ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: Визуализация и оптимизация алгоритма слияния

Выполнил:
Студент группы БПМИ223 _____ Матвеев Денис Дмитриевич
Подпись ФИО

18.09.2024
Дата

Принял руководитель проекта:
Карпов Максим Евгеньевич
Старший преподаватель
ФКН НИУ ВШЭ

Соруководитель:
Миловидов Алексей Николаевич
Технический директор
ClickHouse Inc

Москва 2024

Содержание

Аннотация

Цель данного проекта заключается в разработке системы визуализации и оптимизации слияния данных для ClickHouse сервера. Разработано веб-приложение для визуализации процесса слияния, проведен экспериментальный анализ эффективности алгоритма и предложены рекомендации по его оптимизации.

1 Введение

Современные системы управления базами данных (СУБД) играют ключевую роль в обработке и хранении больших объемов данных, особенно в аналитических задачах, где важна скорость выполнения запросов. В отличие от традиционных СУБД, где данные хранятся по строкам, столбцовые СУБД хранят данные каждого столбца отдельно, что позволяет считывать с диска только те данные, которые непосредственно участвуют в запросе. ClickHouse — одна из самых популярных и мощных систем для аналитической обработки данных, которая активно используется многими компаниями по всему миру. Она была разработана в 2009 году в компании "Яндекс". Благодаря своей архитектуре, ClickHouse является востребованным инструментом среди специалистов по работе с данными. Одним из ключевых компонентов производительности ClickHouse является алгоритм слияния данных SimpleMergeSelector, который управляет процессами слияния сегментов данных на сервере. Эффективность этого алгоритма напрямую влияет на общую производительность системы, особенно в условиях высоких нагрузок и больших объемов данных.

В рамках данного проекта предстоит оптимизировать алгоритм слияния ClickHouse для повышения производительности в условиях больших объемов данных и интенсивных нагрузок. Для достижения этой цели будет разработано веб-приложение для визуализации процессов слияния данных, а также проведен экспериментальный анализ, направленный на выявление оптимальных параметров алгоритма и разработку рекомендаций по его улучшению.

2 Обзор литературы

ClickHouse [4] разработанна на языке программирования C++, использует систему сборки CMake и утилиту Ninja, что обеспечивает её кроссплатформенную совместимость и гибкость в настройке. В основе её архитектуры лежит структура MergeTree [1], которая эффективно управляет хранением и обработкой данных.Особое внимание в данном проекте уделено алгоритму SimpleMergeSelector [3], его текущая реализация и описание работы доступны в открытом исходном коде.

3 Описание функциональных и нефункциональных требований к программному проекту

В результате работы будет предложена комплексная система программ, предназначенная для визуализации и анализа процессов слияния данных в ClickHouse, а также ряд рекомендаций по настройке и улучшению алгоритма слияния. Проектируемая система будет состоять из нескольких ключевых компонентов:

- **Разработка веб-приложения для визуализации процессов слияния данных:**
  - *Язык реализации*: HTML, CSS, JavaScript.
  - *Функциональность*: Веб-приложение должно запрашивать данные из таблицы `system.part_log` ClickHouse и визуализировать текущее состояние сегментов данных в виде прямоугольников, которые отображаются на основе их размеров и позиций в системе. Визуализация должна поддерживать анимацию процессов слияния данных, обновляя информацию в реальном времени.
- **Создание модуля для расчета метрик эффективности алгоритма SimpleMergeSelector:**
  - *Язык реализации*: Python.
  - *Функциональность*: Модуль должен анализировать данные из таблицы `system.part_log` и рассчитывать метрики.
- **Проведение экспериментального анализа и оптимизация алгоритма SimpleMergeSelector:**
  - *Язык реализации*: Python.

– *Функциональность*: Реализовать экспериментальные сценарии с изменением параметров алгоритма и их тестирование на данных, чтобы определить, как различные настройки влияют на метрики эффективности и подобрать наилучшую сетку параметров.

4 Визуализация процесса слияния

4.1 Проектирование архитектуры приложения

Первоначально был выбран веб-интерфейс для реализации визуализации, так как это наиболее универсальный и удобный способ предоставления информации. Веб-приложение разрабатывалось с использованием следующих технологий:

- **HTML, CSS**: для создания структуры страницы и оформления визуальных элементов.
- **JavaScript**: для динамического обновления данных в реальном времени и визуализации процессов слияния посредством анимаций.

4.2 Структура и стилизация веб-страницы

Для создания интерфейса визуализации процессов слияния данных был выбран язык HTML [6], который обеспечивает базовую структуру веб-страницы. Вёрстка страницы включает заголовок, элементы управления и область для отображения визуализированных данных. Область визуализации реализована через элемент `<div id="visualization">`, где в режиме реального времени отображаются сегменты данных и результаты их слияний. Также присутствует текстовое поле `<div id="action_text">`, которое информирует пользователя о текущих операциях и их статусе.

Для стилизации страницы и создания гибкого и современного интерфейса использовался CSS [5]. Прямоугольники, представляющие сегменты данных, стилизованы таким образом, чтобы их размеры отображали объём сегментов, а цвет менялся в зависимости от типа операции — создание нового сегмента или слияние. Анимация плавного появления и исчезновения сегментов добавлена для более наглядного отображения процесса слияния данных. Также стилизованы элементы управления: кнопки имеют цветовую индикацию и реагируют на наведение, увеличивая свой размер и меняя цвет, что повышает интерактивность интерфейса.

4.3 Настройка сервера ClickHouse

Для корректного отображения данных сервера ClickHouse о текущих процессах, необходимо произвести правильную настройку его конфигурационного файла.

Одним из ключевых параметров является `flush_interval_milliseconds`, который задает периодичность сброса данных из буфера в таблицу. Этот параметр измеряется в миллисекундах и определяет, как часто обновляются данные в таблице. Другой ключевой параметр — `buffer_size_rows_flush_threshold`, который определяет пороговое количество строк в буфере, после достижения которого происходит автоматический сброс данных в таблицу. Этот параметр напрямую влияет на частоту операций записи и помогает избежать перегрузки системы при интенсивной нагрузке на сервер.

```
<part_log>
    <database>system</database>
    <table>part_log</table>
    <partition_by>toYYYYMM(event_date)</partition_by>
    <flush_interval_milliseconds>7500</flush_interval_milliseconds>
    <max_size_rows>1048576</max_size_rows>
    <reserved_size_rows>8192</reserved_size_rows>
    <buffer_size_rows_flush_threshold>524288</buffer_size_rows_flush_threshold>
    <flush_on_crash>false</flush_on_crash>
</part_log>
```

Рис. 1: Пример настройки конфига config.xml для сервера ClickHouse.

4.4 Реализация модуля запроса данных

Для эффективного получения и обработки данных из таблицы `system.part_log` [2] в ClickHouse был разработан модуль, осуществляющий HTTP-запросы. Модуль выполняет стандартный SQL-запрос к базе данных:

`SELECT * FROM system.part_log FORMAT JSON.`

На выходе данные возвращаются в формате JSON, который удобен для обработки в веб-приложении благодаря своей структуре и легкости интеграции. Каждая запись в полученном наборе данных представляет собой информацию о конкретном сегменте данных и операции слияния, в которую этот сегмент вовлечен. Для дальнейшей визуализации нас интересуют следующие поля:

- **part_name**: уникальное имя сегмента.
- **event_type**: тип события, указывающий на операцию, произошедшую с сегментом (например, создание нового сегмента или его слияние).
- **event_date**: дата и время события.
- **size_in_bytes**: размер сегмента в байтах.
- **merged_from**: список сегментов, которые были объединены в текущий сегмент в результате операции слияния.

Для обеспечения корректности визуализации и улучшения производительности проводится фильтрация данных. В первую очередь отбрасываются сегменты, которые больше не активны, либо были удалены из системы. Кроме того, данные сортируются по времени (`event_date`), что позволяет визуализировать процессы в хронологическом порядке. Такой подход обеспечивает целостную картину событий, отображая их последовательность и позволяя точно прослеживать изменения в системе слияния данных.

4.5 Визуализация данных

Для реализации визуализации был разработан скрипт на языке программирования JavaScript [7], основной задачей которого является отображение сегментов данных в виде прямоугольников и анимация процесса их слияния.

Для управления визуализацией предусмотрены кнопки "Старт"и "Сброс". По нажатию на "Старт"запускается процесс анимации с использованием метода `setTimeout`, что обеспечивает циклическое обновление элементов с задержкой, заданной в переменной `ANIMATION_DELAY`. На каждом этапе визуализации обновляется текстовое сообщение, которое отображает текущее действие — создание нового сегмента или слияние нескольких сегментов.

Анимация процесса слияния реализована следующим образом: при слиянии старые сегменты плавно исчезают (управляется задержкой `FADE_OUT_DELAY`), а новый сегмент появляется на их месте с эффектом масштабирования (используются задержки `SCALE_UP_DELAY` и `SCALE_DOWN_DELAY`). Цветовая индикация сегментов помогает различать тип события — новые сегменты отображаются синим цветом, а результат слияния — оранжевым. На каждом прямоугольнике также отображаются имя сегмента (`part_name`) и его размер в мегабайтах для упрощения анализа.

На рисунке 2 показан пример визуализации на одном из этапов процесса слияния данных.

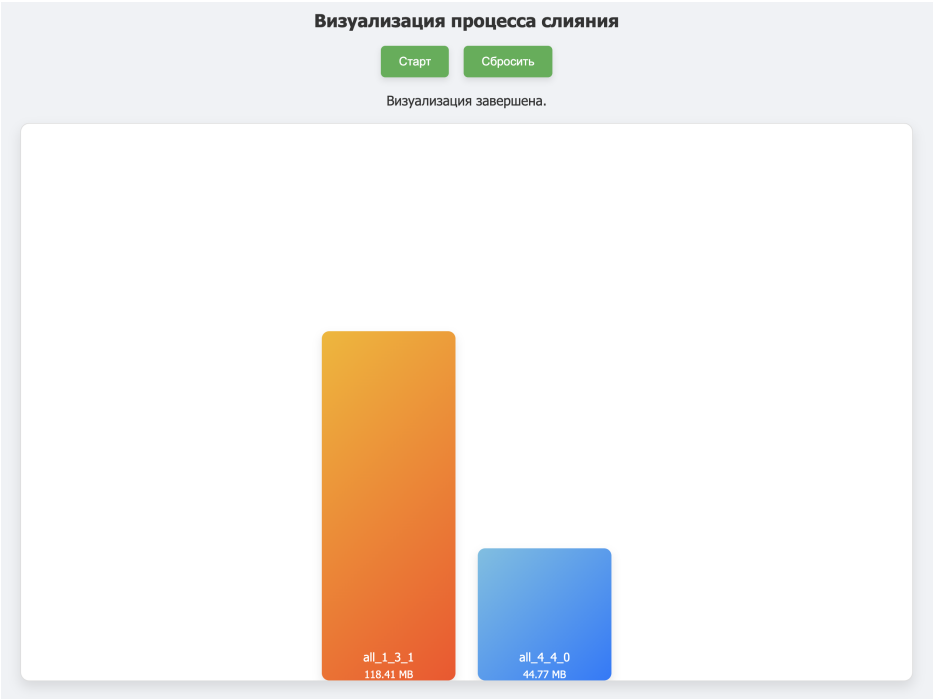

Рис. 2: Пример этапа визуализации процесса слияния данных.

В результате, полученное веб-приложение позволяет наглядно отслеживать текущие действия с данными на сервере.

5 Анализ эффективности алгоритма

5.1 Ключевые метрики

Алгоритм слияния данных в ClickHouse нацелен на оптимизацию двух основных метрик:

- **Write Amplification** — отношение объема данных, записанных на диск в процессе слияний, к объему исходных данных, вставленных в систему.
- **Среднее количество активных сегментов** — количество сегментов данных, доступных в системе в любой случайный момент времени. Чем больше сегментов, тем медленнее выполняются запросы `SELECT`, так как системе приходится обрабатывать большее количество частей данных.

Оптимизация этих двух метрик представляет собой сложную задачу, так как они противоречат друг другу. Для уменьшения количества сегментов необходимо чаще проводить операции слияния, что, в свою очередь, увеличивает *Write Amplification*. С другой стороны, снижение *Write Amplification* требует реже проводить слияния, что приводит к увеличению количества активных сегментов в системе. Поэтому в зависимости от того как работает алгоритм слияния, метрики будут находиться в балансе.

5.2 Реализация модуля расчета метрик

Для расчета данных метрик был разработан Python-скрипт(`get_metrics.py`), который обращается к ClickHouse и извлекает данные из таблицы `system.part_log`. Для взаимодействия с ClickHouse и обработки данных использовались библиотеки `requests`, `pandas` и `numpy`. Библиотека `requests` использовалась для выполнения HTTP-запросов к серверу ClickHouse, что позволяет отправлять запросы и получать данные в формате JSON. Библиотека `pandas` применялась для работы с табличными данными: она позволяет удобно выполнять фильтрацию, сортировку и агрегацию данных. `Numpy` использовалась для выполнения математических расчетов, таких как вычисление среднего значения и других статистических метрик. После получения данных проводится их предварительная обработка, включая фильтрацию событий типа `RemovePart` (удаленные сегменты) и сортировку данных по времени события (`event_time`).

Write Amplification вычисляется следующим образом:

- Сначала вычисляется общий объем данных, записанных в ходе всех операций слияния и вставок.
- Затем вычисляется объем данных, записанных только при вставке новых сегментов.
- *Write Amplification* определяется как отношение общего объема записанных данных к объему данных, записанных в ходе вставки новых сегментов, по следующей формуле:

$$write\_amplification = \frac{D_{total}}{D_{insert}}$$

где D_{total} — общий объем данных, записанных на диск, а D_{insert} — объем данных, вставленных через операцию `INSERT`.

Вторая метрика — **среднее количество активных сегментов** — рассчитывается по следующему алгоритму:

- Для каждого события типа `NewPart` (вставка нового сегмента) увеличивается счетчик активных сегментов.
- Для каждого события типа `MergeParts` (слияние сегментов) счетчик уменьшается на количество объединенных сегментов минус один.
- После каждой операции значение счетчика сохраняется, и по завершении всех операций рассчитывается среднее количество активных сегментов.

5.3 Результаты анализа

На основе данных из таблицы `system.part_log` были проведены расчеты для метрик *Write Amplification* и среднего количества сегментов. Примерный результат показывает, что коэффициент *Write Amplification* составляет в среднем 4-5, что свидетельствует о незначительных накладных расходах на операции слияния данных. Среднее количество активных сегментов, полученное в ходе анализа, составляет 80-90 сегментов.

Эти результаты показывают, что дальнейшая оптимизация алгоритма слияния возможна. Возможной стратегией оптимизации является вариативная настройка параметров алгоритма в зависимости от нагрузки на систему и объема данных.

6 Оптимизация алгоритма

6.1 Проектирование архитектуры симулятора

Целью данной части проекта является увеличение эффективности алгоритма слияния, то есть постараться минимизировать значения метрик. Для этого был разработан симулятор, который включает несколько ключевых модулей, реализованных в отдельных файлах. Все части программы-симулятора были разработаны на языке программирования Python 3.12.2. Расммотрим более подробно его архитектуру.

Файл `merge_selector.py` отвечает за реализацию класса `MergeSelector`, который непосредственно содержит логику выбора сегментов для слияния на основе ряда критериев, таких как размер сегментов, их возраст и общее количество сегментов. Для вычислений в этом файле активно используется библиотека `math`, которая обеспечивает математические операции, такие как вычисление логарифмов и интерполяций.

Файл `simulation.py` реализует основной цикл симуляции, включая добавление новых сегментов и запуск цикла слияния. Класс `Simulation`, описанный в этом файле, управляет всеми аспектами процесса симуляции, отслеживая такие параметры, как общее количество вставленных данных и уровень *Write Amplification*. Здесь также используются библиотеки `random` для генерации случайных размеров сегментов и `pandas` для хранения и обработки результатов экспериментов.

Файл `utils.py` содержит вспомогательные функции, такие как генерация комбинаций параметров (`generate_settings_combinations`) и работа с YAML-файлами настроек (`load_settings_from_yaml`). Этот модуль позволяет загружать настройки симуляции и сохранять результаты. Библиотека `itertools.product` используется для генерации всех возможных комбинаций параметров, что позволяет протестировать алгоритм на множестве конфигураций.

Файл `settings.yaml` содержит параметры настройки алгоритма, которые можно настраивать в ходе экспериментов, такие как значение коэффициента `base`, максимальное количество сегментов для слияния и другие важные настройки алгоритма. Эти параметры подгружаются при запуске симуляции, что позволяет быстро изменять конфигурации и проводить эксперименты с различными значениями.

6.2 Реализация симулятора

Ключевыми классами, реализованными в симуляторе, являются `Part`, `MergeSelector` и `Simulation`.

Класс `Part` представляет сегмент данных в системе. В конструкторе класса задается его размер, а также время создания, которое будет использоваться для расчета возраста сегмента. Это позволяет отслеживать, насколько долго сегмент находится в системе до момента слияния.

Класс `MergeSelector` отвечает за логику выбора сегментов для слияния. Он использует различные методы для оценки подходящих сегментов. Например, метод `score` рассчитывает оценку для слияния на основе размера сегментов и добавленной фиксированной стоимости. Метод `allow_merge` проверяет, можно ли выполнить слияние для выбранной группы сегментов, основываясь на их возрасте, размере и количестве сегментов в системе. Также в классе реализован метод `interpolate_linear`, который используется для линейной интерполяции значений параметров. Наконец, основной метод `select_parts_to_merge` отвечает за выбор наилучших сегментов для слияния. Он сортирует сегменты по возрасту и перебирает возможные группы сегментов для слияния, выбирая оптимальные комбинации на основе расчетных показателей.

Класс `Simulation` управляет процессом симуляции. В нем содержатся методы для вставки новых сегментов (`insert_part`), проведения цикла слияния (`run_merge_cycle`) и запуска симуляции (`simulate`). В процессе симуляции новые сегменты случайного размера добавляются в систему, после чего запускается цикл слияния, где алгоритм `MergeSelector` выбирает группы сегментов для объединения. На каждом шаге симуляции рассчитываются ключевые метрики, такие как *Write Amplification* и среднее количество активных сегментов.

6.3 Описание методики симуляции

Алгоритм `SimpleMergeSelector` содержит 18 параметров, которые могут быть изменены для достижения оптимальных значений ключевых метрик системы. Для каждого из этих параметров было выбрано три возможных значения в рамках определенного диапазона. Чтобы избежать избыточной нагрузки на систему и сократить время экспериментов, в каждом эксперименте изменялись значения только половины параметров (девяти) одновременно. После выполнения симуляций с первой группой параметров, улучшенные значения были зафиксированы, и в следующем цикле экспериментов производились изменения второй группы параметров, с учетом оптимизированных значений первой группы. Этот итеративный процесс позволил пошагово оптимизировать все 18 параметров без значительного увеличения вычислительной нагрузки.

6.4 Результаты оптимизации

Общее количество комбинаций параметров на первом этапе составило 19,683. После выполнения симуляции первой группы параметров, 4438 комбинаций показали какие-то улучшение по сравнению с базовой конфигурацией алгоритма. Эти комбинации использовались для проведения второго этапа симуляций, где тестировались оставшиеся девять параметров. Таким образом на втором этапе были протестированы 79884 возможных комбинаций параметров. Из результатов второго этапа 2180 комбинаций показали значительное улучшение по сравнению с предыдущими результатами. В заключении второго этапа оптимизации были сохранены и зафиксированы параметры с лучшими результатами по метрикам.

Каждая симуляция включала 500 циклов вставки и слияния сегментов данных, и на выполнение одного эксперимента требовалось в среднем 60-70 секунд на локальной машине с процессором 3.2 ГГц.

Таблица 1: Сравнение значений метрик с базовыми и улучшенными параметрами.

Метрика	Базовые настройки	Оптимизированные настройки
Write Amplification	4.530772	3.995551
Среднее количество сегментов	72	70

Из Таблицы 1 видно, что предложенные изменения в конфигурации алгоритма `SimpleMergeSelector` позволили улучшить ключевые метрики производительности системы: *Write Amplification* уменьшилось на 11.8% и среднее количество сегментов в системе также уменьшилось на 2.7%.

Таким образом, предложенные изменения в конфигурации алгоритма позволили улучшить ключевые метрики производительности системы ClickHouse.

7 Заключение

В рамках проекта была проведена значительная работа, включающая следующие достижения:

- Разработано веб-приложение для визуализации процессов слияния данных в системе ClickHouse.
- Проведен подробный анализ эффективности алгоритма слияния данных на основе ключевых метрик, таких как write amplification и количество активных сегментов.
- Выполнены экспериментальные исследования по оптимизации алгоритма слияния, предложены рекомендации по изменению параметров с целью повышения производительности системы.

В дальнейшем планируется проверить новые параметры, полученные из оптимизации, на реальных данных production-серверов, что позволит оценить их эффективность в условиях, максимально приближенных к промышленной эксплуатации. Также необходимо провести углубленный анализ не только параметров алгоритма, но и его общей структуры. Существующий алгоритм может не являться наилучшим решением для текущих задач, поэтому важно рассмотреть возможность разработки альтернативных подходов к процессам слияния данных, что может открыть дополнительные пути для улучшения производительности системы.

Список литературы

[1] *Документация по движку MergeTree*. URL: https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/mergetree.

[2] *Документация по работе с таблицей system.part_log*. URL: https://clickhouse.com/docs/en/operations/system-tables/part_log.

[3] *Исходный код алгоритма SimpleMergeSelector*. URL: https://github.com/ClickHouse/ClickHouse/blob/master/src/Storages/MergeTree/SimpleMergeSelector.h.

[4] *Официальная документация ClickHouse*. URL: https://clickhouse.com/docs.

[5] *Справочник CSS*. URL: https://htmlbook.ru/css.

[6] *Справочник HTML*. URL: https://htmlbook.ru/html.

[7] *Справочник JavaScript*. URL: https://javascript.ru/manual.