Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Основная образовательная программа
«Прикладная математика и информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Программный проект на тему
Минимальная поддержка транзакций для множества вставок/чтений

Выполнил студент группы БПМИ-165, 4 курса,

Кузнецов Максим Анатольевич

Руководитель ВКР:

Руководитель группы разработки ClickHouse

Миловидов Алексей Николаевич

Москва 2020

1. Аннотации **3**
1.1. На русском 3
1.2. На английском 3
2. Введение **4**
2.1. Предмет исследования 4
2.2. Цели и задачи 6
2.3. Актуальность и значимость 7
2.4. Ключевые термины 7
3. Обзор литературы **8**
3.1. Различные способы реализации транзакций 8
3.2. Существующие системы с транзакциями 10
3.2.1. Exasol 10
3.2.2. Postgre 10
3.2.3. MySQL 11
3.2.4. MonetDB 12
4. Описание разработанного алгоритма **13**
4.1. Архитектура движков семейства MergeTree в ClickHouse 13
4.2. Алгоритм транзакций 18
4.3. Специфика алгоритма в случае реплицированной таблицы 23
5. Заключение **25**
6. Список литературы **26**

1. Аннотации

1.1. На русском

ClickHouse - система управления базами данных (СУБД), разработанная для онлайн обработки аналитических запросов. Система была разработана и поддерживается IT компанией Яндекс. Изначально ClickHouse был разработан для Яндекс.Метрики - одной из самых популярных систем для веб аналитики в мире. Сейчас системой ClickHouse пользуются множество компаний из стран со всего мира. ClickHouse была разработана, как система, способная хранить огромное количество данных, при этом максимально эффективно выполнять запросы на их обработку - сейчас эта система способна хранить миллиарды записей и обрабатывать десятки тысяч в доли секунды. В данной работе будет предложен алгоритм, который добавляет поддержку транзакций в движки хранения таблиц семейства MergeTree в ClickHouse. Основным результатом данной работы является корректный алгоритм транзакций, удовлетворяющий свойствам ACID, как для нереплицированных таблиц, так и для реплицированных.

1.2. На английском

ClickHouse is a database management system (DBMS) designed for online processing of analytical queries. The system was developed and maintained by IT company Yandex. ClickHouse was originally developed for Yandex.Metrica - one of the most popular web analytics systems in the world. Now the ClickHouse is used by many companies from countries around the world. ClickHouse was developed as a system that can store a huge amount of data and perform requests for processing them as efficiently as possible - now this system is able to store billions of records and process tens of thousands in a split second. In this paper, we will propose an algorithm of transactions for the MergeTree family of table storage engines in ClickHouse. The main result of this work is a correct transaction algorithm that satisfies the properties of ACID, both for unreplicated tables and for replicated ones.

Ключевые слова: *ClickHouse; СУБД; дерево слияний; OLAP; транзакции*

2. Введение

2.1. Предмет исследования

ClickHouse - колоночная система управления базами данных (СУБД) с открытым исходным кодом, разработанная компанией Яндекс, ориентированная на быстрое выполнение аналитических запросов. Первый прототип системы появился в 2009 году, а с 2016 года проект является open-source под лицензией Apache-2.0. ClickHouse был разработан для решения задач веб-аналитики для Яндекс.Метрики. В тот момент для решения задач аналитики использовали заранее агрегированные данные. Этот подход позволял значительно уменьшить количество хранимых данных, а также ускорить время выполнения запросов. Однако, такой метод имел ряд недостатков. Например, невозможно было построить произвольный отчет по причине того, что все возможные агрегации должны быть заранее фиксированы. Также необходимость агрегации по разным ключам приводил к увеличению количества хранимой информации. Альтернативным способом было хранение всех данных и обработка их с помощью системы управления баз данных, способной обрабатывать петабайты данных с очень высокой эффективностью в реальном времени. Так как на тот момент не существовало подходящих СУБД, то в компании Яндекс начали разработку своей СУБД [1].

Основным из преимуществ ClickHouse по сравнению с другими существующими СУБД является колоночная архитектура. Это позволяет ClickHouse эффективно выполнять аналитические запросы в реальном времени. Данные хранятся колонками и при запросе к таблице обрабатываются только те колонки, которые присутствуют в запросе. ClickHouse хранит данные в сжатом виде, использую такие алгоритмы сжатия, как LZ4 и ZSTD. Также имеется возможность хранить данные по первичному ключу - данные хранятся

сортировано относительного первичного ключа, чтобы запросы по малому диапазону первичных ключей выполнялись быстрее, не требую чтения всей таблицы целиком. Помимо этого, в ClickHouse имеется возможность репликации данных для защиты от потерь.

ClickHouse спроектирован исходя из того, что это должна быть система, способная обрабатывать огромные объемы данных максимально эффективно. По этой причине для имплементации системы был использован язык C++. Также, из-за колоночной архитектуры ClickHouse сильно экономит потребление CPU. Используются наиболее эффективные решения для базовых задач, возникающих в системе - алгоритмы сжатия и разжатия данных, алогритмы сортировок, алгоритмы поиска строк в подстроке, алгоритм вычисления экспоненты и многое другое. По всем этим причинам ClickHouse зарекомендовал себя, как одна из эффективнейших систем, существующих на рынке, что подтверждают бенчмарки (от английского слово benchmark - тест на производительность системы), проведенные как командой, так и сторонними разработчиками. ClickHouse в десятки или сотни раз быстрее других аналогичных систем [2].

В ClickHouse имеется некоторое количество движков данных - способов внутреннего хранения и обработки таблиц. Каждый из движков имеет свои преимущества и недостатки, однако самыми используемыми и эффективными движками являются движки семейства MergeTree. Данное семейство движков позволяет хранить данные сотритовано по первичному ключу, эффективно выполнять запросы чтения и вставки новых данных, а также имеет возможность реплицировать данные не несколько серверов для большей отказоустойчивости [3].

Для аналитики данных на практике не редко бывает необходимым строить сложные отчеты, для которых нужно несколько раз прочитать данные из одной или даже нескольких различных таблиц. Но может возникнуть ситуация, когда

между различными чтениями успела произойти вставка новых данных. В таком случае результаты чтений могут получиться несогласованными, что может привести к некорректному аналитическому отчету.

Для избавления от данной проблемы было решено добавить в движки семейства MergeTree в ClickHouse минимальную поддержку транзакций. В данном случае транзакции должны выполнять две функции - возможность одновременной и атомарной вставки данных в несколько таблиц и возможность одновременно выполнить несколько чтений из таблиц без влияния на чтения новых вставок в таблицы. Так как в нынешней архитектуре ClickHouse в одну таблицу может одновременно выполняться несколько вставок и несколько чтений, одним из требованиев к транзакциям было возможность одновремееного выполняния с потенциальными вставками и чтениями из таблицы.

2.2. Цели и задачи

На начальном этапе выполнения задачи были примерно таковыми: изучить устройство ClickHouse, придумать алгоритм транзакций в рамках существующей архитектуры и реализовать его. По ходу выполнения список задач получился следующим:

- Изучить устройство ClickHouse
- Реализовать метки для единиц хранения данных в таблицах в случае одного сервера
- Реализовать поддержку работы с метками единиц хранения данных для запросов SELECT и INSERT
- Реализовать поддержку работы с метками при слияниях данных
- Реализовать поддержку работы с метками для мутаций (ALTER операции)
- Реализовать работу с метками в случае нескольких серверов, в том числе

для реплицированных таблиц

2.3. Актуальность и значимость

ClickHouse одна из самых популярных СУБД для OLAP систем. Им пользуются большое количество компаний. Среди них Яндекс, Deutsche Bank, S7 Airlines, МКБ. Очень много аналитических отчетов строится с использованием ClickHouse и иногда эти отчеты получаются неверными. Если ошибка заметна, то приходиться делать повторный запрос, иначе же это может привести к неверным выводам и, соответственно, неверным решениям внутри компании. Имплементация транзакций позволит избавиться от данной проблемы.

2.4. Ключевые термины

1. СУБД - система управления базами данных
2. Движок хранения данных - алгоритм, управляющий хранением данных таблицы, вставкой новых данных или чтением данных
3. MergeTree - название движка в ClickHouse
4. OLAP (online analytical processing) - интерактивная аналитическая обработка
5. WAL (write-ahead log) - журналирование событий перед их исполнением
6. Read-write блокировка - блокировка, которую можно либо заблокировать одним клиентов в режиме “write”, либо множеством в режиме “read”
7. OCC (Optimistic Concurrency Control) - оптимистичное управление параллельным доступом
8. MVCC (multiversioning concurrency control) - управление параллельным доступом посредством многоверсионности
9. ACID - набор требований к транзакциям в СУБД

3. Обзор литературы

3.1. Различные способы реализации транзакций

Для того, чтобы начать говорить о транзакциях, надо понять, что же такое транзакция. СУБД имеют набор некоторых операций, которые можно проводить с базой данных. Этого могут быть операции записи, чтения или изменения данных. Иногда становиться необходимым провести несколько операций *одновременно* (дальше мы раскроем истинный смысл это слова для нашего случая), чтобы система не начала нарушать некоторые гарантии.

Рассмотрим стандартный пример, который часто вспоминают, говоря о транзакциях: пусть у нас есть два клиента A и B с банковскими счетами и клиент A хочет перевести клиенту B некоторую сумму денег amount. В таком случае мы обязаны одновременно уменьшить сумму счета клиента A на значение amount и увеличить сумму счета клиента B на значение amount. Эти две операции обязаны произойти в момент, будто это была одна операция, иначе суммарное количество денег на счетах не будет постоянным.

В 1983 году Андреас Рейтер и Тео Хардер сформулировали правило ACID, основываясь на свое работе [4]. При реализации транзакционной модели в СУБД требуют, чтобы она удовлетворяла ACID правилу, чтобы система работала предсказуемо и надежно. ACID акроним от слов atomicity, consistency, isolation и durability.

- **A**tomicity - атомарность

Атомарность гарантирует, что ни одна транзакция не может быть выполнена частично - должны быть выполнены либо все операции, либо ни одна из них. В примере выше это означает, что транзакция не может привести к ситуации, когда только с счета клиента A были сняты деньги или только на счет клиента B деньги пришли.

- **C**onsistency - согласованность

Согласованность гарантирует, что любая транзакция переводит систему из валидного состояния в валидное. В нашем примере это означает, что суммарное кол-во денег на всех счетах должно оставаться постоянным.

- **I**solation - изолированность

Транзакции часто выполняются конкурентно, то есть может существовать несколько транзакций, работающих с одними и теми же данными. Изолированность гарантирует, что параллельные транзакции не будут оказывать влияния на друг друга.

- **D**urability - стойкость

Стойкость гарантирует, что как только операция может считаться выполненной, результаты транзакции будут сохранены в системе.

Есть две основных техники реализации транзакций [5].

1. Упреждающая журнализация (write-ahead logging)

В данном подходе все желаемые транзакции перед выполнением записываются в специальный лог (журнал) операций. Именно он определяет порядок выполнения транзакций.

2. Теневой механизм (shadow paging)

В данном подходе создаются копии данных при их изменении, чтобы читающие операции все еще могли читать старые данные, пока пишущие операции их меняют.

Для достижения гарантий ACID используются несколько алгоритмов. Самый популярных из них - двухфазное блокирование (two-phase locking). В данном алгоритме все данные, с которыми хочет работать транзакция, предварительно должны быть заблокированы, чтобы другие транзакции не могли работать с данными - это называется первой фазой. На второй фазе блокировки освобождаются [6].

Еще одна из возможных реализаций - MVCC (multiversion concurrency

contro). Этот алгоритм дает каждой транзакции версию - некоторое монотонно возрастающее число. Чаще всего для этого используют логические или системные часы. Если данные несколько транзакций одновременно работают с некоторыми данными и хотя бы одна из них их модифицирует, то приоритет дают транзакции с меньшим номером, а остальные транзакции либо отменяются, либо перезапускаются. Благодаря такому подходу может быть достигнута изоляция снапшот (snapshot isolation) - каждая операция будет видеть согласованный снапшот всей системы. При этом разные операции могут видеть разные снапшоты. Данную гарантию предоставляют СУБД Oracle, MySQL, PostgreSQL, SQL Anywhere, MongoDB и Microsoft SQL Server [7].

3.2. Существующие системы с транзакциями

3.2.1. Exasol

В данной системе реализован подход MVCC. При этом упоминается, что блокировка данных используется на уровне таблиц, то есть разные строки одной таблицы не могут быть одновременно изменяться в нескольких транзакциях [10]. Для этого используется read-write блокировка. Также блокировка таблиц происходит в порядки обращения к ним в транзакции, что может привести к конфликту и отмене транзакций [11].

3.2.2. Postgre

В данной системе реализован подход упреждающего журналирования для выполнения условия стойкости. Работа с данными происходит посредством MVCC [9].

В системе есть некоторые свойства запроса (запросом может быть как транзакция, так и отдельная операция).

1. Каждый запрос к системе имеет свою метку, основываясь на которой

запрос понимает, с какими данными можно работать, а с какими нельзя - xid.

2. Каждая строка в данных имеет 2 метки.

t_xmin - метка запроса, при которой данные появились в системе. Эти данные могут читать только те запросы, у которых xid >= t_xmin.

t_xmax - метка запроса, при которой данные удалились из системы. Эти данные могут читать только те запросы, у которых xid < t_xmax.

3. Каждый запрос дополнительно имеет массив меток запросов, которые не успели завершиться на момент начала нашего запроса xip и для оптимизации минимальную метку из xip - xmin.

Запрос может читать только те данные, у которых t_xmin не находится в массиве xip. Для оптимизации условие t_xmin < xmin означает, что t_xmin гарантированно не находится в xip.

3.2.3. MySQL

В MySQL реализованы множество движков хранения данных, рассмотрим один из них - InnoDB. Реализация сочетает в себе два алгоритма - двухфазную блокировку и MVCC [8].

Двухфазная блокировка работает на уровне строк в таблице. Так, чтобы работать с данными, транзакция обязана взять блокировки на нужные строки. При этом используется read-write блокировка. Если операция хочет совершить чтение, то одна берет блокировку на чтения, если запись - на запись. При этом гарантируется, что одновременно может существовать либо только одна блокировка на запись, либо множество блокировок на чтение одной и той же строки.

Если же вся операция только читает данные, ничего не меняя, то она обходит алгоритм двухфазной блокировки и просто читает необходимые данные, основываясь на версиях, которые существуют у каждой строки, тем

самым проверяя, что запись была создана до операции и не была удалена до этой операции.

3.2.4. MonetDB

В MonetDB используется OCC реализованную через изоляцию снапшота с помощью MVCC. При этом в системе не достигается гарантия линеаризуемости. Также в системе используется WAL для гарантий атомарности и стойкости [12].

В данной системе каждая транзакция получает свой собственный снапшот, с которым работает в дальнейшем. Каждая транзакция полностью выполняет свою работу, после чего проверяется, не было ли у этой транзакции конфликтов с другими транзакциями (например, модификация одних и тех же данных). Если конфликтов не обнаружено, то транзакция является успешно выполненной.

4. Описание разработанного алгоритма

4.1. Архитектура движков семейства MergeTree в ClickHouse

Прежде, чем приступить к описанию алгоритма, необходимо понять, как работает существующая версия ClickHouse и в частности движки семейства MergeTree.

Стоит упомянуть, что ClickHouse оптимизирован для случая, когда необходимо хранить большие объемы данных и основным запросом к ним является чтение. По этой причине единственная операция, которая может изменить таблицу в ClickHouse - INSERT, добавляющая новые строки в таблицу. Хоть и есть способы изменять таблицу, они являются крайне неэффективными и не рекомендуются к использованию.

Таким образом основными операциями являются SQL запросы SELECT (возможно, с некоторой агрегацией данных) и INSERT. На практике часто возникает ситуация, когда большинство читающих запросов читают только часть данных из таблицы в диапазоне некоторого ключа. Обычно, этим ключом является дата или уникальный идентификатор пользователя (или пара их этих значений). Если хранить данные отсортировано по этому ключу, то запросы, требующие данные за определенные дни или по определенным пользователям, будут выполняться быстрее. Такой ключ называется первичным ключом. Для того, чтобы максимально эффективно выполнять запросы и учитывать первичный ключ, были реализованы движки семейства MergeTree.

Идея, стоящая за реализацией движков MergeTree, та же самая, что и в LSM-дереве. Данные таблицы хранятся в виде отдельных частей (в исходном коде структура, отвечающая за этот объект, называется IMergeTreeDataPart). Каждая часть является строго неизменяемой и хранит какую-то часть данных, отсортированных по первичному ключу. Если количество частей становится

слишком большим, то MergeTree сливает некоторые из частей в новую часть, поддерживая сортированность. Эта операция требует всего лишь линейное количество времени от размера сливаемых данных. Каждая вставка новых данных создает новую часть, предварительно сортируя данные по первичному ключу. При этом число частей не может превышать 300. Если оно достигает этого значение, то запрещаются все следующие вставки в таблицу до тех пор, пока операции слияние не закончатся.

Рассмотрим подробнее, как происходит процесс выполнения операций чтения, вставки и слияния, а также для понимания некоторые случаи, когда эти процессы происходят одновременно.

Для начала присвоим частям свой собственный номер. Новые части могут появиться только в двух случаях - вставка новых данных или слияние существующих частей. При вставке будем номер части будет равен номеру вставки. Из-за особенностей MergeTree, слияние может происходить только с такими частями, что итоговая часть содержит данные из подряд идущих вставок. В таком случае будем писать, что часть имеет номер [K, N], если она содержит данные из вставок с номера K по номер N.

- Вставка новых данных

Если происходит вставка с номером N, то мы получим часть N.

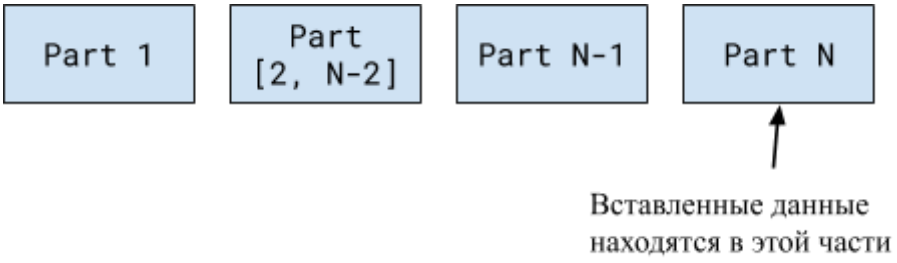

Рис. 4.1.1. Вставка новых данных

- Чтение данных

При чтении данных операция чтения сначала получает список активных на данный момент частей, а потом использует только этот список, несмотря на то,

что множество частей могло поменяться. Например, после завершения вставки на рисунке 4.1.1. операция чтения получит список $[Part_1, Part_{[2, N-2]}, Part_{N-1}, Part_n]$.

- Слияние

Как уже говорилось выше, при слияние может произойти только между теми частями, которые содержат диапазон номеров вставок без промежутков. При этом количество сливаемых частей ничем не ограничено - могут быть слиты как 2 части, так и 300. Так, на рисунке 4.1.2. могут быть слиты части 1 и [2, 5] и получится часть с номером [1, 5] или же могут быть слиты все части, получив часть с номером [1, 42]. Но, например, части 1 и 6 не могут быть слиты между собой.

Рис. 4.1.2. Возможные индексы частей

При этом при слиянии частей старые части никуда не пропадают - они остаются до тех пор, пока не будут удалены. Новая часть добавляются в список, а старые помечаются, как неактивными. Неактивные части удаляются через некоторое время (сейчас это 6 минут), если их никто не использует. Например, если будут слиты части [2, 5], 6 и [7, 11], то мы получим следующую картину:

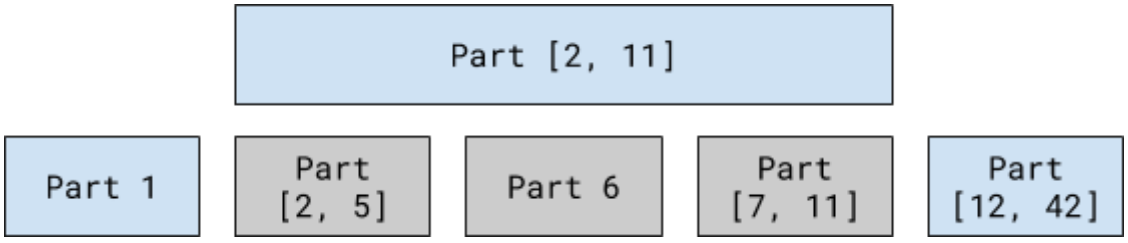

Рис. 4.1.3. Пример слияния частей

Неактивные части помечены серым цветом - это означает, что все последующие чтения из таблицы не будут видеть эти части. То есть, если

придет запрос на чтение, то он получит список частей $[Part_1, Part_{[2, 11]}, Part_{[12, 42]}]$. При этом эти части будут удалены через некоторое время.

Давайте теперь рассмотрим более сложный пример, когда несколько различных операций выполняются друг за другом. Пусть изначально мы имеет только 3 части с номерами 1, 2 и 3, после чего пришел запрос на чтение данных. На рисунке стрелками показано, какие части данных входят в список запроса.

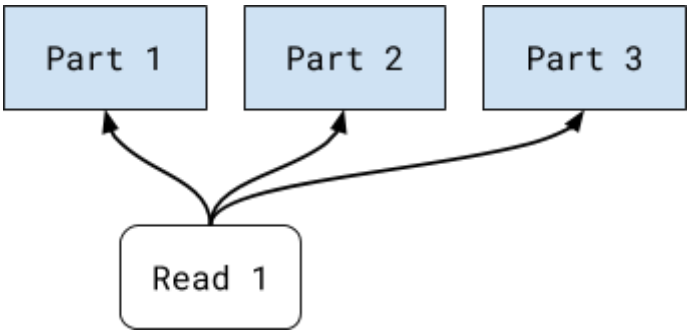

Рис. 4.1.4. Пример операции чтения

После этого приходит вставка новых данных и появляется часть с номером 4, а части 3 и 4 сливаются.

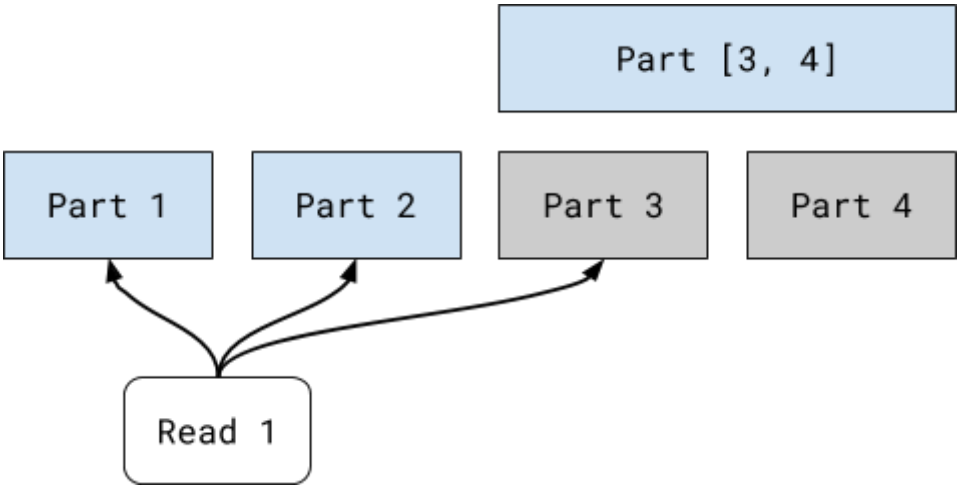

Рис. 4.1.5. Слияние после чтения

При том, что часть 3 больше не является активной, она все еще есть в списке операции чтения и не будет удалена до тех пор, пока операция чтения не закончится. Если придет новая операция чтения, то она уже получит другой

список частей, в котором не будет части с номером 3.

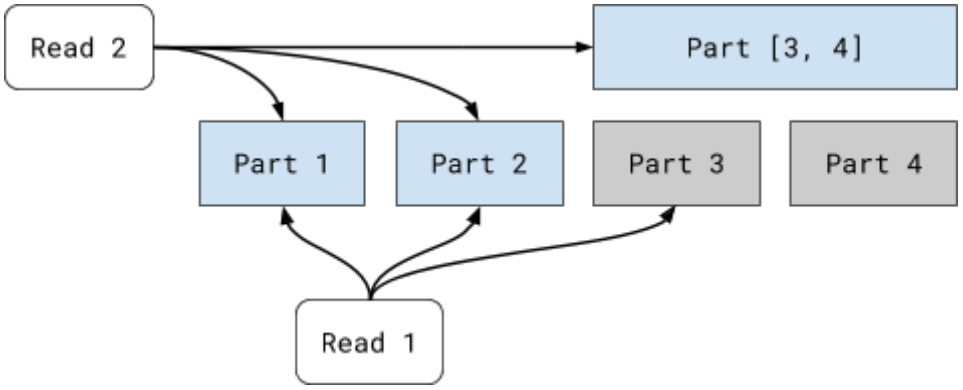

Рис. 4.1.6. Две операции чтения

Допустим, после этого произошло очередное слияние частей 2 и [3, 4], а также вторая операция чтения закончила свое выполнение.

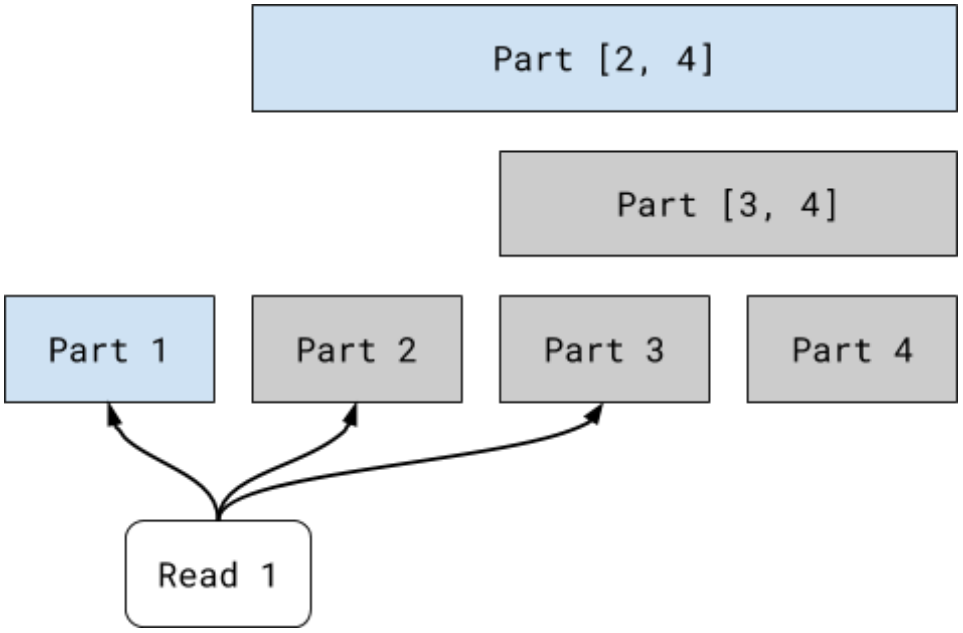

Рис. 4.1.7. Окончание операции чтения и слияние

По прошествию некоторого времени MergeTree попытается удалить части 3 и 4. Часть 4 будет успешно удалена, а часть 3 нет, так как все еще используется в операции чтения. Через еще какое-то время аналогичное произойдет с частями 2 и [3, 4]. В итоге получится то, что можно наблюдать на рисунке 4.1.8.

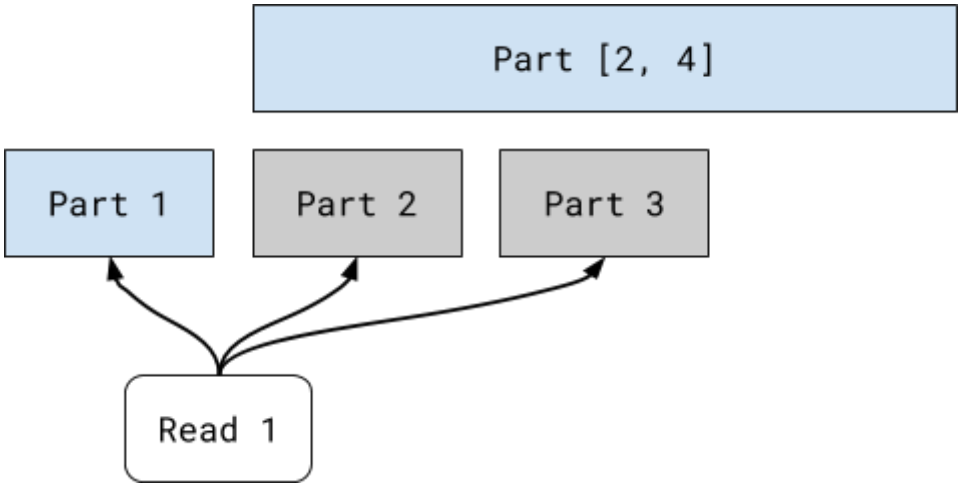

Рис. 4.1.8. Удаление старых частей

Как только операция чтения закончится, части 2 и 3 удаляться, и получится то, что на рисунке 4.1.9.

Рис. 4.1.9. Итоговые части

Все операции по добавлению новых частей (в случае вставки новых данных или же в случае слияния частей) или получение списка активных частей защищены одним мьютексом, что, впрочем, не оказывает большого влияния на производительность, так как критический код выполняется достаточно быстро, а основная работа операций вставки, чтения или слияния выполняется вне критического кода.

4.2. Алгоритм транзакций

Основываясь на архитектуре ClickHouse был придуман алгоритм, который позволит добавить транзакции в движки семейства MergeTree, не требуя при этом больших накладных расходов. Для этого алгоритма был использован подход MVCC. Опишем для начала, как поменялись структуры в ClickHouse.

- Каждый запрос имеет свой timestamp - монотонно возрастающее число
- Каждая часть теперь имеет 2 новых поля:

    - timestamp_created - timestamp операции, которая создала эту часть
    - timestamp_deleted - timestamp операции, которая удалила эту часть
- Каждый запрос дополнительно имеет еще два поля:
    - min_timestamp - минимальный timestamp среди всех выполняющихся в данный момент операций
    - timestamps_running - массив timestamp'ов всех операций, которые выполняются на момент получения массива

Общее описание работы алгоритма следующее - каждая операция (это может быть вставка новых данных, чтение или слияние частей) в начале своего выполнения получает свой собственный timestamp, а также значения min_timestamp и timestamps_running в данный момент. Разберем 3 случая работы алгоритма:

- Вставка данных

При вставки данных создается новая часть. При создании новой части будет присваивать значению поля timestamp_created значения timestamp операции. При этом timestamp_deleted равно бесконечности (в реализации максимальному значению целочисленной переменной)

- Чтение данных

При чтении данных операции необходимо получить список частей. Теперь же операция не просто получает только активные части, но может получить некоторые части, которые уже неактивны, или же не получить уже существующие активные части. Операция получает только те части, которые удовлетворяют следующим трем условиям:

1. $timestamp\_created \notin timestamps\_running$ - часть была создана до того, как нынешняя операция чтения начала свою работу. Для оптимизации можно сразу брать те части, которые удовлетворяют условию $timestamp\_created < min\_timestamp$, так как большинство частей будет удовлетворять этому критерию, который достаточно быстро проверяется

2. $timestamp\_created \leq timestamp$ - часть была создана до нашей операции
3. $timestamp < timestamp\_deleted$ или $timestamp\_deleted \notin timestamps\_running$ - часть либо еще не удалена, либо удалена в операции, которая началась после нашей операции или еще не завершилась.

- Слияние частей

Хоть слияние и не является SQL запросом, оно, как и запросы INSERT (для вставки данных) и SELECT (для чтения данных) будет получать timestamp. При этом сливать можно только те части, которые удовлетворяют следующим условиям:

1. Часть еще не была удалена, то есть timestap_deleted равняется бесконечности
2. $timestamp\_created \notin timestamps\_running$
3. $timestamp\_created \leq timestamp$

Новая часть, полученная при слиянии, получит timestamp_created равный timestamp'у операции слияния, а всем старым частям в timestamp_deleted будет присвоено то же самое значение timestamp'а операции.

Также нужно изменить алгоритм очистки старых частей (непосредственное удаление из списка частей). Раньше удалялись те части, которые помечались неактивными, при условии, что прошло достаточно времени после этого. Сейчас можно будет удалять те части, у которых timestamp_deleted меньше, чем timestamp всех выполняющихся в данный момент операций - то есть ни одна из нынешних или будущих операций не получит доступ к этой части. При этом условие на то, что должно пройти достаточно много времени остается, так как оно связано с синхронизацией диска.

Рассмотрим пример работы данного алгоритма. Дальше timestamp будем обозначать T, timestamps_running - TR, timestamp_deleted - TD, timestamp_created - TC, а бесконечность - INF. Пусть у нас было 2 части и

пришла операция чтения, которая не успела получить список частей. Значения параметров обозначены на рисунке 4.2.1.

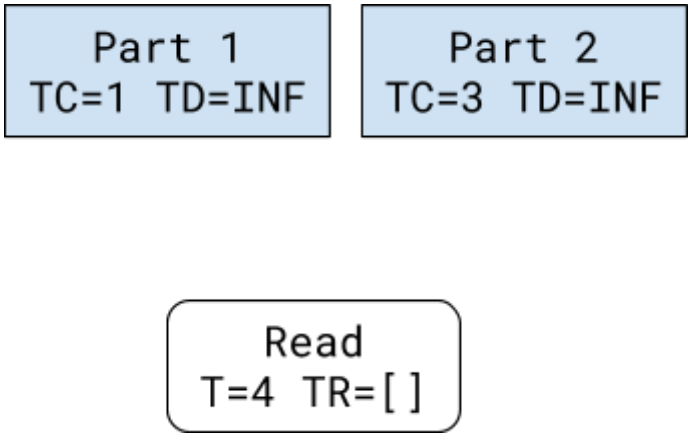

Рис. 4.2.1. Пример хранения частей

Дальше была добавлена новая часть на рисунке 4.2.2.

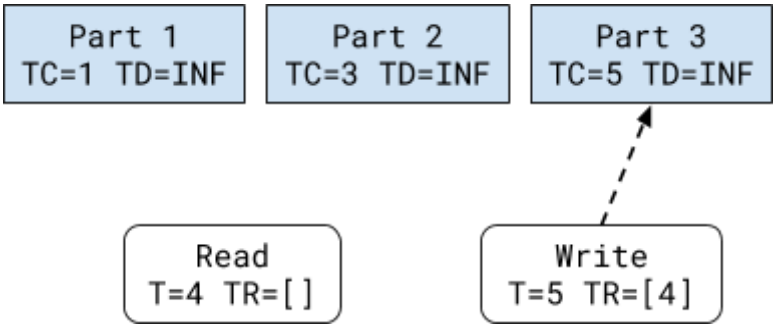

Рис. 4.2.2. Вставка новой части

После чего операция чтения закончилась, а появилась операция слияния с T=6, которая слила 2 и 3 части. Предположим, что прошло много времени, но операция чтения все еще по какой-то причине не попросила список частей. Тогда части 2 и 3 все еще не будут удалены из списка частей, так как потенциально могут пригодиться операции чтения.

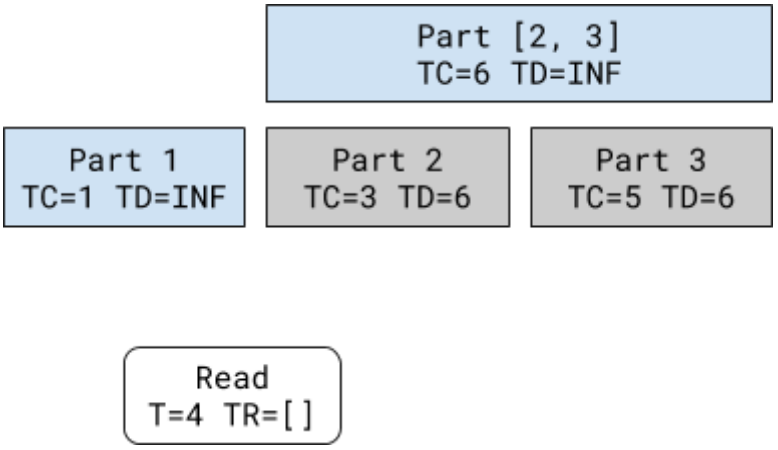

Рис. 4.2.3. Слияние частей

После запроса частей, операция чтения получит только части 1 и 2, так как только они удовлетворяют описанным выше условиям.

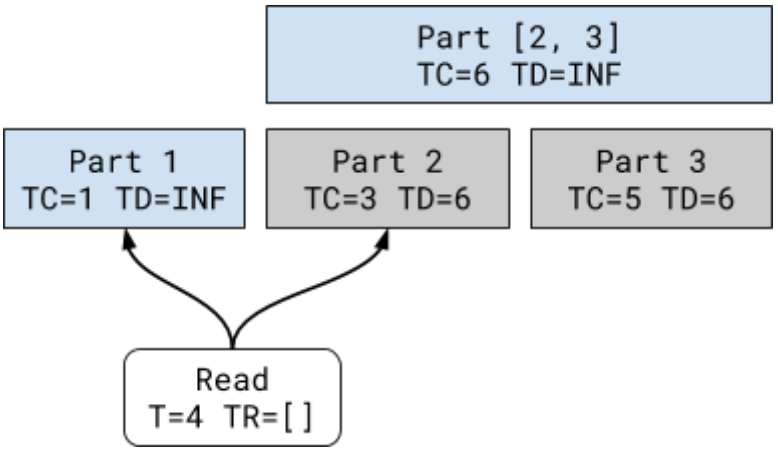

Рис. 4.2.4. Получение списка частей

После завершения операции чтения, старые части будут удалены, так как нет операций, у которых T меньше 6.

Рис. 4.2.5. Итоговое состояние

Рассмотрим еще один пример. Пусть у нас в таблице появились 2 новые части, но операции вставки этих частей еще не завершили работу.

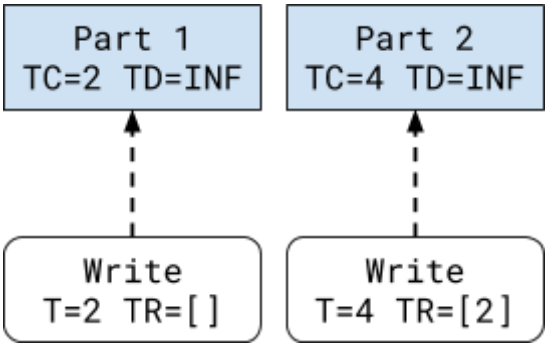

Рис. 4.2.6. Две вставки данных

После этого операция записи с T=4 завершила свою работу, но появилась операция чтения. Тогда операция чтения не сможет получить первую часть, так как связанная с ней операция записи еще не завершилась.

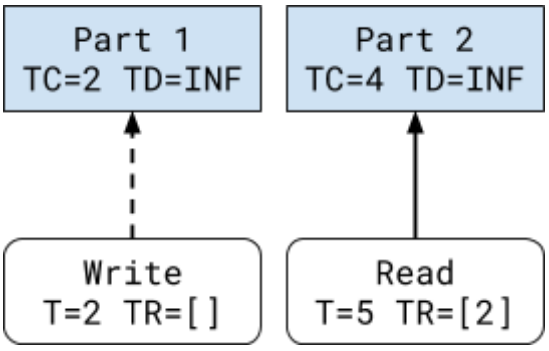

Рис. 4.2.7. Чтение с незавершенной вставкой

Для того, чтобы дать гарантию стойкости из ACID предлагается писать в файл на диск намерение о том, чтобы совершить транзакцию. Также, все части сначала пишуться на диск перед тем, как быть добавленными в рабочий набор частей MergeTree. При рестарте ClickHouse может проверить файл с описанием транзакции и восстановить ее, если все части успешно были записаны на диск, или удалить все части до того, как их мог кто-то прочитать, если произошла какая-то ошибка.

4.3. Специфика алгоритма в случае реплицированной таблицы

В ClickHouse существует движок хранения данных их семейства MergeTree, который может реплицировать данные на несколько серверов

ClickHouse. Он называется ReplicatedMergeTree. По этой причине необходимо расширить алгоритм на работу с этим движком данных.

Синхронизация в таких таблицах между разными серверами происходит посредством ZooKeeper. Это сервис, который позволяет для распределенных систем создавать файлы, которые называются ноды, в своей внутренней системе и записывать туда небольшое количество информации.

В случае реплицированных таблиц получение timestamp'ов можно будет реализовать через специальные последовательные ноды (sequential node). В ZooKeeper любую ноду можно объявить, как последовательной. В таком случае, такую ноду можно будет создавать множество раз, и каждому созданию такой ноды будет приписываться номер, который возрастает при каждом следующем создании.

В данный момент ClickHouse записывает в ZooKeeper в папку "/log/" список созданных частей, из которого другие реплики могут узнать о новых частях, которые необходимо добавить локально. По аналогии с этим предлагается добавить туда папку "/transactions/", в которую будут записываться все транзакции, которые хотят выполниться. После того, как транзакция будет успешно выполнена, туда же будет писаться информация об успешном выполнении. Либо же эту информацию можно писать в ту же самую папку "/log/". Это позволит выполнить гарантию стойкости из ACID в случае распределенных таблиц.

5. Заключение

Был разработан и предложен алгоритм, позволяющий добавить транзакции в архитектуру ClickHouse. Транзакции позволят пользователям избавиться от некоторых существующих проблем, а также добавят новые возможности при использовании ClickHouse.

Данный алгоритм удовлетворяет критериям ACID, которые необходимы для удобного использования транзакций. При этом накладных расходов создается не много по причине того, что получилось подстроить алгоритм под текущую архитектуру ClickHouse: части данных сами по себе неизменяемы, поэтому нет необходимости сохранять их разные версии, а также части данных уже пишутся на диск перед тем, как стать доступными для чтения.

Однако архитектура ClickHouse достаточно сложная, и для реализации транзакций требуется внести изменение в большом количестве логических мест, поэтому алгоритм на данный момент находится на стадии реализации и, потенциально, может претерпеть некоторые изменения.

6. Список литературы

[1] Evolution Of Data Structures In Yandex.Metrica - High Scalability. [Электронный ресурс] URL: http://highscalability.com/blog/2017/9/18/evolution-of-data-structures-in-yandex metrica.html (дата обращения: 18.05.2020)

[2] Performance comparison of analytical DBMS - ClickHouse Documentation. [Электронный ресурс] URL: https://clickhouse.tech/benchmark/dbms/ (дата обращения: 18.05.2020)

[3] MergeTree - ClickHouse Documentation. [Электронный ресурс] URL: https://clickhouse.tech/docs/en/engines/table-engines/mergetree-family/mergetre e/ (дата обращения: 18.05.2020)

[4] "The Transaction Concept: Virtues and Limitations" (PDF). Proceedings of the 7th International Conference on Very Large Databases. Cupertino, CA: Tandem Computers. pp. 144–154. Retrieved March 27, 2015.

[5] ACID Implementation - Wikipedia. [Электронный ресурс] URL: http://en.wikipedia.org/wiki/ACID#Implementation (дата обращения: 18.05.2020)

[6] DBMS Concurrency Control: Two Phase, Timestamp, Lock-Based Protocol. [Электронный ресурс] URL: https://www.guru99.com/dbms-concurrency-control.html (дата обращения: 18.05.2020)

[7] Snapshot isolation - Wikipedia. [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Snapshot_isolation (дата обращения: 18.05.2020)

[8] High Performance MySQL, 2nd Edition - Oreilly. [Электронный ресурс] URL: https://www.oreilly.com/library/view/high-performance-mysql/9780596101718/ ch01.html (дата обращения: 18.05.2020)

[9] How Postgres Makes Transactions Atomic - Brandur. [Электронный ресурс]

URL: https://brandur.org/postgres-atomicity (дата обращения: 18.05.2020)

[10] Transactions - Exasol [Электронный ресурс] URL: https://www.exasol.com/portal/display/SOL/Transactions (дата обращения: 18.05.2020)

[11] Transaction Conflicts for Mixed Read/Write Transactions - Exasol. [Электронный ресурс] URL: https://www.exasol.com/portal/pages/viewpage.action?pageId=22518143 (дата обращения: 18.05.2020)

[12] Zhang Ying; Nedev Dimitar; Koutsourakis Panagiotis; Kersten Martin; Distributed Processing and Transaction Replication in MonetDB - Towards a Scalable Analytical Database System in the Cloud. 2016. [Электронный ресурс] URL: http://www.exanest.eu/pub/2016_RTPBD_Monetdb.pdf (дата обращения: 18.05.2020)