НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Факультет компьютерных наук
Основная образовательная программа
«Прикладная математика и информатика»

ПРОГРАММНЫЙ ПРОЕКТ НА ТЕМУ «ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНОГО GROUP BY С ПОМОЩЬЮ FLAT COMBINING»

Выполнил студент группы БПМИ163, 4 курса,
Серебряков Максим Викторович
Руководитель ВКР:
приглашенный преподаватель, Миловидов Алексей Николаевич

Москва, 2020

ВВЕДЕНИЕ

Что такое ClickHouse?

- ClickHouse – это столбцовая система управления базами данных (СУБД) с открытым исходным кодом для онлайн обработки аналитических запросов (OLAP), разработанная в компании Яндекс
- Очень высокая производительность выполнения запросов и работы с данными, которая может достигать улучшения в десятки раз по сравнению с конкурентами

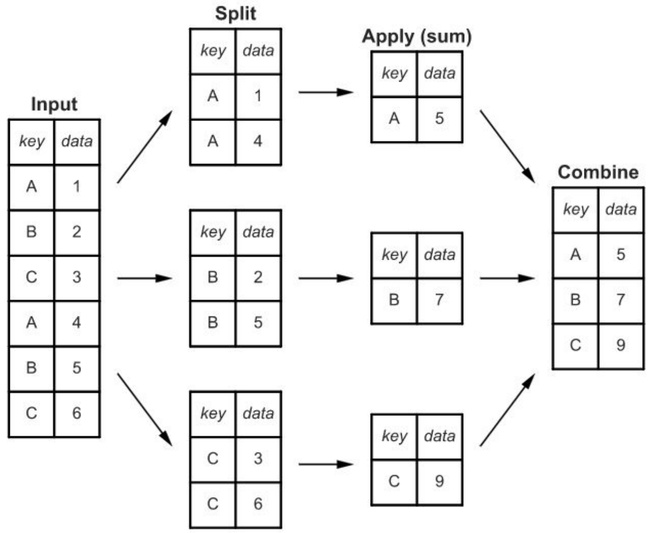
ВВЕДЕНИЕ

Что такое GROUP BY?

- SQL (structured query language — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.
- SELECT – один из операторов манипуляции с данными, выбирает данные, удовлетворяющие заданным условиям
- GROUP BY – один из SQL операторов, которые задают условия в запросах SELECT, с помощью которого производится агрегация данных по ключу
- Агрегация данных по ключу – данные делятся на группы с одинаковыми значениями ключей, а затем для каждой группы вычисляется агрегатная функция, например, количество элементов в группе, сумма значений по какому-либо полю среди элементов группы и так далее

```
[WITH expr_list|(subquery)]
SELECT [DISTINCT] expr_list
[FROM [db.]table | (subquery) | table_function] [FINAL]
[SAMPLE sample_coeff]
[ARRAY JOIN ...]
[GLOBAL] [ANY|ALL] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER] JOIN (subquery)|table USING columns
[PREWHERE expr]
[WHERE expr]
[GROUP BY expr_list] [WITH TOTALS]
[HAVING expr]
[ORDER BY expr_list]
[LIMIT [offset_value, ]n BY columns]
[LIMIT [n, ]m]
[UNION ALL ...]
[INTO OUTFILE filename]
[FORMAT format]
```

Синтаксис запроса SELECT в ClickHouse



Пример работы GROUP BY

АКТУАЛЬНОСТЬ И ЦЕЛИ РАБОТЫ

Почему оптимизация GROUP BY важна?

- Основное продуктовое свойство ClickHouse – высокая скорость обработки запросов
- Агрегация данных по ключу – это один из самых основных и популярных видов запросов, чем быстрее он выполняется, тем пользователь более доволен системой в целом
- Flat combining – это один из современных подходов к распараллеливанию структур данных, который на данный момент еще плохо исследован и редко применяется в продакшен разработке, в силу своей новизны, но может быть полезен для данной задачи

Цель:

- Попытаться ускорить текущий подход, используемый в ClickHouse, любыми новыми способами, возможно только для определенных случаев

Задачи:

- Реализация различных подходов для ускорения группировки данных в запросах с использованием GROUP BY
- Исследование метода flat combining на возможное его применение в ClickHouse
- Проведение сравнительного анализа реализованных подходов
- Реализация и применение самого эффективного алгоритма в ClickHouse в формате созданного pull-request в репозиторий ClickHouse на github.com
- Проверка реализованного метода на функциональных тестах и тестах на производительность

ОБЗОР СУЩЕСТВУЮЩИХ ПОДХОДОВ К АГРЕГАЦИИ

Структура данных для агрегации:

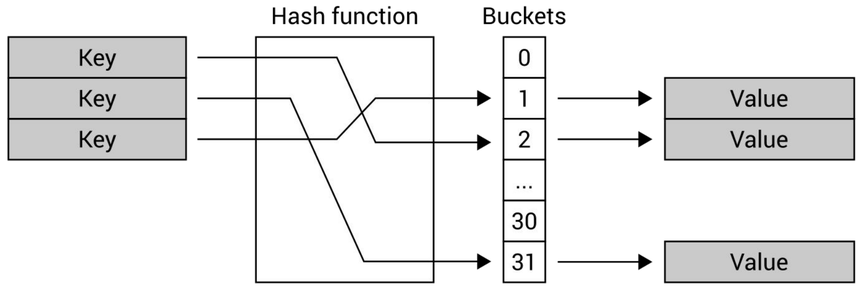
- Обычный массив не подходит – сортировка выполняется за O(n log n)
- Ассоциативный массив – это особый тип данных, позволяющий хранить пары вида “(ключ, значение)”, а так же искать, добавлять и удалять их

Реализации ассоциативного массив:

- Lookup-таблица
- Хеш-таблица
- Бинарное дерево
- Список с пропусками (skip-list)
- Би-дерево
- И так далее

Выбор подходящей реализации для агрегации данных:

- Нужны только две операции: быстрый поиск и добавление элемента
- Абсолютно не важен порядок элементов структуре
- Хеш-таблица – лучший вариант: поиск и вставка за O(1) по времени, объем используемой памяти – O(m), где m – количество уникальных ключей
- Получается, что тривиальный способ использования хеш-таблицы для агрегации работает за O(n), а объем потребляемой памяти O(m)



Пример работы хеш-таблицы

ОБЗОР СУЩЕСТВУЮЩИХ ПОДХОДОВ К АГРЕГАЦИИ

Тривиальный параллельный метод

Алгоритм:

- Изначально имеются k доступных потоков
- Каждый поток параллельно и независимо от других работает со своей частью данных и агрегирует ее в свою собственную хеш-таблицу
- Затем все k хеш-таблиц объединяются в одну в одном потоке последовательно
- Скорость работы: в лучшем случае – в k раз быстрее, в худшем – столько же сколько и тривиальный метод
- Объем потребляемой памяти такой же: O(m), где m – количество уникальных ключей

Плюсы:

- Простота, не надо использовать примитивы синхронизации потоков и заботиться о потоковой безопасности
- Стадия агрегации происходит очень быстро (в k раз быстрее, чем в тривиальном методе)

Минусы:

- Данный алгоритм сильно зависит от доли уникальных ключей в исходных данных, так как стадия объединения происходит в одном потоке: например, если все ключи различны, то стадия объединения будет работать за время O(n), то есть столько же, сколько и тривиальный метод в целом, при этом потратив время на параллельную агрегацию. Если же доля уникальных ключей мала, то данный метод работает очень эффективно.

ОБЗОР СУЩЕСТВУЮЩИХ ПОДХОДОВ К АГРЕГАЦИИ

Two-level метод

Two-level метод – метод агрегации, реализованный в ClickHouse.

Алгоритм:

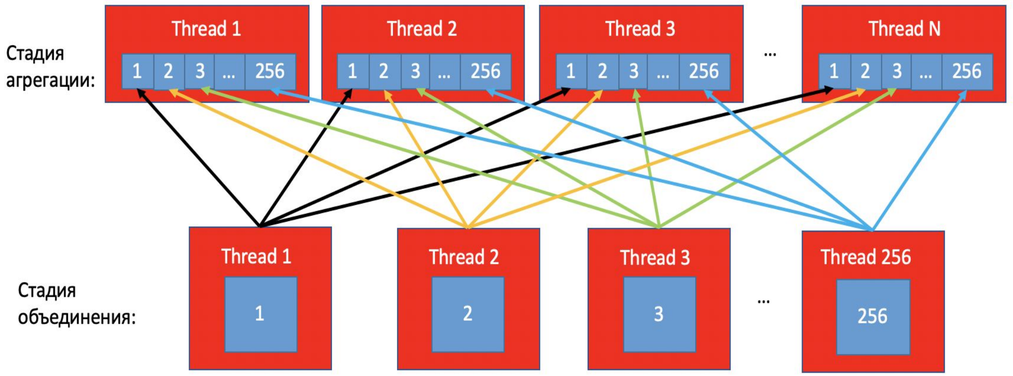
- Стадия подготовки: исходные данные делятся на части (например, ровно на k частей, где k – количество потоков). Создаются k Two-level хеш-таблиц. Данная стадия не занимает времени.
- Стадия агрегации: каждый поток агрегирует свою часть, используя свою Two-level хеш-таблицу. На выходе получаются k Two-level хеш-таблиц (k*256 обычных внутри них).
- Стадия объединения: каждому потоку назначается номер дочерней хеш-таблицы, и он объединяет k дочерних хеш-таблиц с этим номером в одну. То есть 256 номеров добавляются в очередь, и k потоков их разбирают параллельно. Независимость объединения от дочерних хеш-таблиц с другими номерами достигается за счет того, что в каждой из дочерних хеш-таблиц с одинаковым номером лежат ключи из одной группы, которых нет в других хеш-таблицах по свойству выше. В итоге получаем одну итоговую Two-level хеш-таблицу.

Плюсы:

- И стадия агрегации и стадия объединения выполняются параллельно без использования синхронизации потоков, что делает данный метод очень эффективным
- Нет зависимости от доли уникальных ключей

Минусы:

- Очевидных минусов нет



Визуализация метода Two-level.

ОБЗОР НОВЫХ РЕАЛИЗОВАННЫХ ПОДХОДОВ

Splitting-aggregator метод

Алгоритм:

- Стадия подготовки: сначала для всех ключей считаются значения хеш-функций, находится минимальное и максимально значение, и ключи разбиваются на k групп по значению хеш-функции (где k – количество потоков).
- Стадия агрегации: каждый поток параллельно с другими потоками обрабатывает только ключи из своей группы, агрегирует их в свою собственную хеш-таблицу. На выходе получаются k хеш-таблиц.
- Стадия объединения: не производится, так как в разных хеш-таблицах находятся разные ключи (ни у одной пары хеш-таблиц нет хотя бы одного общего ключа).

Плюсы:

- Простота
- Отсутствие стадии объединения

Минусы:

- Зависимость от доли уникальных ключей: чем меньше, тем медленнее работает алгоритм

ОБЗОР НОВЫХ РЕАЛИЗОВАННЫХ ПОДХОДОВ

Splitting-aggregator метод с быстрым делением

Отличие от базовой версии алгоритма:

- Предварительный подсчет значений хеш-функции не делается
- Определение номера потока происходит “на лету” с помощью формулы, которая заменяет обычное взятие остатка от деления по модулю: $x * k \gg 32$, где x – значение хеш-функции, k – количество потоков, $\gg$ - операция битового сдвига вправо.
- Вычисление данной формулы работает в 4 раза быстрее обычного деления

Плюсы:

- На практике работает быстрее базового splitting-aggregator метода

ОБЗОР НОВЫХ РЕАЛИЗОВАННЫХ ПОДХОДОВ

Локальные хеш-таблицы + общая Two-level хеш-таблица

Алгоритм:

- Стадия подготовки: данные делятся на k равных частей, где k – количество потоков. Заводится одна общая Two-level хеш-таблица. Никаких вычислений не производится.
- Стадия агрегации: каждый поток сначала добавляет в свою собственную локальную хеш-таблицу ключи, но делает это до момента, пока ее размер (количество уникальных ключей) меньше порога (например, 4096). Если же размер локальной хеш-таблицы больше, то тут два варианта:
    - Если ключ имеется в локальной хеш-таблице, то он кладется туда (пересчитывается значение агрегатной функции)
    - Если же в локальной хеш-таблице ключа нет, то производится попытка вставить его в общую Two-level хеш-таблицу, при этом взяв блокировку на определенную дочернюю хеш-таблицу, используя примитивы синхронизации потоков. Это необходимо, так как в одинаковый момент времени разные потоки могут вставлять ключ в одну и ту же дочернюю хеш-таблицу, что без блокировки приведет к аварийному завершению программы. Если же взять блокировку не удалось, то ключ вставляется в локальную хеш-таблицу.
- Стадия объединения: по очереди объединяются все локальные хеш-таблицы с общей Two-level хеш-таблицей в одном потоке.
- Алгоритм lock-free: для синхронизации используются атомарные переменные с атомарными операциями “сравнение с обменом” (CAS)

Плюсы:

- Данный алгоритм работает очень эффективно, если доля уникальных ключей не велика, так как обе стадии выполняются быстро:
    - В стадии агрегации большинство ключей попадают в локальную таблицу без синхронизаций
    - В стадии объединения участвуют только локальные хеш-таблицы, размер которых мал

Минусы:

- Зависимость от доли уникальных ключей: чем больше, тем больше синхронизаций и локальные таблицы начинают разрастаться, что приводит к медленной стадии объединения

ОБЗОР НОВЫХ РЕАЛИЗОВАННЫХ ПОДХОДОВ

Локальные хеш-таблицы + Splitting aggregator с очередями

Является улучшением алгоритма Splitting-aggregator для случаев, когда доля уникальных ключей мала.
Алгоритм:

- Стадия подготовки: данные делятся на k групп, где k – количество потоков. Создаются k глобальных хеш-таблиц, каждая предназначенная для своего потока, и содержащая только свои ключи. Создаются k очередей для каждого потока, то есть всего $k * k$ очередей.
- Стадия агрегации: работа с локальной хеш-таблицей происходит аналогично предыдущему методу, но с отличием, когда в локальной хеш-таблице уже больше порога ключей и очередной ключ в ней не нашелся. Для такого ключа определяется номер потока, которому он принадлежит: если текущему, то элемент кладется в глобальную хеш-таблицу для текущего потока, если же другому, то добавляется в очередь для этого потока, то есть в очередь с индексами $[i,j]$, где i – номер текущего потока, а j – номер потока, которому принадлежит данный ключ. Собрав какое-то количество ключей для других потоков, для диапазона с ними в буффере выставляется флажок (атомарная переменная), что означает, что диапазон готов для чтения. Потоки раз в какое-то количество итераций смотрят на готовые для них диапазоны в буффере, читают ключи из них и добавляют в свою часть глобальной хеш-таблицы.
- Стадия объединения: в глобальных хеш-таблицах уже лежат разбитые на группы ключи, поэтому остается только добавить в них ключи из локальных хеш-таблиц. Так как локальные хеш-таблицы маленькие, то объединение происходит в одном потоке без синхронизации.
- Алгоритм lock-free: для синхронизации используются атомарные переменные с атомарными операциями “сравнение с обменом” (CAS)

Плюсы:

- Алгоритм работает быстрее обычного Splitting-aggregator на малой долей ключей, на средних и больших примерно столько же.

Минусы:

- Реализации сложнее, чем у остальных методов
- Расходуется немного больше памяти из-за создания буфферов

ОБЗОР НОВЫХ ПОДХОДОВ

Flat combining

Основные особенности:

- Flat combining – это новая парадигма синхронизации для построения конкурентных структур данных.
- Главные объекты: мьютекс и список анонсов
- У каждого потока запись в списке анонсов хранится локальном хранилище (TLS)
- Поглощающая стратегия (elimination back-off) – позволяет производить взаимопоглощающие операции без реального их выполнения на структуре данных (например, push/pop для стека)

Почему данный метод не подходит:

- Результат операции не нужен (происходит только добавление элемента в хеш-таблицу)
- Нет слишком большой конкурентности между потоками (в большинстве случаев количество потоков не превышает 32)
- Невозможно реализовать поглощающую стратегию
- Очень сложный код, который трудно внедрить в ClickHouse

Но идея выполнения операций массово из flat combining была использована для создания нового метода

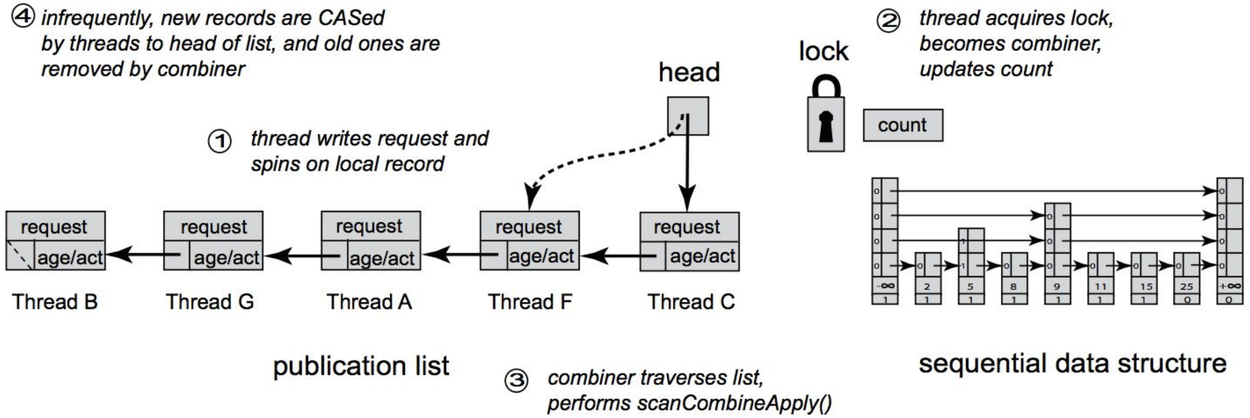


Схема подхода flat combining

ОБЗОР НОВЫХ РЕАЛИЗОВАННЫХ ПОДХОДОВ

Локальные хеш-таблицы с буфферами + общая Two-level хеш-таблица

Является улучшением алгоритма “Локальные хеш-таблицы + общая Two-level хеш-таблица”, с использованием идеи из flat combining.
Алгоритм:

- Стадия подготовки: такая же, как в третьем методе.
- Стадия агрегация: аналогичная третьему методу, но с существенным изменением – если потоку не удается получить блокировку на дочернюю хеш-таблицу в Two-level хеш-таблице, то он кладет ключ не в локальную хеш-таблицу, а в свой локальный буффер для определенной дочерней хеш-таблицы. Если же удается взять блокировку, то он добавляет все ключи из буффера для этой же дочерней хеш-таблицы вместе с текущим ключом в нее за раз. Этим как раз метод и похож на flat combining, где операции выполняются комбайнером массово. После того, как все элементы обработаны, оставшиеся ключи из буфферов, которые не были добавлены в общую хеш-таблицу, добавляются в локальную.
- Стадия объединения: аналогичная третьему методу.

Плюсы:

- Локальные хеш-таблицы не разрастаются и почти не выходят за порог по количеству уникальных ключей, в следствие чего, стадия объединения происходит практически мгновенно даже в худшем случае

Минусы:

- Очевидных минусов нет

ОПИСАНИЕ СПОСОБА ТЕСТИРОВАНИЯ

Тестирование производилось:

- На различных реальных анонимизированных и искусственных наборах данных
- На разных долях уникальных ключей
- На разных объемах данных (от 10^3 до $2 * 10^8$)
- С использованием различного количества потоков (16, 24 и 32)
- На сервере с 32 ядрами и 24GB оперативной памяти

Технические характеристики:

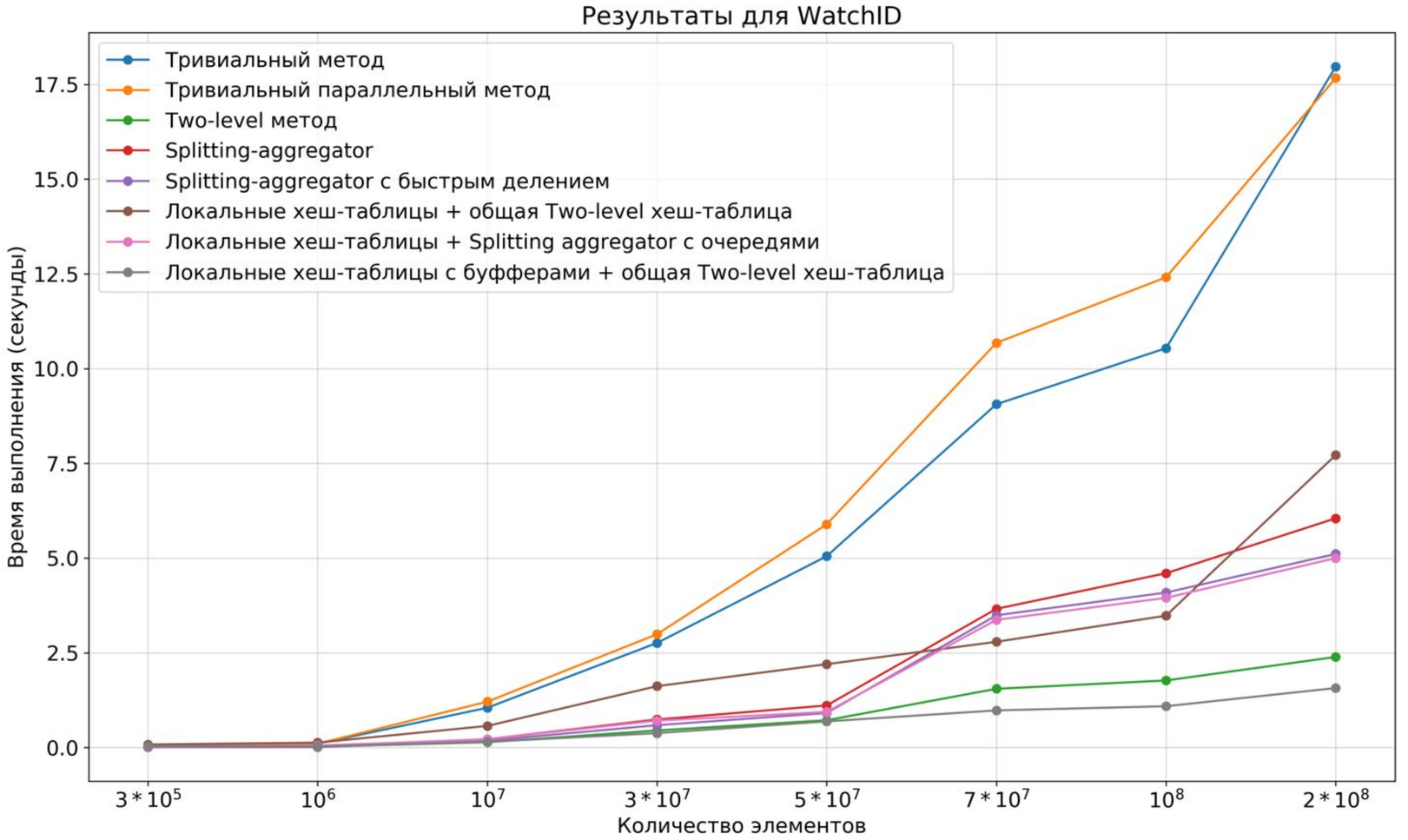
- Код был написан на языке программирования C++
- Используемый компилятор – gcc-9
- Операционная система – Ubuntu 18.04.2 LTS

Для автоматизации тестирования был написан бенчмарк.

Для описания результатов самыми репрезентативными наборами данных являются:

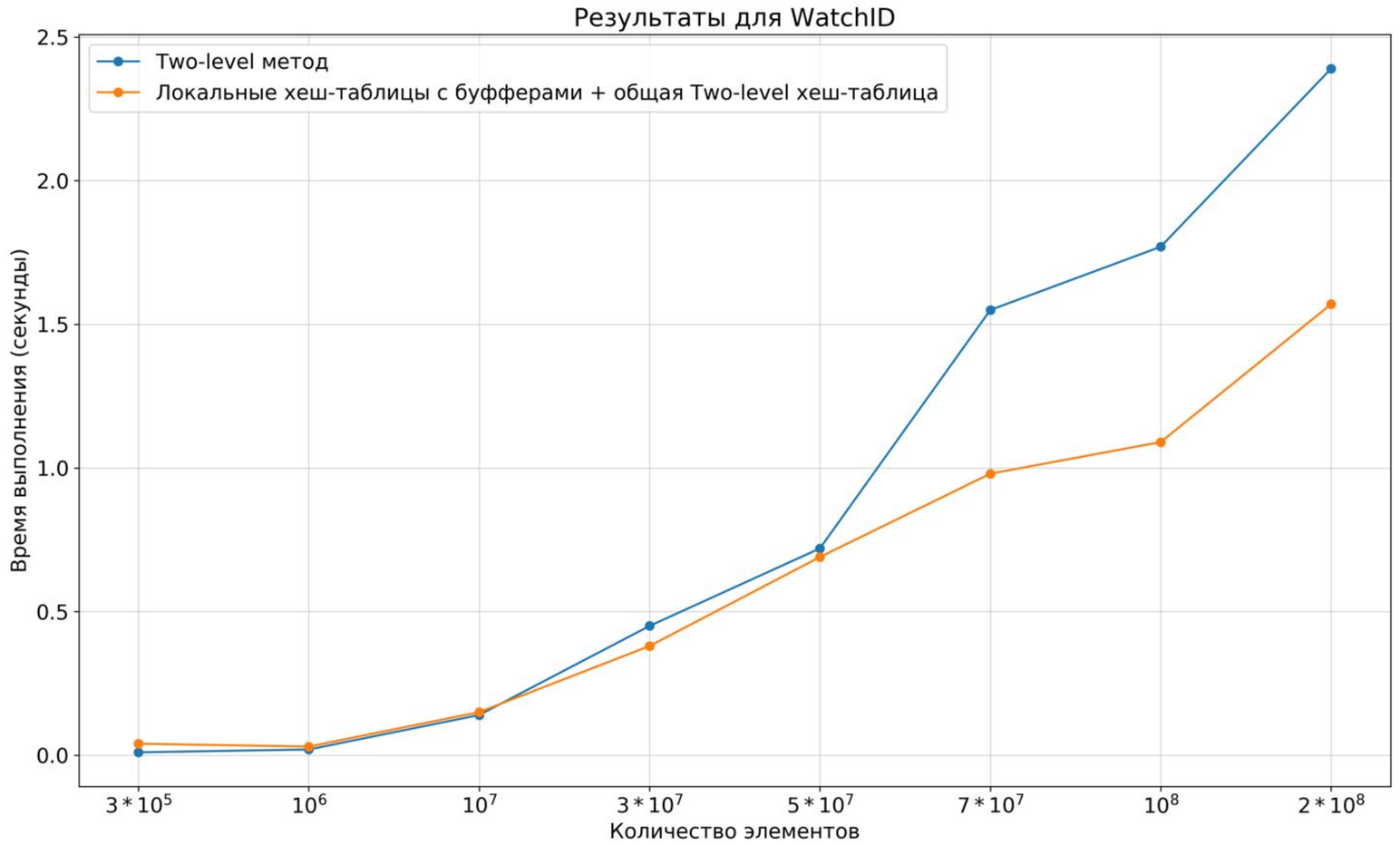
- WatchID – доля уникальных ключей 99.99%.
- RefererHash – доля уникальных ключей 21.5%
- SearchPhrase – доля уникальных ключей 6%

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ



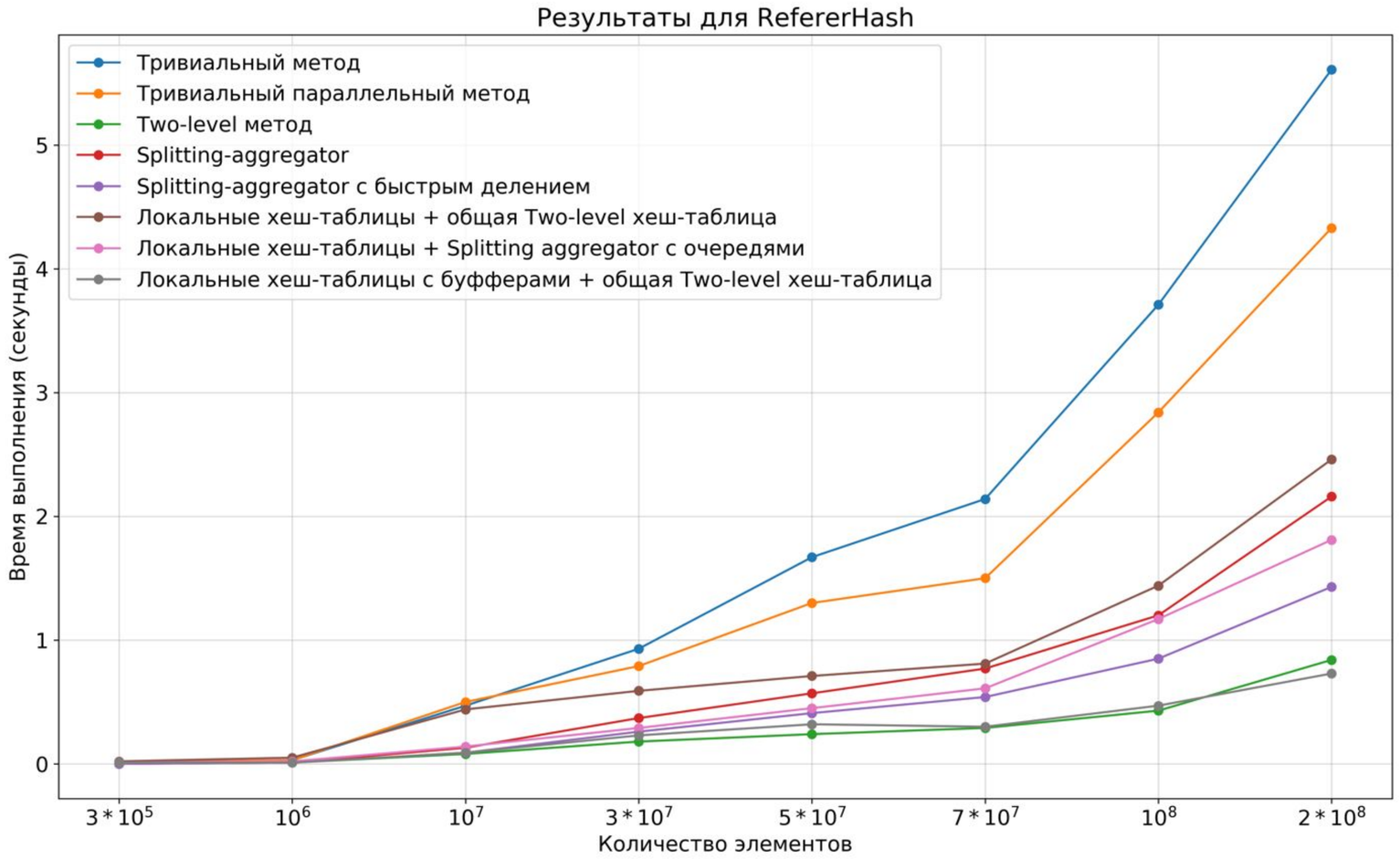
Результаты тестирования для набора данных WatchID (доля уникальных ключей 99.99%)

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ



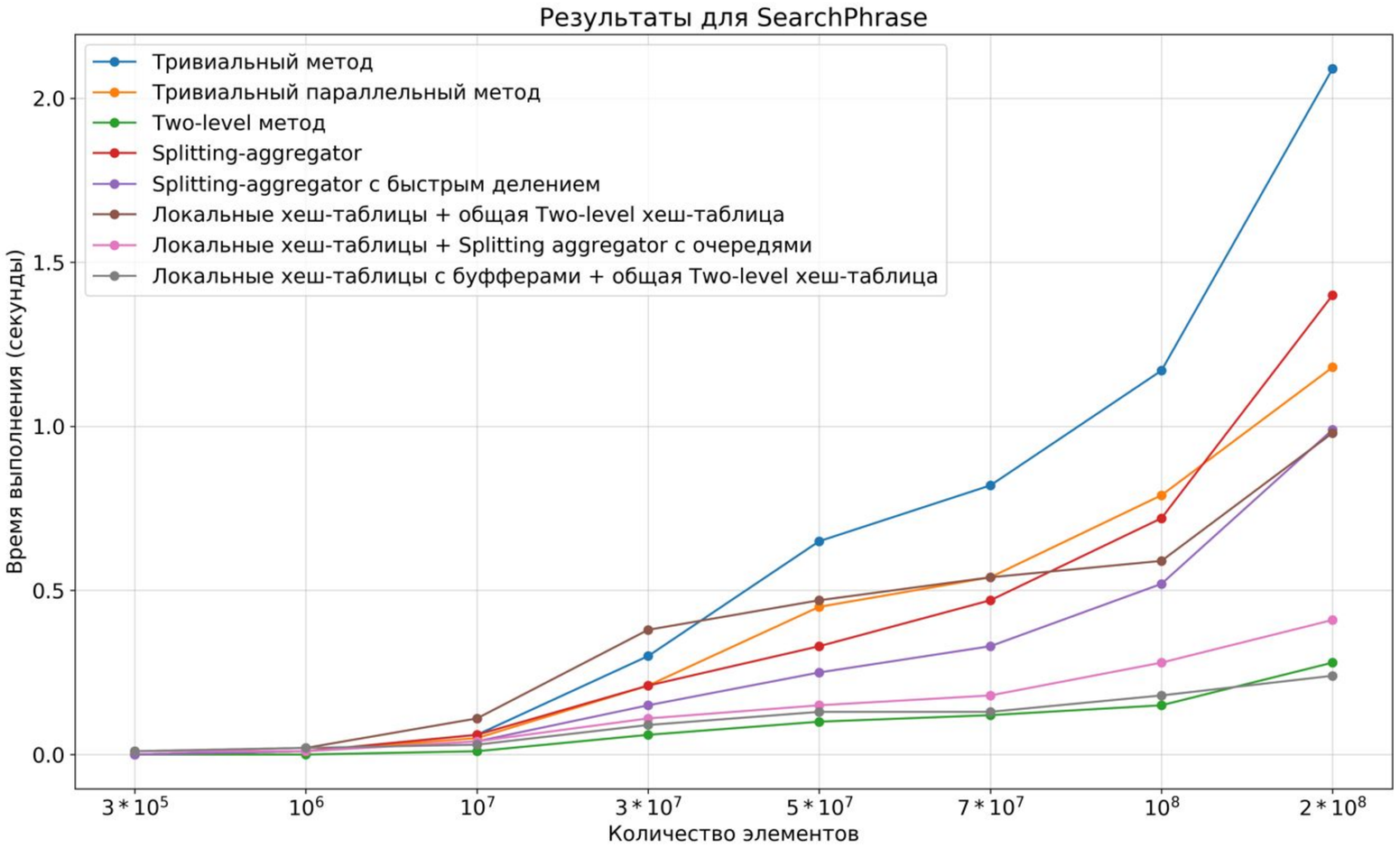
Результаты тестирования для набора данных WatchID (доля уникальных ключей 99.99%)

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ



Результаты тестирования для набора данных RefererHash (доля уникальных ключей 21.5%)

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ



Результаты тестирования для набора данных SearchPhrase (доля уникальных ключей 6%)

ВЫВОДЫ ПОСЛЕ ТЕСТИРОВАНИЯ

В совокупности на различных типах данных самыми эффективным оказались методы:

- Для маленького количества данных – тривиальный способ с одной хеш-таблицей, так как все методы отрабатывают почти мгновенно, но этот является самым простым в реализации и использует меньше всего памяти.
- Для большого количества данных, если доля уникальных ключей небольшая (меньше 35%) – “Two-level метод”, алгоритм, который используется в ClickHouse сейчас.
- Для большого количества данных, если доля уникальных ключей велика (больше 35%) – реализованный новый алгоритм “Локальные хеш-таблицы с буфферами + общая Two-level хеш-таблица”.

Поэтому метод новый алгоритм “Локальные хеш-таблицы с буфферами + общая Two-level хеш-таблица” был применен и реализован в ClickHouse в формате pull-request в репозиторий на github.com: https://github.com/ClickHouse/ClickHouse/pull/10956/

ОСОБЕННОСТИ ПРИМЕНЕНИЯ В CLICKHOUSE

- Все операции в ClickHouse выполняются по блокам
- Блок – это часть данных из таблицы, максимальный размер которого ограничен и по умолчанию равен 65536
- Нужен, чтобы сократить потребление оперативной памяти, так как в таблице могут храниться гигабайты данных
- Агрегация так же выполняется по блокам
- Для определения целесообразности применения нового метода нужно как-то определить долю уникальных ключей для данных, на которых выполняется GROUP BY запрос
- Поэтому был придуман эвристический способ: определять долю уникальных ключей только по первому блоку, если в нем доля уникальных ключей велика, то предполагается, что такое же распределение ключей будет и в оставшихся блоках, и используется новый метод вместо Two-level

Плюсы:

- Данный способ походит для большинства наборов данных

Минусы:

- Не подходит для следующего случая: ключи состоят из нескольких групп, в каждой из которых элементы имеют значения от 0 до k, где k – число, большее максимального размера блока. То есть, если максимальный размер блока 65536, всего элементов 10^8, а k равно 10^5, то для набора данных $0..10^5,0..10^5,...,0..10^5$, будет использован новый метод, хотя доля уникальных ключей в нем составляет всего 0.001%.

Данную проблему можно решить, если определять уникальность данных не после первого блока, а когда после обработки хотя бы 1% данных.

РЕЗУЛЬТАТЫ ПРИМЕНЕНИЯ В CLICKHOUSE

Тестирование новой функциональности происходило на множестве тестов:

- Функциональных
- На производительность
- Интеграционных
- На компилируемость
- Форматирование кода

Все тесты кроме проверяющих производительность прошли без изменений, то есть были пройдены.

Тесты на производительность ожидаемо показали:

- Сильное ускорение на данных с большой долей уникальных ключей

Запрос	Время выполнения без нового метода, в секундах	Время выполнения с новым методом, в секундах	Абсолютное ускорение, в секундах	Относительное ускорение	Уникальность ключей
SELECT number % toUInt32(1e8) AS k, count() FROM numbers_mt(toUInt32(1e8)) GROUP BY k FORMAT Null	1.8195	1.0905	-0.729	-0.401	100%
SELECT number % 10000000 AS k, count() FROM numbers_mt(80000000) GROUP BY k FORMAT Null	0.9153	0.6163	-0.299	-0.327	12.5%

Пример результатов тестов на производительность

ЗАКЛЮЧЕНИЕ

- Были предложены и реализованы новые и уже существующие способы агрегации данных
- Было произведено тестирование каждого метода
- На полученных результатах тестирования был проведен сравнительный анализ
- Был исследован метод flat combining
- Лучшие результаты показал новый придуманный метод “Локальные хеш-таблицы с буфферами + общая Two-level хеш-таблица”, который сочетает в себе идеи из метода flat combining и использования Two-level хеш-таблиц
- Данный метод был реализован в ClickHouse в виде pull-request в репозиторий на github.com
- Было произведено тестирование нового метода в ClickHouse, которое показало значительное ускорение на запросах с GROUP BY

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ