как CUDA®/ OpenCL™ [16] требуют от разработчика заботиться об эффективном использовании параллелизма внутри одной единицы исполнения («ядра»). При использовании графического ускорителя такой подход оправдан, поскольку аппаратура построена на принципе выполнение одинакового кода и оптимизирована для этого случая. Для центрального процессора возможно использование более гибкого механизма параллельных вычислений, на уровне потоков. Планирование единичных потоков значительно проще в реализации как с точки зрения генерируемого (однопоточного, но, возможно, векторного) кода [4], так и с точки зрения политик использования ядер процессора.
(3) Множество различий в производительности элементарных операций, таких как примитивы синхронизации, сложные арифметические операции вроде деления, инструкции ветвления, обращения в памяти и другие [17].
(4) Различия в предоставляемой функциональности. Пример: динамическое выделение памяти — для графического процессора, обычно, требуется заранее выделить необходимые буферы в памяти до фактического исполнения кода. Это влияет на организацию классических структур данных, таких как хеш-таблица [18].

Рассмотрим подходы к реализации основных операций аналитических запросов в СУБД OmniSciDB [19] и отличия в генерируем коде для ЦПУ и графического

процессора.

В основе эффективной СУБД с компиляцией запросов в машинный код лежит важная оптимизация объединения реляционных операторов в одно «ядро» исполнения, обрабатываемое отдельным логическим потоком, — код нескольких операторов выполняется внутри одного цикла, оставляя значения записей в регистрах и, таким образом, экономя расходы на промежуточную материализацию [20]. Модель параллелизма в OmniSciDB опирается на разделение данных таблиц на «фрагменты» (аналогично «morsel» [21]). Потоки могут обрабатывать данные для одного или более фрагмента за раз. Множественная обработка фрагментов позитивно влияет на исполнение на графическом процессоре, поскольку снижает количество обращений к драйверу. Поскольку исполнение на графическом процессоре не независимо, для каждого выделяется поток на ЦПУ, который ожидает результата. Результаты всегда агрегируются на центральном процессоре, даже если часть запроса была выполнена на графическом процессоре. Однако, в случае последовательного исполнения ядер на графическом процессоре, промежуточные результаты не перемещаются на ЦПУ между их вызовами. Так достигается минимизация использования относительно медленной передачи данных по PCIe шине.

1) Group-by.

OmniSciDB покрывает широкий класс запросов с помощью шаблона группировки (group-by). Для вычислений необходимо сформировать буферы для групп. В случае графического процессора размеры этих буферов следует оценить или вычислить заранее, чтобы обеспечить достаточное количество «слотов». Исполняющее ядро строит хеш-таблицу с буфером для каждой группы или значения агрегации. Значения в слотах занимают ключи и идентификаторы записей в таблице. Поскольку на ЦПУ, в отличие от графического процессора, отсутствует многопоточность внутри ядра (intra-query parallelism), нет необходимости в синхронизации доступа к хеш-таблице [22]. Принципиально, однако, на алгоритм вычисления паттерна это не влияет.

2)Scan

Выборка реализуется за счет стратегии параллелизма и состоит из двух операторов группировки. Первый оценивает количество необходимой памяти (count(*)) для локального буфера, второй выполнит необходимые вычисления. Для получения итогового результата строится дерево агрегаций результатов всех ядер.

3)Join

В OmniSciDB поддержаны два алгоритма соединений: хешированием и с помощью вложенных циклов. Основным методом, в силу его высокой производительности, является соединение хешированием, поэтому его мы и рассмотрим подробнее. СУБД использует два подхода: классический механизм линейного разрешения коллизий и его оптимизированный вариант с использованием информации о данных и построении «идеальной» хеш-функции. Для повышения уровня доступного параллелизма и, как следствия, эффективности исполнения, алгоритм соединения разделен на три стадии:

- Выделение памяти и инициализация хеш-таблицы.
- Подсчет количества коллизий и вычисление смещений позиций значений.
- Фактическое заполнение значений в хеш-таблице.

Реализация этого подхода опирается на специальную структуру памяти, которая одинакова для разных устройств. Хеш-таблица состоит из трех последовательно-расположенных в памяти массивов. Первый хранит ключи. Второй — позиции списков элементов в третьем массиве, соответствующих ключам из первого. Таким образом, для поиска элемента в таблице, как обычно, вычисляется хеш и по нему, по схеме линейного разрешения коллизий находится ключ. Если ключ присутствует, то обращение к искомой записи происходит следующим образом. Индекс найденного ключа в первом массиве используется для поиска смещения. Искомая запись находится в третьем массиве по найденному смещению. Сам механизм соединения не претерпевает изменений.

4)Sort

Сортировка опирается на существующие решения и в значительной степени зависит от устройства исполнения. Так, ЦПУ использует параллельную сортировку из TBB ([23]). Отличительной особенностью алгоритма является то, что получаемый результат перестановки не обязательно необходим пользователю мгновенно. Конвейер операторов может потреблять результат сортировки. По этой причине, в целях экономии расходов памяти на промежуточную материализацию, результатом сортировки становится сама перестановка.

IV. Применение паттернов для оценки стоимости исполнения

Выделенные паттерны составляют набор базовых блоков построения аналитических запросов. В гетерогенной системе реализации операторов (а точнее, конвейеров операторов, разделенных точками обязательной материализации промежуточных результатов) могут варьироваться для обеспечения оптимальности использования возможностей аппаратуры. Для СУБД, использующих компиляцию, однако, предпочтительнее схема с генерацией единого пути для кода запроса. Поэтому, мы предпринимаем попытку унификации базовых блоков для различных устройств. Другими словами, реализация каждого шаблона будет выражаться одним универсальным алгоритмом, выраженным средствами

кроссплатформенного языка. В работе в качестве такой платформы мы выбрали язык SYCL [24]. Реализации всех приведенных в статье шаблонов находятся в открытом доступе[1].

Использование существующих примитивов для аппроксимации аналитических задач в большинстве случаев оказывается нетривиальной задачей, даже когда задача опирается на классические алгоритмы. В качестве примера, рассмотрим сортировку. Существующая имплементация параллельной сортировки на ЦПУ в стандартной библиотеке SYCL значительно проигрывает по производительности оптимизированной реализации TBB (рис. 1).

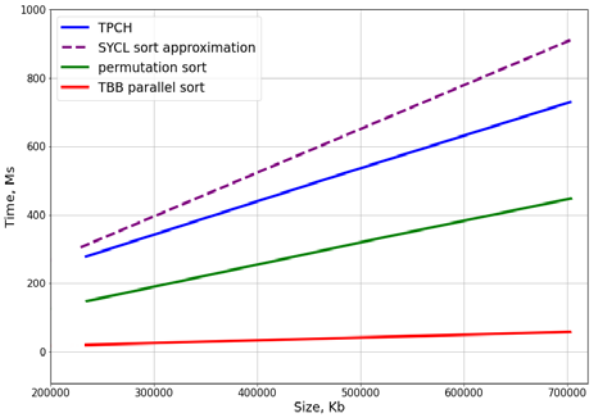

Рисунок 1. Сравнение производительности запроса, состоящего из сортировки (TPCH), и паттернов в различной реализации

A. Пример

```
select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as
sum_base_price,
  sum(l_extendedprice * (1 -
l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount)
* (1 + l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from
  lineitem
where
  l_shipdate <= date '1998-12-01' -
interval '90' day (3)
group by
  l_returnflag,
  l_linestatus
order by
  l_returnflag,
  l_linestatus;
```

Листинг 1. TPC-H Q1.

[1] https://github.com/kurapov-peter/dwarf_bench

Логический план запроса (рис. 2) стандартной системы бенчмаркинга СУБД TPC-H [25] (листинг 2) порождает единое ядро исполнения, состоящее из двух вложенных операторов — физический план (рис. 3). Информация о вычисляемых агрегациях содержится внутри оператора RelCompound.

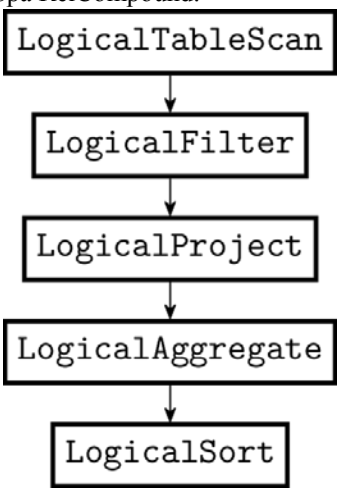

Рисунок 2. Логический план запроса TPC-H Q1.

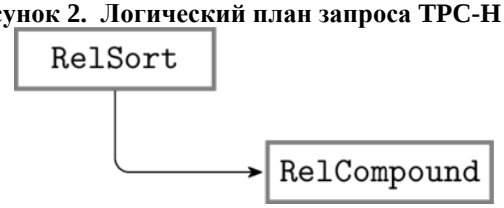

Рисунок 3. Физический план запроса TPC-H Q1.

Таблица 1. Зависимость времени исполнения от вида агрегации для TPCH-Q1, SF=10. Количество выражений агрегации означает последовательное добавления их в запрос в порядке сверху-вниз. Последнее измерение соответствует полному запросу.

Кол-во выражений агрегации	Среднее время исполнения, мс	Стандартное отклонение, мс
1	718	8
2	1044	7
3	1361	9
4	1682	13
5	1684	12
6	1679	13
7	1682	9
8	1674	19

Таблица 2. Зависимость времени исполнения от вида агрегации для TPCH-Q1, SF=20. Количество выражений агрегации означает последовательное добавления их в запрос в порядке сверху-вниз. Последнее измерение соответствует полному запросу.

Кол-во выражений агрегации	Среднее время исполнения, мс	Стандартное отклонение, мс
1	1366	23
2	1983	24
3	2622	14
4	3227	25
5	3211	30
6	3219	20
7	3235	26
8	3227	23

План запроса представляет собой набор примитивов (рис. 3). В таком виде запрос может быть составлен из паттернов group-by-agg и сортировки. Поскольку вычисление агрегации нетривиально, мы сперва делаем эмпирическую оценку накладных расходов, которые вносятся за счет вычисления нескольких агрегаций (таблица 1). Данные наглядно демонстрируют, что добавление функции агрегации, использующей уже задействованный столбец в запросе, не вносит ощутимого вклада во время исполнения (начиная с 4-х выражений, используются все те же столбцы l_extendedprice, l_quantity, l_discount, l_tax, см. листинг 1). Причиной такой зависимости является то, что генерация кода в СУБД строит цикл обхода нужных столбцов и обновляет значение аккумулятора агрегации для каждой записи, удовлетворяющей предикату фильтра. Добавление еще одного аккумулятора в данном случае подразумевает несколько дополнительных арифметических инструкций в цикле, что не является существенным, по сравнению с затратами на обращение в память.

Кроме того, можно заметить, что добавление обращения к столбцам одинакового типа вносит равные затраты на обработку за счет одинаковых размеров данных (табл. 3) — примерно 300мс на столбец. Таким образом, моделирование запроса должно состоять из group-by-agg, обращающегося к четырем столбцам, и сортировки результата. Закономерность сохраняется при изменении размеров таблиц (рис. 4). Для краткости, здесь мы приводим еще только одно измерение (таблица 2). Вдвое больший размер таблиц (SF=20) дает вдвое больший вклад во время исполнения при добавлении столбца.

Таблица 3. Размеры интересующих нас полей в таблице lineitem.

Поле таблицы	Размер в байтах
l_partkey, l_quantity, l_extendedprice, l_discount, l_tax	8
l_shipdate	2
l_returnflag, l_linestatus	1

Полученная таким образом базовая модель хорошо аппроксимирует и запросы вида TPC-H Q6 (рис. 5), без эксплицитного выражения группировки, в силу устройства генератора кода.

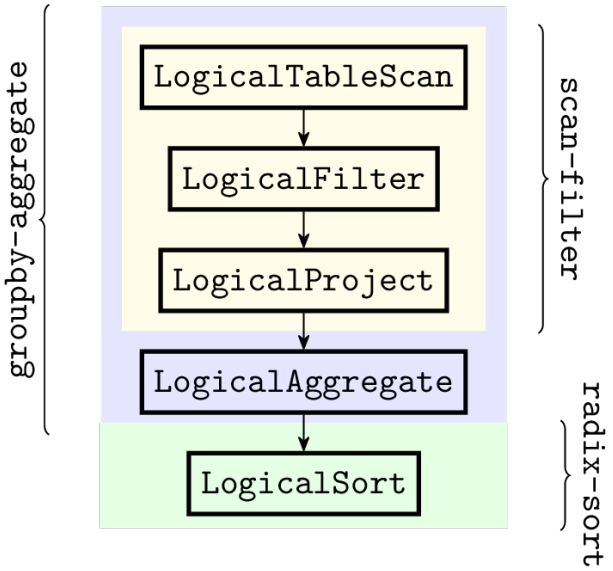

Рисунок 3. Разбиение логического плана TPC-H Q1 на аналитические паттерны

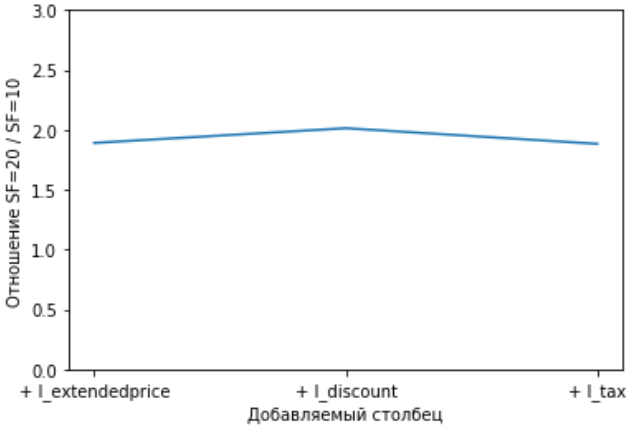

Рисунок 4. Отношение изменения времени исполнения запроса TPC-H Q1 при добавлении агрегаций по столбцам между SF=20 и SF=10.

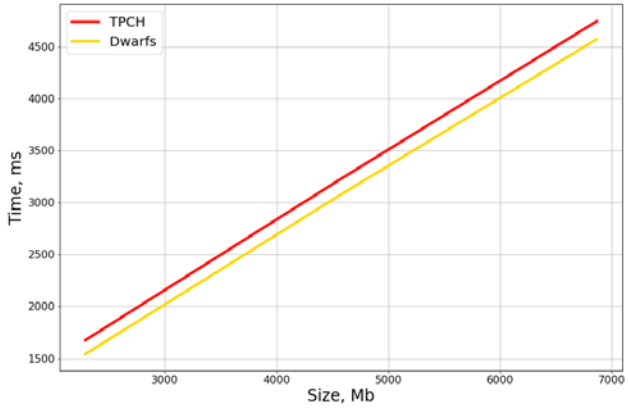

Рисунок 5. Аппроксимация запроса TPC-H Q6 с помощью шаблона group-by.

```
select
  sum(l_extendedprice * l_discount) as
revenue
from
  lineitem
where
  l_shipdate >= date '1994-01-01'
```

```
  and l_shipdate < date '1994-01-01' +
interval '1' year
  and l_discount between.06 - 0.01
  and.06 + 0.01
  and l_quantity < 24;
  l_returnflag,
  l_linestatus
order by
  l_returnflag,
  l_linestatus;
```

Листинг 2. TPC-H Q6.

Стоит отметить, что наивная реализация паттерна в виде вызова функции стандартной библиотеке std::copy_if на языке SYCL дает значительно расходящиеся результаты. Это особенно выражено для центрального процессора. Поэтому, в секции результатов мы приводим сравнение для ЦПУ.

B. Алгоритм оценки стоимости запроса

В основе алгоритма вычисления стоимости запроса лежит поиск аналитических паттернов в графе запроса. Задача поиска аналитических паттернов значительно проще, чем обобщенная задача поиска паттернов в графе [26], поскольку «шаги» исполнения описываются независимо и большинство покрывается линейной последовательностью паттернов. Кроме того, сами паттерны являются вложенными, что дает возможность использовать жадную стратегию для поиска подходящего. Для этого мы вводим меру покрытия для шаблона (ordinal) и осуществляем поиск среди шаблонов с покрытием от наибольшего к наименьшему.

Стоимость операторов, покрытых шаблоном, определяется с помощью экстраполяции заранее полученных моделью данных. Удобство подхода заключается в переносимости самих шаблонов. Это позволяет автоматически формировать модель затрат на этапе калибровки для новой системы. Поскольку шаблоны являются реальным исполняемым кодом, модель учитывает микроархитектурные эффекты, а добавление поддержки нового устройства в СУБД требует лишь поддержки инструментов разработки (OpenCL/SYCL платформы).

Algorithm 1 Алгоритм вычисления стоимости запроса

Input: G, inputs, estimations ▷ Граф запроса и метаданные
Input: $(P, ordinal)$ ▷ Набор шаблонов P и соответствующая мера покрытия
Output: $cost$

```
cost ← 0
Ordinal_max ← max(ordinals)
for step ∈ topological_sort(G) do ▷
    op ← root
    Add op to queue Q
    while Q! = Ø do
        while ordinal(op) < Ordinal_max do
            Add all children of op to Q
            op append Q.pop()
        end while
        ord ← Ordinal_max
        for pattern ∈ P[ord] do
            if pattern matches op then
                cost ← cost + cost(pattern)
                Remove all nodes corresponding to ord and break
            end if
            ord ← ord − 1
        end for
    end while
end for
```

V. Результаты

Поскольку в OmniSciDB исторически основным устройством исполнения был графический процессор, генерируемый код оптимизировался под его особенности. Соответственно, использование параллельных шаблонов вычислений хуже приближает данные производительности, получаемые на ЦПУ. Поэтому здесь мы приводим отклонения модели от производительности различных классов запросов из набора TPC-H (размер данных варьировался стандартным образом — scale factor от 1 до 30) полученные при исполнении на центральном процессоре. Измерения проводились на 2-х-сокетном Intel(R) Xeon(R) Platinum 8280L ЦПУ с тактовой частотой 2.70GHz. Средняя ошибка модели на превышает 25% в худшем случае. Данные по средним отклонениям модели от реального времени исполнения представлены на рис. 6.

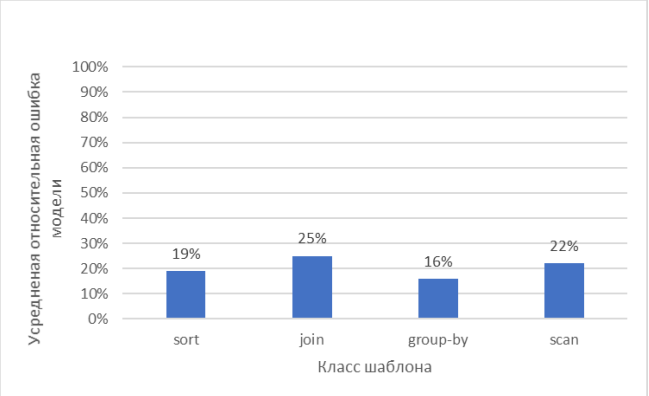

Рисунок 6. Средняя относительная ошибка модели затрат для различных классов запросов.

VI. Заключение

В статье представлены основные принципы построения модели затрат для оптимизации аналитических запросов в гетерогенных системах. Дана оценка ошибки запросов на различных классах шаблонов. Полученные результаты свидетельствуют о том, что подход является возможной альтернативой аналитическим моделям и имеет ряд преимуществ перед ними. В частности, использование набора шаблонов в качестве внешней библиотеки позволяет оптимизаторам запросов напрямую оценивать свойства системы. При переходе на новые устройства с поддержкой SYCL трудозатраты переиспользования подхода минимизируются. Изменение конфигурации системы так же решается автоматически, дополнительной калибровкой. К недостаткам подхода следует отнести требование к унифицированности алгоритмов описания шаблонов. Это ограничение ставит производительность в зависимость от возможностей компилятора и снижает предсказательную силу получаемых моделей.

В дальнейшем, модель требует расширения покрытия функциональности СУБД по типам операций и типам данных, а также частичную специализацию для плохо воспроизводимых особенностей производительности системы.

Библиография

[1] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9, 3 (Nov. 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[2] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. 2018. Vbench: Benchmarking Video Transcoding in the Cloud. Association for Computing Machinery, New York, NY, USA, 797–809. https://doi.org/10.1145/3173162.3173207

[3] Chihping Wang and Ming-Syan Chen. 1996. On the complexity of distributed query optimization. IEEE Transactions on Knowledge and Data Engineering8, 4(1996), 650–662. https://doi.org/10.1109/69.536256

[4] Timo Kersten, Viktor Leis, Alfons Kemper, Thomas Neumann, Andrew Pavlo, and Peter Boncz. 2018. Everything You Always Wanted to Know about Compiled and Vectorized Queries but Were Afraid to Ask. Proc. VLDB Endow. 11, 13 (Sept. 2018), 2209–2222. https://doi.org/10.14778/3275366.3284966

[5] Sebastian Breß, Max Heimel, Norbert Siegmund, Ladjel Bellatreche, and Gunter Saake. 2014. GPU-Accelerated Database Systems: Survey and Open Challenges. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–35. https://link.springer.com/chapter/10.1007/978-3-662-45761-0_1

[6] Periklis Chrysogelos, Panagiotis Sioulas, and A. Ailamaki. 2019. Hardware-conscious Query Processing in GPU-accelerated Analytical Engines. In CIDR. http://cidrdb.org/cidr2019/papers/p127-chrysogelos-cidr19.pdf

[7] Сергей Кузнецов. 2020. В ожидании нативных архитектур СУБД на основе энергонезависимой основной памяти. Труды Института системного программирования РАН 32, 1 (2020). https://ispranproceedings.elpub.ru/jour/article/view/12717

[8] Yannis E. Ioannidis. 1996. Query Optimization. ACM Comput. Surv. 28, 1 (March 1996), 121–123. https://doi.org/10.1145/234313.234367

[9] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. Database System Concepts, Seventh Edition. McGraw-Hill Book Company. https://www.db-book.com/db7/index.html

[10] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. Proc. VLDB Endow. 12, 11 (July 2019), 1705–1718. https://dl.acm.org/doi/10.14778/3342263.3342644

[11] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In Proceedings of the 2021 International Conference on Management of Data (Virtual Event, China) (SIGMOD/PODS '21). Association for Computing Machinery, New York, NY, USA, 1275–1288. https://dl.acm.org/doi/10.1145/3448016.3452838

[12] Krste Asanovi´c, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. 2006. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[13] Duane G. Merrill and Andrew S. Grimshaw. 2010. Revisiting Sorting for GPGPU Stream Architectures. In Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (Vienna, Austria) (PACT'10). Association for Computing Machinery, New York, NY, USA, 545–546. https://doi.org/10.1145/1854273.1854344

[14] Jeronimo Castrillon, Matthias Lieber, Sascha Kl ¨uppelholz, Marcus V ¨olp, Nils Asmussen, Uwe Aßmann, Franz Baader, Christel Baier, Gerhard Fettweis, Jochen Fr ¨ohlich, Andr´es Goens, Sebastian Haas, Dirk Habich, Hermann H ¨artig, Mattis Hasler, Immo Huismann, Tomas Karnagel, Sven Karol, Akash Kumar, Wolfgang Lehner, Linda Leuschner, Siqi Ling, Steffen M ¨arcker, Christian Menard, Johannes Mey, Wolfgang Nagel, Benedikt N ¨othen, Rafael Pe ˜naloza, Michael Raitza, J ¨org Stiller, Annett Ungeth ¨um, Axel Voigt, and Sascha Wunderlich. 2018. A Hardware/Software Stack for Heterogeneous Systems. IEEE Transactions on Multi-Scale Computing Systems 4, 3 (2018), 243–259. https://doi.org/10.1109/TMSCS.2017.2771750

[15] Peter Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-pipelining query execution. In In CIDR. http://cidrdb.org/cidr2005/papers/P19.pdf

[16] John E. Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. Computing in Science Engineering 12, 3 (2010), 66–73. https://doi.org/10.1109/MCSE.2010.69

[17] Emily Furst, Mark Oskin, and Bill Howe. 2017. Profiling a GPU Database Implementation: A Holistic View of GPU Resource Utilization on TPC-H Queries. In Proceedings of the 13th International Workshop on Data Management on New Hardware (Chicago, Illinois) (DAMON '17). Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3076113.3076119

[18] Daniil Kulikov, Daria Nikolskaia, and Petr Kurapov. 2021. Efficient Hardware-Agnostic DBMS Operator Implementation Using SYCL. In 2021 International Conference Engineering and Telecommunication (En T). 1–5. https://doi.org/10.1109/EnT50460.2021.9681747

[19] Yansong Zhang, Yu Zhang, Jiaheng Lu, Shan Wang, Zhuan Liu, and Ruichen Han. 2020. One size does not fit all: accelerating OLAP workloads with GPUs. Distributed and Parallel Databases 38 (12 2020). https://doi.org/10.1007/s10619-020-07304-z

[20] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. Proc. VLDB Endow. 4, 9 (June 2011), 539–550. https://doi.org/10.14778/2002938.2002940

[21] Viktor Leis, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2014. Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (Snowbird, Utah, USA) (SIGMOD '14). Association for Computing Machinery, New York, NY, USA, 743–754. https://doi.org/10.1145/2588555.2610507

[22] Henning Funke, Sebastian Breß, Stefan Noll, Volker Markl, and Jens Teubner. 2018. Pipelined Query Processing in Coprocessor Environments. In Proceedings of the 2018 International Conference on Management of Data (Houston, TX, USA) (SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 1603–1618. https://doi.org/10.1145/3183713.3183734

[23] Michael Voss, Rafael Asenjo, and James Reinders. 2019. Pro TBB: C++ Parallel Programming with Threading Building Blocks (1st ed.). Apress, USA. https://dl.acm.org/doi/book/10.5555/3364289

[24] Ben Ashbaugh, Alexey Bader, James Brodman, Jeff Hammond, Michael Kinsner, John Pennycook, Roland Schulz, and Jason Sewall. 2020. Data Parallel C++: Enhancing SYCL Through Extensions for

Productivity and Performance. In Proceedings of the International Workshop on OpenCL (Munich, Germany) (IWOCL'20). Association for Computing Machinery, New York, NY, USA, Article 7, 2 pages. https://doi.org/10.1145/3388333.3388653

[25] http://www.tpc.org/tpch/

[26] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. 2010. Graph Pattern Matching: From Intractable to Polynomial Time. Proc. VLDB Endow. 3, 1–2 (sep 2010), 264–275. https://doi.org/10.14778/1920841.1920878

A cost model for analytical query optimization

Petr Kurapov, Daniil Kulikov, Areg Melik-Adamyan

***Abstract*— Analytical query performance improvement can be achieved via efficient work distribution among devices of a heterogeneous system. The resulting performance gain highly depends on the ability of an optimizer to compare execution plans. One way to manage the complexity of a heterogeneous system is to develop cost models to that support multiple devices. Reusing existing CPU models is complicated if not impossible. This paper introduces a methodology for analytical query execution time estimation in a heterogeneous system by matching its plan to a set of computational patterns with known performance characteristics. We identify key and most common patterns and show how a query plan maps to them. We provide a general algorithm for cost calculation and evaluate model effectiveness by building a library of portable implementations and comparing their performance to a real in-memory DBMS.**

Keywords: DBMS; GPU; analytical query optimization; OLAP; analytical query processing; cost model; in-memory databases; heterogeneous query processing;

REFERENCES

[1] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9, 3 (Nov. 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[2] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. 2018. Vbench: Benchmarking Video Transcoding in the Cloud. Association for Computing Machinery, New York, NY, USA, 797–809. https://doi.org/10.1145/3173162.3173207

[3] Chihping Wang and Ming-Syan Chen. 1996. On the complexity of distributed query optimization. IEEE Transactions on Knowledge and Data Engineering8, 4(1996), 650–662. https://doi.org/10.1109/69.536256

[4] Timo Kersten, Viktor Leis, Alfons Kemper, Thomas Neumann, Andrew Pavlo, and Peter Boncz. 2018. Everything You Always Wanted to Know about Compiled and Vectorized Queries but Were Afraid to Ask. Proc. VLDB Endow. 11, 13 (Sept. 2018), 2209–2222. https://doi.org/10.14778/3275366.3284966

[5] Sebastian Breß, Max Heimel, Norbert Siegmund, Ladjel Bellatreche, and Gunter Saake. 2014. GPU-Accelerated Database Systems: Survey and Open Challenges. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–35. https://link.springer.com/chapter/10.1007/978-3-662-45761-0_1

[6] Periklis Chrysogelos, Panagiotis Sioulas, and A. Ailamaki. 2019. Hardware-conscious Query Processing in GPU-accelerated Analytical Engines. In CIDR. http://cidrdb.org/cidr2019/papers/p127-chrysogelos-cidr19.pdf

[7] Kuznetsov S.D. In anticipation of native DBMS architectures based on non-volatile main memory. Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2020;32(1):153-180. (In Russ.) https://doi.org/10.15514/ISPRAS-2020-32(1)-9

[8] Yannis E. Ioannidis. 1996. Query Optimization. ACM Comput. Surv. 28, 1 (March 1996), 121–123. https://doi.org/10.1145/234313.234367

[9] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. Database System Concepts, Seventh Edition. McGraw-Hill Book Company. https://www.db-book.com/db7/index.html

[10] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. Proc. VLDB Endow. 12, 11 (July 2019), 1705–1718. https://dl.acm.org/doi/10.14778/3342263.3342644

[11] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In Proceedings of the 2021 International Conference on Management of Data (Virtual Event, China) (SIGMOD/PODS '21). Association for Computing Machinery, New York, NY, USA, 1275–1288. https://dl.acm.org/doi/10.1145/3448016.3452838

[12] Krste Asanovi´c, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. 2006. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[13] Duane G. Merrill and Andrew S. Grimshaw. 2010. Revisiting Sorting for GPGPU Stream Architectures. In Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (Vienna, Austria) (PACT'10). Association for Computing Machinery, New York, NY, USA, 545–546. https://doi.org/10.1145/1854273.1854344

[14] Jeronimo Castrillon, Matthias Lieber, Sascha Kl ¨uppelholz, Marcus V¨olp, Nils Asmussen, Uwe Aßmann, Franz Baader, Christel Baier, Gerhard Fettweis, Jochen Fr¨ohlich, Andr´es Goens, Sebastian Haas, Dirk Habich, Hermann H¨artig, Mattis Hasler, Immo Huismann, Tomas Karnagel, Sven Karol, Akash Kumar, Wolfgang Lehner, Linda Leuschner, Siqi Ling, Steffen M¨arcker, Christian Menard, Johannes Mey, Wolfgang Nagel, Benedikt N¨othen, Rafael Pe˜naloza, Michael Raitza, J¨org Stiller, Annett Ungeth ¨um, Axel Voigt, and Sascha Wunderlich. 2018. A Hardware/Software Stack for Heterogeneous Systems. IEEE Transactions on Multi-Scale Computing Systems 4, 3 (2018), 243–259. https://doi.org/10.1109/TMSCS.2017.2771750

[15] Peter Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-pipelining query execution. In In CIDR. http://cidrdb.org/cidr2005/papers/P19.pdf

[16] John E. Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. Computing in Science Engineering 12, 3 (2010), 66–73. https://doi.org/10.1109/MCSE.2010.69

[17] Emily Furst, Mark Oskin, and Bill Howe. 2017. Profiling a GPU Database Implementation: A Holistic View of GPU Resource Utilization on TPC-H Queries. In Proceedings of the 13th International Workshop on Data Management on New Hardware (Chicago, Illinois) (DAMON '17). Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3076113.3076119

[18] Daniil Kulikov, Daria Nikolskaia, and Petr Kurapov. 2021. Efficient Hardware-Agnostic DBMS Operator Implementation Using SYCL. In 2021 International Conference Engineering and Telecommunication (En T). 1–5. https://doi.org/10.1109/EnT50460.2021.9681747

[19] Yansong Zhang, Yu Zhang, Jiaheng Lu, Shan Wang, Zhuan Liu, and Ruichen Han. 2020. One size does not fit all: accelerating OLAP workloads with GPUs. Distributed and Parallel Databases 38 (12 2020). https://doi.org/10.1007/s10619-020-07304-z

[20] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. Proc. VLDB Endow. 4, 9 (June 2011), 539–550. https://doi.org/10.14778/2002938.2002940

[21] Viktor Leis, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2014. Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (Snowbird, Utah, USA) (SIGMOD '14). Association for Computing Machinery, New York, NY, USA, 743–754. https://doi.org/10.1145/2588555.2610507

[22] Henning Funke, Sebastian Breß, Stefan Noll, Volker Markl, and Jens Teubner. 2018. Pipelined Query Processing in Coprocessor Environments. In Proceedings of the 2018 International Conference on Management of Data (Houston, TX, USA) (SIGMOD '18).

Association for Computing Machinery, New York, NY, USA, 1603–1618. https://doi.org/10.1145/3183713.3183734

[23] Michael Voss, Rafael Asenjo, and James Reinders. 2019. Pro TBB: C++ Parallel Programming with Threading Building Blocks (1st ed.). Apress, USA. https://dl.acm.org/doi/book/10.5555/3364289

[24] Ben Ashbaugh, Alexey Bader, James Brodman, Jeff Hammond, Michael Kinsner, John Pennycook, Roland Schulz, and Jason Sewall. 2020. Data Parallel C++: Enhancing SYCL Through Extensions for Productivity and Performance. In Proceedings of the International Workshop on OpenCL (Munich, Germany) (IWOCL'20). Association for Computing Machinery, New York, NY, USA, Article 7, 2 pages. https://doi.org/10.1145/3388333.3388653

[25] TPCH http://www.tpc.org/tpch/

[26] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. 2010. Graph Pattern Matching: From Intractable to Polynomial Time. Proc. VLDB Endow. 3, 1–2 (sep 2010), 264–275. https://doi.org/10.14778/1920841.1920878