Федеральное государственное автономное образовательное

учреждение высшего образования

«Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Основная образовательная программа

Прикладная математика и информатика

ГРУППОВАЯ КУРСОВАЯ РАБОТА

(программный проект)

на тему

Оптимизация сортировки в ClickHouse

Выполнили студенты:

Гумеров Арслан Рустемович, группа 176, 3 курс

Кидрачев Альберт Тимурович, группа 176, 3 курс

Морозов Василий Игоревич, группа 176, 3 курс

Научный руководитель:

Руководитель группы разработки ClickHouse, Миловидов Алексей Николаевич

Москва

2020

Оглавление

1. Аннотация

1.1 Аннотация

Сортировка элементов - одна из наиболее часто использующихся функций в современных базах данных. Её производительность очень важна и сильно влияет на скорость работы базы в целом. Целью данной работы являлось изучение методов оптимизации алгоритмов сортировки и их реализация с целью увеличения производительности ClickHouse, системы управления базами данных разработанной компанией Яндекс.

1.2 Abstract

Sorting is one of the most commonly used functions in modern databases. Its performance is very important and greatly affects the speed of the base as a whole. The aim of this work was to study methods for optimizing sorting algorithms and their implementation in order to increase the performance of ClickHouse, a database management system developed by Yandex.

1.3 Ключевые слова

ClickHouse, sort, performance, C++, СУБД

1.4 Словарь терминов

БД - База данных

СУБД - Система управления базами данных

ClickHouse - СУБД разработанная Яндексом

Компаратор - функция, которая сравнивает элементы

SQL - язык программирования для управления данными в базе данных

Бенчмарк - программа для сравнения производительности

2. Введение

ClickHouse - это колоночная система управления базами данных (СУБД), которая распространяется под лицензией Apache 2.0 и является Open-Source продуктом компании Яндекс. Изначально она разрабатывалась для системы веб-аналитики Яндекс.Метрика, которой была необходима быстрая обработка огромного объема информации. Так как основной ее задачей ставилось быстродействие выполнения аналитических запросов в реальном времени, то решено было выбрать колоночную архитектуру, так как часто подобные запросы содержат в себе агрегатные функции по столбцам. Подобный подход позволяет сильно сэкономить на обращениях к последовательным кускам памяти.

Хотя код СУБД довольно хорошо оптимизирован, что позволяет ему показывать высокую производительность в сравнении с аналогами, с ростом интереса пользователей к данной программе повышаются и требования к скорости обработки запросов. Довольно яркой точкой роста является оптимизация запросов с модификатором ORDER BY, который подразумевает вывод результата команды SELECT в отсортированном порядке.

2.1 Актуальность

В аналитической деятельности довольно часто существует необходимость представить некоторые статистические данные в отсортированном порядке по некоторому множеству заданных ключей. На первый взгляд это простая задача решается написанием правильного компаратора и использованием оптимизированных функций сортировки из стандартной библиотеки C++, однако подобный подход конечно же обречен на провал. Объемы данных, с которыми работает ClickHouse настолько огромен, что он элементарно может

не влезть в оперативную память компьютера, а в отдельных случаях может не хватить даже дискового пространства.

Для решение этих и других проблем была разработана определенная архитектура обработки запросов, представления данных и их передача между процессами. Однако в рамках всей системы сложно предусмотреть все, из-за чего в коде возникают места неэффективного использования ресурсов, лишние действия или неоптимальное использование уже вычисленной информации.

2.2 Цели и задачи

Основной целью курсовой работы является изучение способов оптимизации сортировок в ClickHouse и их реализация с целью оценить прирост в производительности каждого из них.

Для достижения данной цели были поставлены следующие задачи:

- ❏ Изучить существующие сортировки, которые используются в СУБД
- ❏ Изучить процесс выполнения обработки запросов, которые включают в себя модификатор ORDER BY
- ❏ Рассмотреть текущую реализацию обработки запросов с модификаторами ORDER BY & LIMIT. Оптимизировать неэффективную обработку блоков данных при частичной сортировке.
- ❏ Оптимизировать выполнение запросов с ORDER BY, уменьшив количество виртуальных вызовов при сортировке.
- ❏ Изучить текущую реализацию алгоритма сортировки LSD Radix Sort и оптимизировать ее
- ❏ Изучить и оптимизировать сортировку колонок-кортежей

2.3 Основные результаты

В ходе выполнения курсовой работы было сделано несколько подходов в оптимизации текущих алгоритмов, что дало прирост производительности обработки запросов с ORDER BY в десятки процентов.

3. Обзор существующих алгоритмов сортировки

3.1 Пирамидальная Сортировка

Пирамидальная сортировка (heapsort)[1] — алгоритм сортировки, основанный на на бинарной куче, благодаря которой гарантированно работает за $\Theta(n \cdot logn)$. Из преимуществ: имеет возможность сортировать данные частично (доставать первые k элементов), а также требует лишь $O(1)$ дополнительной памяти. Из недостатков: сортировка является неустойчивой (элементы с совпадающими ключами после сортировки могут изменить порядок относительно друг друга), а также ее нельзя ускорить параллельной обработкой данных. Используется например в Postgres[2] в случаях, когда пирамидальная сортировка выгоднее быстрой сортировки.

3.2 Быстрая сортировка

Быстрая сортировка (quicksort)[3] — алгоритм сортировки основанный на методе “разделяй и властвуй”, в среднем время работы алгоритма — $O(n \cdot logn)$. Из преимуществ: довольна проста в реализации, требует всего $O(logn)$ дополнительной памяти, можно ускорить с помощью параллельной обработки данных. Из недостатков: в худшем случае может отработать за $O(n^2)$, что довольно плохо, поэтому в чистом виде практически нигде не используется, тем не менее с рядом модификаций используется например в Postgres[4].

3.3 pdqsort

Pattern-defeating quicksort[5] — относительно недавно придуманный алгоритм сортировки, объединяющий в себе быстроту быстрой[3] сортировки в среднем и быстроту сортировки кучей[1] в худшем случае с линейной скоростью на данных удовлетворяющих определенным паттернам. В худшем случае работает за $O(n \cdot logn)$ и требует $O(logn)$ дополнительной памяти. Используется например в ClickHouse[6].

3.4 Поразрядная сортировка

Поразрядная сортировка (Radix sort)[7] — алгоритм сортировки, который сильно отличается от предыдущих, так как не использует сравнения элементов, сортировка производится с помощью раскладывания элементов по корзинам равным числу текущего разряда, если у элемента больше одного разряда, то процесс повторяется для каждого. Работает за $O(w \cdot n)$, где w — число разрядов у элементов и требует $O(w + n)$ дополнительной памяти. Существует два типа поразрядной сортировки, от младших разрядов к старшим и наоборот.

3.4.1 От младших к старшим (Least significant digit, LSD)

Является устойчивой сортировкой, может быть реализована без рекурсии из-за чего в среднем работает чуть лучше, чем MSD Radix sort. Используется в ClickHouse[6] для сортировки числовых данных.

3.4.2 От старших к младшим (Most significant digit, MSD)

Обычно реализуется в виде неустойчивой сортировки. Использует рекурсию, из-за чего немного проигрывает в производительности LSD Radix sort, зато позволяет делать частичную сортировку.

4. Наш вклад

4.1 Оптимизация сортировки с модификатором LIMIT

Упрощенно обработка запросов в ClickHouse происходит с помощью некоторых интерпретаторов `IInterpreter`, которые в процессе изучения AST-дерева, построенного на основе самого запроса, выстраивают некоторый граф обработки вершины которого являются одной из сущностей ClickHouse, называемая процессором `IProcessor`. Процессор принимает в себя данные по входящим ребрам, некоторым образом обрабатывает их и прокидывает дальше на выходящие в другие процессоры. Данные между собой процессоры передают в блоках `Block`. Это некоторое множество колонок с метаинформацией (названия столбцов и их типы). [8]

В настоящий момент в ClickHouse реализованы десятки типов процессоров для выполнения различных задач: от форматирования вывода, до сортировок слиянием. При стандартном запросе с ORDER BY & LIMIT для сортировки будет построен граф обработки с последовательным выполнением следующих процессоров: `PartialSortingTransform -> MergeSortingTransform`. Первый процессор принимает на вход поток блоков, которые сортирует с помощью функции `sortBlock(block, description, limit)`. Данная функция после сортировки оставляет первые `limit` строк. Затем `PartialSortingTransform` передает данные в процессор `MergeSortingTransform`, который использует слияние для объединения отсортированных блоков из предыдущего процессора. На выходе `MergeSortingTransform` получаем поток отсортированных блоков.

Во всей этой схеме наблюдает неэффективное использование ресурсов при сортировки заведомо лишних элементов. Так как блоки в процессоре обрабатываются последовательно, то можно поддерживать некоторый порог, по которому мы будем удалять строки из блока, заведомо не попадающие в `limit`

элементов окончательного ответа сервера. Для этого в `PartialSortingTransform` поддерживаем строку, относительно которой выполняем предфильтрацию блока перед функцией сортировки. После каждого отсортированного блока пробуем улучшить наш порог, используя `block[limit-1]` элемент.

4.1.1 Выводы и результаты

Данных подход на бенчмарках с LIMIT показал прирост в производительности порядка 57%.

```
localhost:9000, queries 500, QPS: 54.534, RPS: 57182724.752, MiB/s: 436.270,
 result RPS: 5453.370, result MiB/s: 0.042.

0.000%          0.017 sec.
10.000%         0.017 sec.
20.000%         0.018 sec.
30.000%         0.018 sec.
40.000%         0.018 sec.
50.000%         0.018 sec.
60.000%         0.018 sec.
70.000%         0.019 sec.
80.000%         0.019 sec.
90.000%         0.020 sec.
95.000%         0.020 sec.
99.000%         0.021 sec.
99.900%         0.023 sec.
99.990%         0.023 sec.
```

Рис. 4.1.1 Результаты бенчмарка на неоптимизированной версии сортировки

```
localhost:9000, queries 500, QPS: 85.318, RPS: 89462409.392, MiB/s: 682.544,
 result RPS: 8531.800, result MiB/s: 0.065.

0.000%          0.010 sec.
10.000%         0.011 sec.
20.000%         0.011 sec.
30.000%         0.011 sec.
40.000%         0.011 sec.
50.000%         0.012 sec.
60.000%         0.012 sec.
70.000%         0.012 sec.
80.000%         0.012 sec.
90.000%         0.013 sec.
95.000%         0.013 sec.
99.000%         0.014 sec.
99.900%         0.016 sec.
99.990%         0.016 sec.
```

Рис.4.1.2 Результаты бенчмарка на оптимизированной версии сортировки

4.2 Оптимизация сортировок сокращением количества виртуальных вызовов

ClickHouse - это столбцовая база данных, и следовательно, в отличии от более распространенных строчных аналогов, в нем данные хранятся отдельно по столбцам. За каждый тип данных отвечает отдельный класс: ColumnString, ColumnVector, ColumnInt64 и т.п. Все эти классы являются наследниками интерфейса IColumn. В каждом из классов реализованы методы для хранения и работы с данными соответствующего типа, а Column предоставляет возможность осуществлять неявные виртуальные вызовы этих методов. Наиболее важным для нас будет являться метод столбца comapeAt:

```
virtual int compareAt(size_t n, size_t m, const IColumn & rhs, int nan_direction_hint);
```
C++

Рис. 2.4.1 compareAt

Он принимает на вход второй столбец и два индекса - позиции на которых нужно сравнить элементы этих столбцов. И возвращает одно из трех значений -1, 0 или 1 означающие, что первый элемент меньше, равен, или больше второго соответственно.

Данный метод активно используется при сортировке. Когда пользователь делает любой SQL запрос с ORDER BY (сортировкой) по двум или более столбцам, запускается функция SortBlock, которая сортирует строки таблицы, используя следующий компоратор:

```C++
struct PartialSortingLess
{
    const ColumnsWithSortDescriptions & columns;

    explicit PartialSortingLess(const ColumnsWithSortDescriptions & columns_)
        : columns(columns_) {}

    bool operator() (size_t a, size_t b) const
    {
        for (const auto & elem : columns)
        {
            int res;
            if (elem.column_const)
                res = 0;
            else
                res = elem.description.direction * elem.column->compareAt(a, b,
                    *elem.column, elem.description.nulls_direction);
            if (res < 0)
                return true;
            else if (res > 0)
                return false;
        }
        return false;
    }
};
```

Рис.2.4.2 Компоратор для сортировки

Для каждой операции сравнения строк в цикле по каждому столбцу вызывается виртуальная функция Icolumn::compareAt

Это не оптимально - как из-за короткого цикла по неизвестному в compile-time количеству элементов, так и из-за виртуальных вызовов. Задачей которая перед нами стояла было уменьшить количество виртуальных вызовов, и тем самым ускорить выполнение сортировки всех таблиц в ClickHouse. Чтобы достичь поставленной цели, для каждого типа столбцов был разработан новый метод, названный updatePermutaion. Данный метод принимает частично отсортированную перестановку и набор интервалов. И сортирует перестановку

в пределах каждого интервала из набора отдельно. А затем формирует новый набор интервалов, такой что на каждом интервале элементы перестановки равны по компаратору этого столбца.

Данный метод позволил нам избавиться от огромного количества виртуальных вызовов. Если раньше для таблицы из n строк и m столбцов в худшем случае необходимо было сделать $n \cdot log(n) \cdot m$ виртуальных вызовов compareAt, то теперь необходимы всего лишь m виртуальных вызовов updatePermutation. Что не могло не отразиться на скорости работы ClickHouse.

4.2.1 Выводы и результаты

Как можно видеть на представленном изображение (рис. 4.2.3), тесты производительности показали прирост скорости выполнения некоторых классов запросов к ClickHousena 30%. Что довольно много. Учитывая то как часто возникает необходимость в том или ином виде сортировать данные, решение этой задачи позволит конечным пользователям ClickHouse сохранить огромное количество своего и процессорного времени.

Changes in performance

Old, s.	New, s.	Relative difference (new – old) / old	p < 0.001 threshold	Test
0.2513	0.1742	-0.307	0.305	logical_functions_small
0.4298	0.3242	-0.246	0.243	mingroupby-orderbylimit1
0.1238	0.0951	-0.232	0.228	joins_in_memory_pmj
0.1224	0.0955	-0.22	0.218	joins_in_memory_pmj
0.0653	0.0519	-0.206	0.195	parse_engine_file
0.344	0.286	-0.169	0.165	logical_functions_medium
0.3137	0.2653	-0.155	0.151	mingroupby-orderbylimit1
0.0849	0.0729	-0.142	0.14	string_sort
0.117	0.1034	-0.117	0.111	string_sort
0.093	0.084	-0.097	0.09	string_sort
0.1087	0.0983	-0.096	0.092	string_sort
1.2477	1.1311	-0.094	0.09	constant_column_search
0.1227	0.1115	-0.092	0.087	string_sort
1.1583	1.055	-0.09	0.081	constant_column_search
0.2052	0.1906	-0.072	0.069	string_sort
0.6292	0.5848	-0.071	0.069	array_reduce
0.2271	0.211	-0.071	0.064	string_sort
0.421	0.3933	-0.066	0.06	array_reduce

Рис. 4.2.3 Изменения в производительности

4.3 Оптимизация сортировки LSD Radix Sort

4.3.1 Устройство ClickHouse

В ClickHouse сортировки устроены таким образом, что они не меняют исходные данные, а лишь вычисляют нужную перестановку, применив которую к этим данным они станут отсортированными. В связи с этим использование поразрядной сортировки немного осложнено, так как этой сортировке требуется изменять порядок значений элементов. Тем не менее в ClickHouse эту проблему решили с помощью склеивания самих данных с их исходными индексами, получив новые временные данные, которые в свою очередь отправляются на сортировку. Таким образом после окончания сортировки достаточно будет лишь пройтись по второй части новых данных (после чего от них можно будет избавиться), чтобы извлечь индексы и получить искомую перестановку.

4.3.2 Задача

У этого подхода есть небольшой недостаток, он заключается в том, что сортировка в целом работает с большим объемом данных, чем это задумывалось изначально, хоть и побороться с этим особо не получится, тем не менее можно немного улучшить ситуацию с помощью отказа от временных данных на последней итерации сортировки и прямым переносом индексов в массив перестановки. Таким образом мы избавляемся от лишнего копирования данных, что может улучшить производительность при больших объемах данных

4.3.3 Выводы и результаты

К сожалению, этот метод не показал значительных улучшений во времени сортировки, тем не менее его реализация была добавлена в основную кодовую базу ClickHouse, так как лишние копирование данных точно не идет на пользу,

также возможно при сортировке более тяжелых данных (сейчас в ClickHouse LSD Radix sort используется только для сортировки числовых данных) это может сыграть большую роль, чем сейчас.

5. Заключение

В ходе выполнение курсовой работы была изучена архитектура СУБД ClickHouse, а также реализовано несколько подходов в оптимизации сортировок при запросах с модификатором ORDER BY:

- При частичной сортировки данных были оптимизированы лишние сравнения для элементов, заведомо не попавших в ответ.
- Для оптимизации сортировок кортежей была реализована идея поразрядной сортировки.
- Было убрано лишние копирование данных для оптимизации реализованной в ClickHouse LSD Radix Sort.

Суммарно в результате данной работы производительность ClickHouse при обработке запросов с сортировкой данных выросла на несколько десятков процентов.

6. Источники

1. Contributors to Wikimedia projects. Пирамидальная сортировка. Фонд Викимедиа, 2006.
2. git.postgresql.org Git - postgresql.git/blob - src/backend/utils/sort/tuplesort.c [Электронный ресурс]. URL: https://git.postgresql.org/gitweb/?p=postgresql.git;a=blob;f=src/backend/utils/sort/tuplesort.c;hb=6a918c3ac8a6b1d8b53cead6fcb7cbd84eee5750#l1834 (дата обращения: 27.05.2020).
3. Contributors to Wikimedia projects. Быстрая сортировка. Фонд Викимедиа, 2005.
4. git.postgresql.org Git - postgresql.git/blob - src/backend/utils/sort/tuplesort.c [Электронный ресурс]. URL: https://git.postgresql.org/gitweb/?p=postgresql.git;a=blob;f=src/backend/utils/sort/tuplesort.c;h=d59e3d5a8d5f30060a6db888c3620f2b9bf7430f;hb=6a918c3ac8a6b1d8b53cead6fcb7cbd84eee5750#l17 (дата обращения: 27.05.2020).
5. orlp. orlp/pdqsort [Электронный ресурс] // GitHub. URL: https://github.com/orlp/pdqsort (дата обращения: 27.05.2020).
6. ClickHouse. ClickHouse/ClickHouse [Электронный ресурс] // GitHub. URL: https://github.com/ClickHouse/ClickHouse (дата обращения: 27.05.2020).
7. Contributors to Wikimedia projects. Radix sort. Wikimedia Foundation, Inc., 2001.
8. ClickHouse. Architecture Overview | ClickHouse Documentation [Электронный ресурс] // Yandex LLC. 2017. URL: https://clickhouse.tech/docs/en/development/architecture/ (дата обращения: 27.05.2020).