Выпускная квалификационная работа

ПОДДЕРЖКА ИСПОЛЬЗОВАНИЯ В CLICKHOUSE СИСТЕМ КООРДИНАЦИИ ПОМИМО ZOOKEEPER

Левушкин Алексей Сергеевич БПМИ 165

Миловидов Алексей Николаевич Доцент, М базовая кафедра Яндекс

Координация реплик ClickHouse

ClickHouse – современная, все больше набирающая популярность по всему миру СУБД, которая как и любая современная система управления базами данных заточена на скорость и надежность. Один из аспектов надежности достигается за счет реплицируемых таблиц. Для их хранения используется сразу несколько реплик, в целях координации которых ClickHouse использует Apache ZooKeeper. Но не все пользователи ClickHouse хотят использовать его для репликации. Из-за чего актуальной задачей становится поддержание альтернативной системы координации, схожей по своему функционалу. Наиболее близкая из таких систем - Etcd, о которой и пойдет речь.

Актуальность задачи

- Надежность один из важнейших аспектов СУБД
- ClickHouse уже использует ZooKeeper для координации реплицируемых таблиц
- ZooKeeper стар и с каждым годом все менее актуален из-за чего пользователи хотят использовать альтернативную систему координации, подтверждением этому является популярное issue в репозитории проекта
- А значит поддержание альтернативной системы координации становится актуальной темой несмотря на то что задача реализует дополнительный а не новый функционал

Цель и задачи дипломной работы

Целью работы является возможность использования альтернативной системы координации

Задачи дипломной работы:

- Исследовать алгоритм репликации таблиц ClickHouse
- Исследовать систему ZooKeeper, которая используется для координации реплик
- Найти систему наиболее близкую к функционалу ZooKeeper
- Разработать схему хранения данных оптимизировав ее под выбранную систему
- Реализовать код взаимодействия с выбранной системой

Формальная постановка

- Есть СУБД ClickHouse надежность которой достигается за счет реплицируемых таблиц
- Эти реплицируемые таблицы уже координирует key-value хранилище ZooKeeper
- Есть потребность в замене Apache ZooKeeper на другую систему координации
- Новая система должна быть современнее, а так же более прозрачной для пользователей
- Для того что бы подменить систему координации, необходимо адаптировать API под схему хранения данных в ZooKeeper
- И реализовать код взаимодействия ClickHouse с альтернативной системой координации

Обзор используемых систем

ClickHouse

ClickHouse – open-source столбцовая СУБД для онлайн обработки аналитических запросов, в основе работы которой лежит семейство движков MergeTree, основанных на разделении таблиц на блоки данных, сливаемых в фоне.

Движок этого семейства под названием ReplicatedMergeTree обладает такими же свойствами, а также поддерживает репликацию данных. В отличие от других СУБД, в ClickHouse реплицируются именно таблицы, а не сервера.

Алгоритм репликации ClickHouse

- T - таблица семейства ReplicatedMergeTree, которая содержит в себе блоки b_1, …, b_n
- В ClickHouse сервер приходит клиент, и записывает новые данные
- Создается новый блок, который записывается на ту реплику, на которую пришел клиент
- Информация про записанный блок отображается в ZooKeeper
- Остальные реплики заметив и считав информацию из ZooKeeper начинают скачивать этот блок себе по средствам обычного http-запроса
- Для такой координации реплик используется ZooKeeper

ZooKeeper

Распределенное key-value хранилище, с помощью API которого можно управлять простыми объектами данных, организованных иерархически, как в файловых системах.

Данные в ZooKeeper представлены в виде znode, схожих с узлами в файловой системе, где полный путь до него это key, а содержимое это value. По аналогии с файловой системой, узлы имеют детей.

Znode бывают 3 типов:

- Обычные
- Эфимерные - которые существуют пока жив создавший его клиен
- Последовательные - в путь к которым добавляется последовательный номер (например /blocks/block-0000000001)

Важной особенностью является «watch» запросы, которые позволяют подписываться и следить за изменениями узлов

ZooKeeper API

- create(path, data, flags) – создает znode с переданными параметрами
- delete(path, version) – удаляет znode если переданная версия коректна
- exists(path, watch) – сообщает о наличии znode, и устанавливает флаг наблюдения если такой передан
- getData(path, watch) – возвращает данные znode и устанавливает флаг наблюдения
- setData(path, watch) – устанавливает новое значение znode если переданная версия корректна
- getChildren(path, watch) – возращает список имен всех потомков znode
- check(path, version) – проверяет соответствие версии вершины
- multi(requests) – атомарно применяет все запросы, если все применились успешно, иначе не применяет ни один (поддерживаются только create, remove, set, check).
- sync(path) – ждет завершения всех обновлений

* передаваемая версия корректна, если она совпадает с версией znode или равна -1

Использование ZooKeeper в ClickHouse

Как было сказано раннее, любая таблица ClickHouse состоит из блоков. Так как эти блоки имеют большой размер, ZooKeeper лишь оркестрирует их метаданными, а также метаданными реплик.

Примеры данных, которые хранятся в ZooKeeper:

- метаданные реплики - такая информация как активность реплики, наличие блоков таблицы, обрабатываемые сейчас операции и другие данные, записывается в потомков узла /replicas.
- хэш каждого блока для последующей проверки блока на целостность, записывается в потомков узла /blocks.
- информация о появлении новых блоков, записывается в /log.

Etcd

Распределенное key-value хранилище для наиболее важных данных распределенных систем.

Использует алгоритм консенсуса RAFT, что делает его более простым в реализации, чем ZooKeeper с ZAB.

Взаимодействие осуществляется с помощью gRPC.

Мы будем использовать 3 grpc сервиса: KV, Watch, Lease. И следующие запросы:

- TxnRequest(compares, success_ops, failure_ops) – транзакция со сравнениями, а так же операциями, которые необходимо применить в зависимости от результата compares. ops = {Range, DeleteRange, Put}
- WatchRequest(key) – запрос на создание watch
- LeaseGrandRequest(ttl) – запрос на создание нового lease
- LeaseKeepAliveRequest(lease_id) – запрос на обновление ttl у lease

gRPC

Высокопроизводительный фреймворк для вызова удаленных процедур (RPC).

Производительность достигается за счет использования протокола HTTP/2 и Protocol Buffers.

Асинхронный режим достигается благодаря использованию очереди запросов CompletionQueue. Алгоритм работы с которой выглядит так:

- связываем запрос с ComplitionQueue уникальным тегом
- вызываем метод ComplitionQueue::Next, блокирующий поток, до того момента пока не будет получен ответ какого либо из завершившихся запросов
- ComplitionQueue возвращает тег, связанный с запросом.

ZooKeeper vs Etcd

Минусы ZooKeeper:

- Написан на языке java, а значит нуждается в JVM фиксированной версии на каждом узле кластера, а так же наследует множество проблем java (garbage collector и тд)
- Настройка требует множество конфигурации, что увеличивает порог входа
- Очередной проблемой является недоступность сервиса при создании снепшотов

Сильные стороны Etcd:

- Новая, активно разрабатывающаяся система
- Написан на языке go, благодаря чему это всего лишь бинарный файл без конфигураций, лишенный описанных выше болячек

Шаги для решения задачи

gRPC в ClickHouse

- Для того что бы взаимодействие через grpc стало возможным необходимо добавить gRPC в ClickHouse как third_part
- Так как gRPC использует библиотеки, которые уже используются в ClickHouse необходимо было переписать CMakeLists для gRPC переиспользовав их, а так же соблюдая правила стиля CMake принятые в ClickHouse

Анализ ключевых различий систем

- Плоское хранилище Etcd не позволяет использовать понятия «ребенок» и «родитель», которыми оперирует ZooKeeper
- Znode в ZooKeeper – это не только ключ – значение, а еще и набор метаданных, изменяющихся атомарно, при изменении хранилища (например numChildren)
- В ZooKeeper создание watch запроса атомарно с основным запросом, в Etcd – это два отдельных запроса
- Узлы ZooKeeper делятся на 3 типа (обычные, эфимерные и последовательные), в Etcd поддерживаются лишь обычные узлы
- И наиболее важное отличие – это различие в определении операции «транзакция»

Изучение похожих решений

Zetcd – open-source проект, который как и Etcd написан на языке Go и является некоторой прокси, которая принимает вызовы отправленные в ZooKeeper, интерпретирует и отправляет их в Etcd, тем самым позволяет работать с Etcd по правилам ZooKeeper API. Данная библиотека не подходит для задач репликации в ClickHouse, так как не может использоваться как часть кода, имеет низкую производительность. А так же не развивается на данный момент, из-за чего не совместим с используемой в ClickHouse версией Zook

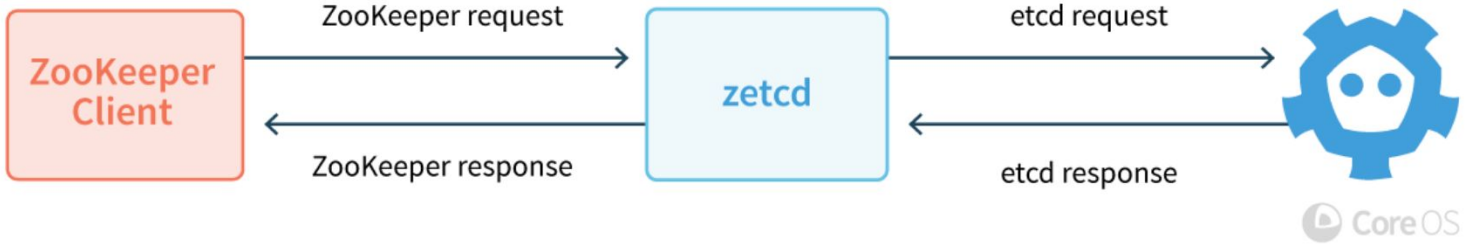


Рис.1 Иллюстрация использования zetcd как прокси

Разработка схемы хранения

- Добавим уровень в ключ (например /1/clickhouse) что бы оперировать понятиями родитель-ребенок
- Создадим по отдельному узлу для метаданных (например /1/ctime/clickhouse)
- Так как любой запрос состоит из 2 и более подзапросов будем использовать только транзакции
- Для эфимерных узлов воспользуемся сервисом Lease, для последовательных узлов создадим узел со счетчиком и будем создавать последовательный узел за два запроса
- Для того что бы удалить вершину, необходимо проверить нет ли у нее потомков, для этого снова воспользуемся двумя запросами
- Вышеописанные пункты вводят такое понятие как составной запрос – это запрос, который состоит из предварительного запроса для получения метаданных и основного запроса изменяющего состояние хранилища
- Тогда мультизапрос будет составным, если хотя бы один из подзапросов составной

Пример составного запроса (удаление узла)

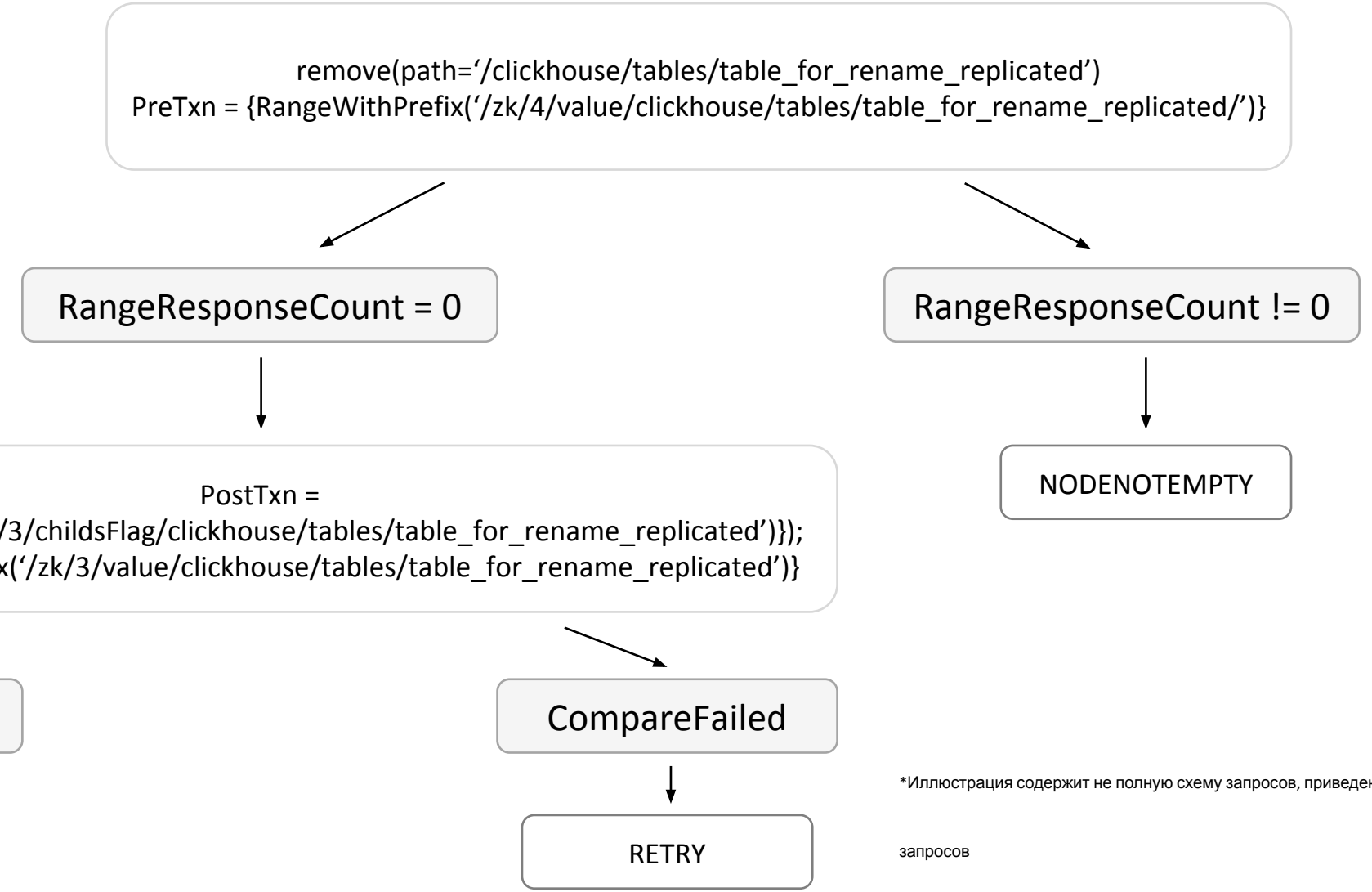


*Иллюстрация содержит не полную схему запросов, приведена для лучшего понимания составных запросов

Пример составного запроса (создание seq узла

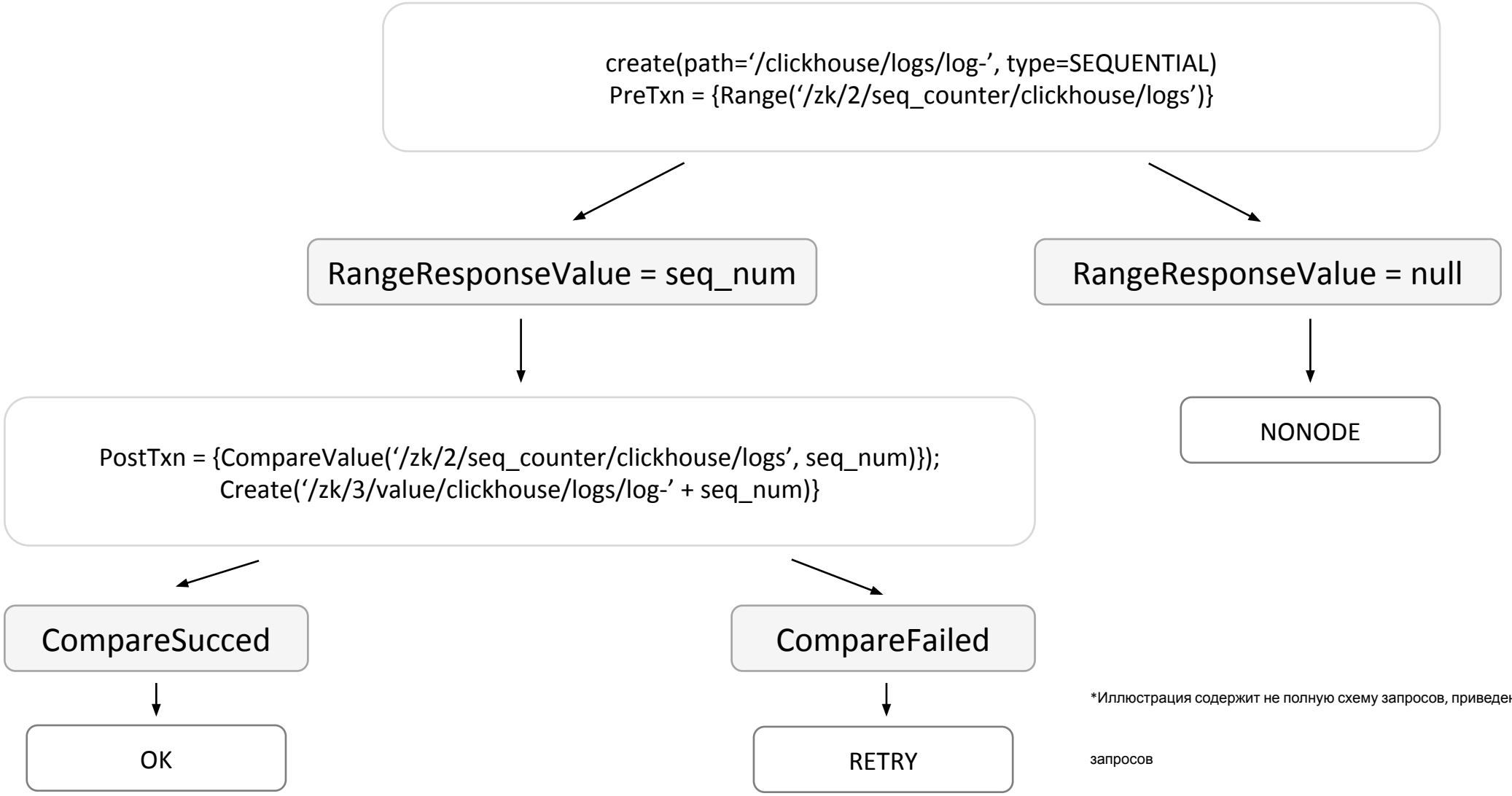


*Иллюстрация содержит не полную схему запросов, приведена для лучшего понимания составных запросов

Почему с мульти-запросами все сложнее

- Пусть есть multi_request состоящий из запросов r_1, r_2….r_n. И состояние хранилища X до применения multi_request-а.
- Тогда выполнение мультизапроса будет выглядеть как поочередное применение запросов, где r_i применяется к состоянию x_{i-1}, где состояние x_j – это примененные запросы r_1…r_j к состоянию X
- Тогда запрос либо применяется полностью, либо не применяется вовсе
- Смоделируем ситуацию в которой X = {/root}, r_1 = create(/root/tmp), r_2 = create(/root/tmp/x1), r_3 = set(/root/tmp, “tmp dir”), r_4 = remove(/home), в таком случае, multi_request не применится с ошибкой NONODE при применении r4.
- Мультизапрос в нашей схеме – это составной запрос полученный путем объединения подзапросов, из-за чего предварительные запросы применяются к состоянию X, а не к состоянию x_i

Как модифицировать схему составных запрос

- Учитывать создание родителя или удаление детей в соседних подзапросах
- Обрабатывать создание сразу нескольких последовательных узлов для одной директории
- Обернуть все запросы изменяющие состояние сравнениями
- Так как результат сравнения един для всех сравнений, необходимо продублировать каждое сравнение RangeRequest-ом
- Доводить каждый запрос до конца, даже если он неуспешен на этапе предварительного запроса, и уже после завершения всех запросов выбирать первый неуспешный
- Обрабатывать запросы для проверки существования узла (create, remove), как Compare

Реализация

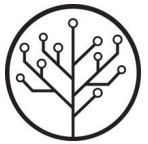
Так как код ClickHouse написан на языке C++, а также все составляющие принято разрабатывать как часть кода, а не отдельные подсистемы, был реализован класс EtcdKeeper, на основе интерфейса IKeeper.

Работа класса состоит из 5 потоков:

1. Создает запросы и добавляет их в очередь на обработку
2. Достает запросы из очереди и отправляет в Etcd, а так же watch запросы и LeaseKeepAlive запросы
3. Обрабатывает ответы из kv completion queue
4. Обрабатывает ответы из watch completion queue
5. Обрабатывает ответы из lease completion queue

блокируются вызовом ComplitionQueue::Next

Результат



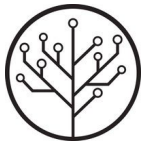
На данный момент PR на этапе ревью и нуждается в доработках, так как из-за неатомарного создания watch-ей реализация не эффективна при высоких нагрузках.

Сравнение производительности

Test name	TestKeeper	ZooKeeper	Zetcd	Etcd
01277_alter_rename_column_constraint_zookeeper	0.2	0.4	-	0.7
01213_alter_rename_column_zookeeper	0.5	0.6	-	0.7
01213_alter_rename_primary_key_zookeeper	1.17	1.25	-	1.37
01213_alter_rename_with_default_zookeeper	0.18	0.23	-	0.3
01135_default_and_alter_zookeeper	0.17	0.2	-	0.28
01103_optimize_drop_race_zookeeper	15.3	15.3	-	16
01090_zookeeper_mutations_and_insert_quorum	0.2	0.42	-	0.9
01079_alter_default_zookeeper	0.2	0.3	-	0.4
01079_bad_alters_zookeeper	3.7	3.7	-	3.8

Рис.2 Сравнение по скорости разных систем координации на примере тестов.

Источники



1. PR[online] // GitHub 2020 URL: https://github.com/ClickHouse/ClickHouse/pull/8435
2. PR[online] // GitHub 2020 URL: https://github.com/ClickHouse/ClickHouse/pull/10376
3. ClickHouse [online] // https://clickhouse.tech/docs/en/
4. ZooKeeper [online] // https://zookeeper.apache.org/
5. Etcd [online] // https://etcd.io/docs/v3.4.0/
6. Zookeeper [online] // https://static.usenix.org/event/atc10/tech/full_papers/Hunt.pdf
7. gRPC [online] // https://grpc.github.io/