Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

КУРСОВАЯ РАБОТА

Программный проект на тему

"Полиморфные куски данных в таблицах типа MergeTree в ClickHouse"

Выполнил студент группы 176, 3 курса,
Попов Антон Дмитриевич

Руководитель КР:
Руководитель группы разработки ClickHouse
Миловидов Алексей Николаевич

Москва 2020

Содержание

1 Аннотация

ClickHouse - это распределенная колоночная СУБД для аналитических запросов. В ClickHouse существует несколько движков таблиц, самым функциональным и производительным из которых является MergeTree. Но есть одна особенность, с которой сталкиваются все пользователи, начинающие работать с ClickHouse: данный движок плохо поддерживает частые INSERT запросы с маленьким количеством строк из-за поколоночного хранения данных. В этой работе рассматриваются способы решения данной проблемы с помощью реализации новых форматов хранения данных.

ClickHouse is a distributed column-oriented DBMS for analytical queries. There are several table engines in ClickHouse, but the most functional and performant is MergeTree. But is has one drawback, that every new user face with: this engine doesn't support frequent INSERT queries with small number of rows, because of its column-oriented data storing. This work resolves this problem by implementing new formats of data storing in MergeTree tables.

2 Введение

2.1 Описание архитектуры ClickHouse

ClickHouse [1] - это распределенная колоночная СУБД для аналитических запросов. Ее исходный код написан на C++ и находится в открытом доступе.

В ClickHouse существует несколько движков таблиц, которыми определяется как хранятся данные и какие запросы поддерживает таблица. Самым функциональным из них является MergeTree [2]. Он поддерживает индексы, партиционирование, репликацию (ReplicatedMergeTree [3]) и другие возможности. Данные в MergeTree хранятся в так называемых **кусках, отсортированных по первичному ключу**. При каждой вставке создается новый кусок и далее в фоне происходят их **слияния** для более оптимального хранения данных. В кусках каждый столбец хранится в отдельном файле (или нескольких файлах для сложных типов), при этом он сжимается по блокам. Кусок разделяется на **гранулы**, которые являются минимальными неделимыми единицами для чтения с диска. Размер гранулы задается настройками таблицы. Затем первая строка каждой гранулы помечается соответствующим значением первичного ключа и засечкой. **Засечка** хранит смещения в файле и разжатом блоке, по которым можно прочитать гранулу, начиная с первой строки, а также количество строк в грануле. Засечки пишутся для каждого файла с данными. Таким образом получается разреженный индекс, который хранится в оперативной памяти. С помощью него при анализе запроса можно выбирать гранулы, из которых нужно читать данные.

2.2 Постановка задачи

Хранение столбцов в отдельных файлах необходимо для производительности движка, а именно:

- Для быстрых манипуляций со столбцами (Запросы ALTER DROP/MODIFY COLUMN).

- Для более оптимального хранения данных. Данные в одном столбце могут иметь заранее известную нам структуру (например, являться монотонно возрастающей последовательностью), и в таком случае к ним можно применить специализированные алгоритмы сжатия, сэкономив место на диске.

- Для более оптимального чтения столбца. При чтении одного столбца в таком формате будет гораздо меньше seek-ов по файлу или не будет вовсе. Как известно случайные чтения с HDD работают намного дольше, чем последовательные.

Но у такого подхода есть одна проблема: вставки маленького количества строк в таблицы с большим количеством столбцов является неэффективным, поскольку требуют создания большого количества файлов. Это является одной из главных особенностей ClickHouse, с которой сталкиваются пользователи в начале работы с базой данных. Если вставок будет слишком много, новые куски не будут успевать сливаться в большие, и их количество начнет расти. После того как количество кусков преодолеет некоторый порог пользователь при вставке получит ошибку "Too many parts", потому что дальнейшая работа таблицы становится неэффективной.

2.3 Цель и актуальность работы

Целью работы является создание новых форматов хранения кусков данных для таблиц семейства MergeTree, благодаря которым улучшится производительность мелких вставок.

Работа является актуальной, потому что решает одну из важных проблем ClickHouse, с которой сталкивается множество пользователей.

3 Существующие решения

3.1 Введение

На данный момент существуют два подхода к решению проблемы частых вставок: использование функциональности, встроенной в СУБД - движки таблиц Buffer [4] и Kafka [5], и различные сторонние прокси-сервера для ClickHouse. Из множества сторонних решений рассмотрим две наиболее популярные open-source библиотеки для буферизации вставок в ClickHouse: KittenHouse [6] и ClickHouse-Bulk [7].

Далее рассматриваются достоинства и недостатки каждого из упомянутых решений.

3.2 Движок таблиц Buffer

При создании таблицы типа Buffer указывается имя целевой таблицы, пороги по количество строк, байт и времени для хранения данных в буфере. При вставке в таблицу данные сначала буферизуются в оперативной памяти, а затем вставляются в целевую таблицу. При чтении из таблицы возвращаются данные, находящиеся в буфере, а также данные из целевой таблицы.

Достоинства:

- Решение является простым в настройке и не требует установки и администрирования дополнительного ПО.
- В простых случаях позволяет решить проблему частых вставок.

Недостатки:

- При аварийном завершении сервера данные, находящиеся в буфере, теряются без возможности восстановления. Ограниченная поддержка ALTER запросов (запросов на изменение структуры таблицы). Для большинства видов ALTER-ов необходимо удалить Buffer-таблицу, выпол-

нить модификацию над целевой таблицей и затем заново создать Buffer-таблицу.

- Отсутствует индекс, что может негативно сказаться на производительности чтения в случае накопления большого объема данных в буфере.

- Вставка частично блокирует чтение. При интенсивном потоке INSERT-ов, SELECT запросы могут замедлится. Если же читать напрямую из целевой, а не Buffer таблицы, то теряется важное свойство: пользователь некоторое время не сможет прочитать данные, которые он только что вставил.

- Данные в Buffer таблице переупорядочиваются. Из-за этого теряется возможность дедупликации данных в реплицируемых таблицах. Механизм дедупликауции не дает вставить блок, который полностью идентичен какому-то ранее вставленному блоку. Это является важным методом борьбы с дупликацией данных, иногда возникающей в случае ошибок сети и переотправки запросов клиентским приложением.

3.3 Apache Kafka

Apache Kafka является одним из самых популярных среди пользователей способов доставки данных в ClickHouse. Клиентское приложение пишет данные в Kafka, при этом подписчиком является еще одно приложение, задачей которого является группировка данных и перекладывание их в ClickHouse, или же таблица с движком Kafka непосредственно в самом ClickHouse, которая сама группирует и перекладывает данные в целевую таблицу.

Достоинства:

- Apache Kafka - является надежной, популярной технологией, по которой имеется большое количество обучающих материалов и ответов на вопросы.

- Можно организовать доставку данных в другие СУБД и хранилища, помимо ClickHouse, универсальным способом.

  Недостатки:

- Необходимость использования еще одной технологии для того, чтобы воспользоваться ClickHouse. Это добавляет дополнительную работу для администраторов, DevOps-инженеров и разработчиков. Также, чтобы развернуть Apache Kafka необходимо как минимум 2-3 сервера для репликации, что приводит в итоге к дополнительным расходам компании.

- К движку таблиц Kafka у пользователей возникали вопросы к корректности его работы [8]. Были случаи, когда нарушалась гарантия exactly-once, или данные вовсе терялись. Также были вопросы к производительности движка таблиц Kafka.

3.4 Сторонние технологии

У представленных ниже решений, а также у остальных похожих проприетарных решений есть схожие недостатки:

- Они могут не поддерживать всех возможностей ClickHouse, а также реализовывать новые возможности с запозданием, или не реализовывать вовсе. Например, поддержку новых типов данных.

- Они добавляют сложность в разработку и поддержку приложения, использующего ClickHouse, а также могут являться точками отказа.

3.4.1 KittenHouse

Прокси-сервер для INSERT и SELECT запросов в ClickHouse, разработанный в компании "ВКонтакте". Позволяет балансировать нагрузку, а также буферизовать данные для вставок.

Достоинства:

- Буферизация в памяти и на диске. Можно восстановить данные, которые не удалось вставить в ClickHouse.
- Поддерживаются форматы Values и более производительный RowBinary.

Недостатки:

- Нельзя настроить пороги для группировки данных при буферизации.
- Не поддерживает остальные популярные форматы (TabSeparated, CSV и т.д.), которые доступны в ClickHouse.
- Не развивается с 2018 года.

3.4.2 ClickHouse-Bulk

Имеет единственную функцию - буферизация данных для вставки в ClickHouse.
Достоинства:

- Очень прост в использовании
- Гибкая настройка. Позволяет сконфигурировать пороги по размеру и времени для группировки данных и последующей вставки в ClickHouse.
- Данные, которые не удалось вставить сохраняются на диск, и их можно восстановить.

Недостатки:

- Поддерживаются только 2 формата: Values и TabSeparated.

4 Описание работы

Предполагается реализовать два новых формата:

- Компактный. Все столбцы хранятся в одном файле. Засечки также записываются в один файл.
- Формат с буферизацией данных в оперативной памяти со сбросом на диск. Для него также будет реализован опциональный Write-Ahead-Log.

Текущий формат хранения назовем “широким”.

Из одного формата в другой куски будут переходить во время слияний при достижении порогов по размеру, выраженному в количестве строк или байт. При этом из формата в оперативной памяти слияние может происходить только в формат, хранящий данные на диске. Это должно снизить сложность мелких вставок за счет снижения количества операций с файловой системой и не изменить производительность всей системы. Особенно заметно улучшение будет на медленных файловых системах. Например, в ClickHouse планируется реализовать возможность создавать таблицы семейства MergeTree поверх облачного объектного хранилища Amazon S3 [9]. Обращение к нему стоит в разы дороже, чем к обычной файловой системе.

Несмотря на то, что чтение из компактных кусков немного замедлится (причины этого описаны в 2.2), это не будет сильно влиять на скорость SELECT запросов, т.к. предполагается хранить в компактном формате только небольшие куски (размером примерно до 10МБ). Чтение из них будет вносить незначительный вклад во время выполнения запроса на фоне чтения из оставшейся части таблицы. ALTER запросы будут выполняться неэффективно, т.к. придется переписывать весь кусок, а не один столбец, но за счет маленького размера компактных кусков это не критично. Также требования на скорость выполнения ALTER запросов не такие строгие как к SELECT запросам. Куски в оперативной памяти на производительность запросов повлияют только в положительную сторону.

5 Реализация работы

5.1 Подготовительные работы

Кратко опишем процесс обработки запроса. Строится конвейер выполнения (ациклический граф), который состоит из процессоров, каждый из которых умеет выполнять какое-то полезное действие (сортировка, агрегация и т.д.). Данные по конвейеру между процессорами передаются в блоках. Блок представляет собой подмножество столбцов, в которых хранятся последовательности строк таблицы. Если представить таблицу как двумерную матрицу, то блок будет являться ее подматрицей. Конвейер SELECT запроса начинается с процессора (Source), который умеет читать блок из таблицы и передавать его дальше. Конвейер INSERT запроса заканчивается процессором (Sink), принимающим блок, и записывающим его в таблицу. При этом блок может содержать несколько гранул.

Для того чтобы реализовать новый вид кусков в MergeTree, необходимо реализовать классы (Reader и Writer), которые умеют читать блоки из таблицы и записывать блоки в таблицу в нужном формате, а также написать необходимую логику, которая вызывается при изменении столбцов а ALTER-ах и мутациях (аналог запросов UPDATE и DELETE в традиционных реляционных СУБД [10]). Source и Sink процессоры одинаковые для всей таблицы типа MergeTree, поэтому при выполнения запроса, обрабатывая очередной блок, они вызывают нужные реализации Reader-а и Writer-а в зависимости от типа куска, с которым сейчас работают.

Сначала был проведен рефакторинг кода, выделены интерфейсы для чтения, записи куска и хранения самой его структуры. Рефакторинг был необходим, поскольку ранее не предполагалось, что куски могут иметь разный формат. Код для широких кусков уже есть. Далее были написаны реализации Reader-а и Writer-а для компактного формата и кусков в памяти.

5.2 Компактный формат

В компактном формате каждая гранула записывается в виде набора подряд сериализованных столбцов. Гранулы записываются подряд в один файл. Индекс считается и записывается также, как и в широком формате. Одна засечка представляет собой количество строк в грануле и массив смещений для каждого столбца, также как в широком формате. Всего в куске получается константное число файлов: данные, засечки, индекс и еще несколько служебных файлов, необходимых для работы таблицы. Тогда как в широком формате их число пропорционально числу столбцов в таблице.

5.3 Формат в оперативной памяти

Кусок хранит сортированный блок, находящийся в оперативной памяти. Этот же блок является единственной гранулой такого куска. Соответственно в индекс записаны первая и последние строки блока. Для кусков в памяти реализована опциональная поддержка (включена по умолчанию) Write-Ahead-Log (WAL). В WAL в базах данных обычно записывается намерение произвести какую-то операцию с таблицей, чтобы в случае аварийного завершения, можно было ее продолжить или отменить уже выполненную часть. В нашем случае в WAL будут писаться все приходящие при INSERT-е, а также при полученные при репликации, блоки в Native формате ClickHouse. Native формат по сути представляет собой бинарно сериализованные блоки (почти такой же, как в компактных кусках). WAL позволит легко восстанавливать вставленные данные в случае аварийного завершения сервера, но его также можно будет отключить, если пользователю нужна еще большая производительность, и он готов потерять свойство устойчивости базы данных. При этом стоит отметить, что решение с WAL будет оптимальней, чем запись куска сразу в компактном формате, поскольку создается только один файл с WAL вместо создания директории и нескольких файлов на каждый кусок.

6 Заключение

6.1 Итог работы

Были реализованы два новых формата хранения данных в кусках таблиц семейства MergeTree. Были добавлены настройки таблиц min_rows_for_wide_part, min_bytes_for_wide_part, min_bytes_for_compact_part, min_rows_for_compact_part, с помощью которых можно регулировать формат данных в зависимости от объема и таким образом подстраивать таблицу под интенсивность вставок в нее. У пользователей во многих случаях появится возможность отказаться от дополнительнительных приложений для буферизации данных, а также от создания дополнительных промежуточных таблиц в ClickHouse.

Пулл-реквесты в GitHub репозиторий ClickHouse с реализацией работы:
https://github.com/ClickHouse/ClickHouse/pull/8290
https://github.com/ClickHouse/ClickHouse/pull/10697

6.2 Сравнение производительности

Сравнение было произведено с помощью утилиты clickhouse-benchmark. Использовалась таблица hits из тестового датасета Яндекс Метрики, которая содержит 133 столбца различных типов. Запускался запрос типа:

INSERT INTO test.hits_mem(CounterID) SELECT rand() FROM numbers(100)

Он генерирует 100 случайных значений, записывает их в столбец CounterID и заполняет остальные столбцы значениями по умолчанию. На момент написания работы столбцы, полностью состоящие из значений по умолчанию, материализовывались и записывались на диск. Поэтому производительность такого запроса точно такая же, как если бы мы заполняли все столбцы случайными значениями. Тесты проводились на виртуальной машине со следующими характеристиками: 32 vCPU 2.10GHz, 46GB RAM, 500GB HDD. В таблице 6.1 представлены результаты сравнения QPS (Query Per Second) в

зависимости от формата данных и числа строк в одном INSERT запросе.

Таблица 6.1: Сравнение QPS в зависимости от формата

	1 строка, 1 поток	100, 1	1000, 1	1, 4	100, 4	1000, 4
Широкий формат	118	92	80	13	11	16
Компактный формат	432	391	258	1233	1151	773
Формат в памяти	423	464	242	1126	1352	705
Buffer таблица	688	531	334	2391	1718	563

Во время записи в 4 потока а таблицу с широким форматом кусков утилизация диска достигала 100%. Этим объясняется такие низкие значения QPS. Можно сказать, что данный формат не поддерживает такую интенсивную запись. Производительность компактного формата и формата в памяти оказались практически одинаковыми и в несколько раз превышают производительность широкого формата. Потеря производительности по сравнению с Buffer таблицей не такая большая. Поэтому, учитывая все ее недостатки, описанные в 3.2, новые форматы вполне могут заменить Buffer таблицы и стать решением проблемы частых вставок.

Список источников

1. Официальная документация
2. Таблицы MergeTree
3. Таблицы ReplicatedMergeTree
4. Таблицы Buffer
5. Таблицы Kafka
6. Библиотека KittenHouse
7. Библиотека ClickHouse-Bulk
8. Список issues на GitHub о таблицах Kafka

9. Amazon S3

10. Мутации в ClickHouse