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
Виртуальная файловая система для ClickHouse

Выполнил студент гр. **БПМИ165**, 4 курса,
Ершов Олег Владиславович

Руководитель ВКР:
Доцент, М базовая кафедра Яндекс,
Миловидов Алексей Николаевич

Содержание

Abstract

ClickHouse is an open source column-based analytical database management system aimed at online processing of analytical queries. The main feature of ClickHouse is its amazing performance on large datasets. All large companies process huge amounts of data every second, and distributed file systems are becoming more and more popular because of their scalability, reliability, and safety. The HDFS is an open source project, which stands out against this background. By default, ClickHouse cannot work on top of the file systems that are not compatible with POSIX. In this paper, we look at the architecture of each of these systems in detail and offer a solution that allows you to use ClickHouse on top of HDFS.

Аннотация

ClickHouse - столбцовая аналитическая система управления базами данных с открытым исходным кодом направленная на онлайн обработку аналитических запросов. Главная особенность ClickHouse - это поразительная производительность на больших объемах данных. Все крупные компании каждую секунду обрабатывают огромные объемы данных, и все более популярным решения становятся распределенные файловые системы, из-за своих показателей масштабируемости, доступности и сохранности. На их фоне выделяется проект с открытым исходным кодом HDFS. По умолчанию ClickHouse не может работать поверх файловых систем которые не совместимы с POSIX. В этой работе мы подробно рассматриваем архитектуру каждой из этих систем и предлагаем решени, которое позволяет использовать ClickHouse поверх HDFS.

1 Введение

Обработка данных и системы созданные для их обработки являются ключевыми в таких крупных компаниях в отрасли информационных технологий как Яндекс, Google, Amazon. Они обрабатывают десятки тысяч пользовательских запросов в секунду, и во много раз больше во внутренних сервисах компаний. Большая часть их бизнеса построена на использовании огромных объемов данных, благодаря которым учатся сложные нейронные модели, строятся подробные статистики и прочее. Поэтому зачастую размер не только всех данных, а даже одного файла может превышать несколько петабайт и это значительно превышает вместимость одной машины. Для работы с такими объемамми данных компаниии вынуждены существенно инвестировать и разработку инфраструктурных решений, которые позволят обрабатывать такое количество информации.

Google первым предоставил свое решение данной проблемы в виде Google File System (GFS), масштабируемая распределенная файловая система для огромных распределенных продуктов с интенсивным использованием данных [**?**]. Это ознаменовало собой прорыв в развитии науки о распределенных алгоритмах и системах. Первоначально GFS состояла из единственного мастера, который обслуживал запросы клиентов и хранил метаинформацию о файловой системе и множество chunk-серверов, на которых непосредственно хранились сами данные. Google File System не совместима с Portable Operating System Interface (POSIX), но вместо этого предоставляет схожий интерфейс. На основе опубликованной статьи позднее была создана Hadoop Distributed File System - проект фонда Apache Software Foundation с открытым исходным кодом [3]. Значительная часть кодовой базы, около 80%, была написана в Yahoo в 2006 году для обработки около 25 петабайт данных.

Большинство компаний, которые не могут позволить себе собственные крупные инфраструктурные разработки, такие как Dropbox, Twitter, используют готовые решения в виде HDFS или Amazon Simple Storage

Service (Amazon S3). Amazon S3 - сервис от непосредственно компании Amazon, который предоставляет объектное хранилище, которое использует свою распределенную файловую систему внутри. Распределенные хранилища предоставляют такие необходимые показатели как неограниченное горизонтальное и вертикальное масштабирование, высокая доступность и безопасность данных. Также такие системы зачастую сопровождаются дополнительными сервисами, которые позволяют нативно выполнять аналитиеские SQL запросы или запускать задачи в парадигме MapReduce над хранимыми данными.

ClickHouse - столбцовая система управления базами данных (СУБД) для онлайн обработки аналитических запросов (OLAP). Столбцовая система хранения отличается от классической строковой, которая используется в большом количестве СУБД, например Postgres, MySQL, MS SQL Server, тем что значения относящиеся к одной строке физически не хранятся рядом, а значения разных столбцов хранятся отдельно, а одного столбца вместе. ClickHouse отличается своей впечатляющей производительностью, эффективной утилизацией железа и линейной масштабируемостью. Бенчмарки показывают, что ClickHouse значительно превосходит своих конкурентов, например он более чем в три раза быстрее чем Vertigo [?]. Таким образом ClickHouse обладает значительным числом преимуществ, но он обязательно требует от файловой системы совместимость с POSIX. Соответственно ClickHouse может быть развернут локально на одной машине, либо над целом кластером, но в таком случае требуется значительная работа по его конфигурации и настройки.

Учитывая широкую распространенность ClickHouse среди множества пользователей, возможность использовать поразительную скорость ClickHouse для работы с аналитическими запросами, используя даннные, которые хранятся в надежном распределенном хранилище от Hadoop.

1.1 Цели и задачи

Основной целью данной выпускной квалификационной работы является реализация поддержки HDFS в качестве файловой системы ClickHouse. Для этого были поставлены следующие задачи:

- Изучить существующие решения по использованию распределенных систем в качестве хранения данных.
- Рассмотреть подходы к обработке данных
- Изучить архитектуру HDFS, ее особенности и ограничения
- Разобрать внутреннее устройство ClickHouse, его интерфейсы по работе с файловой системой
- Спроектировать и реализовать программное решение

2 Обзор литературы

2.1 FUSE

Filesystem in Userspace (FUSE) - свободный модуль для UNIX и UNIX-подобных операционных систем, который позволяет пользователям без привилегий создавать свои собственные файловые системы без правок в код ядра. Это достигается за счет запуска программ в пользовательском пространстве, в то время как FUSE модуль маршрутизирует все связанные со смонтированной файловой системой системные вызовы.

Проект FUSE зачастую состоит из двух компонент модуля ядра fuse и библиотеки пользовательского пространства libfuse, которая предоставляет эталонную реализацию для коммуникации с FUSE модуля ядра.

Hadoop's Fuse-DFS - модуль, который позволяет монтировать HDFS как стандартную файловую систему. Fuse-DFS реализован на C используя библиотеку libhdfs как интерфейс к HDFS. Сама библиотека исполь-

зует Java Native Interface (JNI) для вызова клиента файловой системы для Java.

Как показывают исследования E. Sangaline и J. Lauret, использование Fuse-DFS в среднем замедляет чтение из HDFS в 2 раза [6].

2.2 HBase

HBase - распределенная столбцовая база данных, построенная поверх HDFS [9]. HBase это приложение Hadoop, которое используется для произвольного чтения и записи в режиме реального времени.

Существует бесчисленное множество различных подходов к построению баз данных, но большинство реляционных решений специально не проектировались с учетом большого масштабирования. Тем не менее все проекты предоставляют решения для репликации и масштабирования, чтобы расширить используемые мощности за пределы одной машины, но как правило они получаются недостаточно эффективными и сложными в установке и использовании. Также большинство функций СУБД, как joins, сложные запросы, вторичные ключи сильно деградируют в производительности.

Архитектура HBase специально построена для решения задач линейной масштабируемости за счет добавления новых узлов. HBase нереляционная база данных и не поддерживает Sql, но она значительно лучше подходит для хранения больших разряженных таблиц.

Данные в HBase хранятся в таблицах, которые разбиваются на регионы, которые состоят из нескольких подряд идущих строк. Именно регионы являются единицой репликацией. Сам сервис состоитиз трех компонент master server (HMaster), regionservers и ZooKeeper. Главная задача HMaster - распределять регионы между region servers, и балансировать нагрузку между ними. Regionservers обрабатывают сами регионы, хранят их данные, метаинформацию и лог операций, а также отвечают клиенту на запросы по обработке данных. ZooKeeper мониторит состояние region servers и восстанавливает данные при их отказе или недо-

ступности. Также ZooKeeper является координатором между HMaster и regionservers и поддерживает целостность системы.

Как и ClickHouse, HBase - столбцовая база данных и поддерживает сложные аналитические запросы. Результаты тестирования производительности на одинаковом датасете показывают, что HBase в 10 раз медленнее чем ClickHouse. Также кроме аналитических запрсоов HBase поддерживает множество других возможностей для обработки данных, например нативный MapReduce который позволяет обрабатывать терабайты и петабайты данных с относительно быстрым произвольным чтением и записью.

2.3 Amazon S3

Amazon предлагает набор различных сервисов для решения различных пользовательских задач[10]. Amazon Simple Storage Service (Amazon S3) - сервис для хранения объектов, который предоставляет передовые показатели по масштабируемости, производительности, безопасности и доступности данных. Он разрешает хранить объекты любого типа, что позволяет решать задачи как создание хранилища для Интернет приложений, резервное копирование и восстановление данных, хранение архивов данных. Устройство внутренней архитектуры публично не раскрывается, но она обещает высокую сохранность данных до 99.999999999% и 99.99% доступности.

Amazon DynamoDB - система управления базами данных класса, которая поддерживает формат данных в виде ключ-значение, так и подокументную. В отличие от многих сервисов Amazon, что подписчики оплачивают требуемую произовдительность, а не потребляемое емкость хранения. Она также предоставляет линейную масштабируемость и отличается от многих других систем, что имеет архитектуру мульти-мастера. Поэтому конфликты с различными версиями разрешаются на стороне клиента, DynamoDB использует синхронную репликацию в нескольких датацентрах для высокой доступности и сохранности данных.

Amazon FSx for Lustre - сервис, позволяющий легко и эффективно пользоваться самой популярной высокопроизводительной файловой системой Lustre [12]. Ее часто используют для задач чувствительных к производительности, например машинное обучение, обработка видео и построение финансовых моделей. Lustre - продукт с открытым программным кодом, который был создан для быстрой и эффективной обработки самых больших датасетов в мире, и именно ее используют в самых быстрых суперкомпьютерах мира. Она обеспечивает задержку до нескольких миллисекунд, пропускную способность до нескольких сотен гигабайт в секунду и миллионы операций ввода-вывода.

Главным преимуществом помимо высокой производительности, доступности, масштабируемости сервисов Amazon, является то что пользователю никак не нужно управлять ресурсами в облаке. Все операции по обновлению программного обеспечения, обслуживания серверов, резервного копирования данных и их репликации выполняются за пользователя, который пользуется готовым продуктом и может сакцентировать свое внимание на бизнес логике приложения. За такие услуги необходимо платить, причем единицей может считаться разный объект в зависимости от продукта. Также пользователь полностью полагается на авторитет компании, предоставляющей услуги, потому что он никак не контролирует возможную утечку приватной информации, долгую недоступность серверов. В то время как сервисы с открытым програмным кодом как HDFS или ClickHouse можно разворачивать на собственных мощностях к которым есть непосредственный доступ.

2.4 Вывод

Были рассмотрены и изучены самые популярные решения по работе с распределенными файловыми системами и способами обработки информации поверх них. Обнаружилось что нет готовых продуктов, работающих поверх HDFS, которые бы обеспечивали сравнимую с ClickHouse производительность тяжелых аналитических запросов. Решения с за-

крытым программным обеспечением имеют свои серьезные преимущества в удобстве, но из-за невозможности полностью контролировать протекающие в них процессы не могут служить полным решениям для ряда задач. Учитывая заметный успех и популярность ClickHouse у мирового сообщества, можно быть уверенным в том, что возможность запускать тяжелые аналитические запросы поверх распределенной файловой системы от Hadoop будет высоко востребована.

3 Глава 2

3.1 HDFS

Для начала работы нам требуется разобрать архитектуру Hadoop Distributed File System [7]. Данные в HDFS по умолчанию разбиваются на блоки по 128 мегабайт. Полученные блоки - уже независимые объекты, которые хранятся уже на сервисах DateNodes. Создание абстракции блоков для распределенной файловой системы имеет ряд преимуществ. Во-первых, в такой конфигурации файл может иметь неограниченный размер, так как он необязательно должен целиком хранится на одной машине. Во-вторых, переход к единице хранения в виде блока, а не произвольного файла, сильно упрощает устройство самой файловой системы. Поскольку все блоки имеют фиксированный размер, легко вычислить объем оставшегося свободного места и для каждого блока можно хранить меньшее количество метаинформации. Поскольку блоки - просто фрагмент данных, то настройки доступа, разрешенный объем и другое можно хранить в отдельном сервисе. Кроме того абстракция блоков хорошо подходит для репликации с целью достичь отказоустойчивости и доступности. Для защиты от повреждения диска или отказа машины, каждый блок нужно хранить на нескольких различных физических машинах. При отказе одной машины, его легко прочитать с еще живой реплики прозрачно для клиента и в фоновом процессе реплицировать на другие. Точно также сами приложения могут запросить высокий коэф-

фицент репликации для часто запрашиваемых файлов, чтобы распределить нагрузку на них.

Кластер HDFS состоит из двух типов узлов, которые работают по патерну мастер-рабочие, NameNode - мастер и некоторое число рабочих - DateNodes. NameNode управляет пространством имен файловой системы, она поддерживает дерево файловой системы и метаинформацию для всех файлов и директорий. Эта информация хранится персистентно на диске в двух файлах: лога изменений и image namespace. Метаинформация представлена в структуре inode, которая в себе содержит различную информацию, например права доступа, дату и время создания или последнего изменения, размер квоты на диск. Также для каждого файла имеется информация обо всех DateNodes, которые хранят блоки данного файла, но она хранится не персистентно, поскольку перезапрашивается при старте кластера.

DateNodes - основные узлы системы, именно они хранят блоки, и отвечают на запросы NameNode или клиенты. Для поддержания связности кластера, DataNodes регулярно сообщают NameNode список хранимых блоков.

Клиент взаимодействует с файловой системой от имени пользователя. Сперва запрос на операцию отправляется на NameNode, которая сообщает адреса необходимых DateNodes. После этого клиент осуществляет операции чтения и записи непосредственно с DateNodes. Пользователю клиент представляет интерфейс, который очень похож на POSIX, поэтому пользователю нет необходимости знать об конкретнос расположении NameNode или DateNodes.

При описанной архитектуре NameNode является единой точкой отказа системы. Если машины, на которых она была запущена, уничтожатся, то пропадет дерево файловой системы и вся метаинформация о хранимых файлах, которую будет невозможно восстановить. Поэтому особенно актуальна задача обеспечения отказоустойчивости мастера и Hadoop предоставляет для этого два механизма. Во-первых, NameNode

может писать свою информацию в несколько файловых систем, в том числе на несколько дисков. Во-вторых, существует возможность запустить Secondary NameNode. Ее главная задача - мержить логи изменений с image log и хранить промежуточные результаты. Это делается для того, что бы лог изменений не становился слишком большим. Таким образом при отказе основной NameNode, состояние системы может быть восстановлено с помощью данных Secondary NameNode, с возможной утечкой части данных из-за отставания от мастера.

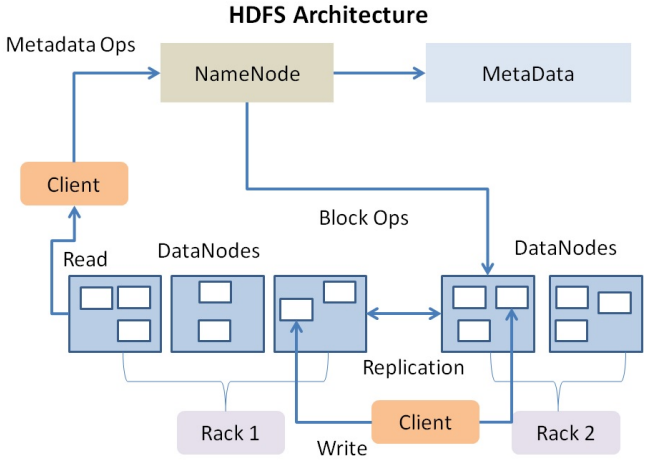

Рис. 1: Архитектура HDFS

3.1.1 Обзор архитектуры IDisk

`IDisk` - интерфейс, который содержит в себе методы для взаимодействия с файловой системой. За счет такой абстракции возможно использовать ClickHouse поверх POSIX несовместимых систем. Он отвечает за управлением хранения файлов и их метаинформации, резервацией и учетом потребляемого места.

Интерфейс предоставляет стандартные методы по работе с файловой системой:

- создание, удаление и перемещение файлов

- создание, удаление и перемещение директорий
- создание интерфейсов для чтение или записи в файл - `ReadBufferFromFile` и `WriteBufferFromFile`
- установка и возвращение информации о временной метке последних изменений
- создание жёстких ссылок на файлы и директории

Для работы с информацией об зарезервированном свободном месте используется интерфейс `IReservation`. Он предоставляет методы, которые позволяют получить или изменить информацию о размере свободного места, а также получить указатель на `IDisk`, который отвечает за диск пути резервации.

Локальная файловая система реализована с помощью библиотеки Poco. Poco - библиотека классов с открытым исходным кодом, которая хорошо интегрируется со стандартной библиотекой STL и широко используется в кодовой базе ClickHouse.

3.1.2 Обзор интерфейсов для ввода/вывода

Для побайтового ввода/выводы существуют специальные абстрактные класс ReadBuffer и WriteBuffer, которые являются улучшенной заменой std::iostream.

ReadBuffer и WriteBuffer - это буфер ограниченного размера и курсор, указывающий на текущую позицию в нем. В зависимости от реализации буфер может владеть собственной памятью и управлять ей. Для всех наследников ReadBuffer достаточно переопределить виртуальный метод nextImpl, который заполняет буфер новыми данными. Для WriteBuffer необходимо реализовать также nextImpl, но в этот раз он должен сбрасывать куда-либо данные из буфера. По умолчанию размер буфера 1048576 байт, поэтому тяжелые виртуальные методы вызываются достаточно редко. [?]

3.1.3 Обзор предлагаемого решения

Интерфейс для `IDisk` предлагается реализовать с использованием C++ клиента libhdfs3 для HDFS [14]. HDFS написана на языке Java, поэтому стандартный C клиент libhdfs основан на использовании Java Native Interface (JNI). JNI - механизм для запуска кода под управлением виртуальной машины Java, который позволяет вызывать из C/C++ кода функции написанные на Java. Его серьезным недостатком является необходимость на каждую машину дополнительно деплоить HDFS Java archive. Libhdfs3 создавался как альтернатива стандартной библиотеке, и взаимодействие с HDFS реализовано через нативные Hadoop RPC и Hadoop data transfer протоколы, что не требует работы с JNI [15].

Локально на диске для каждого пути хранится лишь небольшой файл с метаинформацией. В нем хранятся следующие поля:

- String disk_path - путь до каталога диска в локальной файловой системе
- String metadata_file_path - относительный путь до метадаты в локальной файловой системе
- size_t total_size - суммарный размер всех объектов в HDFS
- PathAndSize hdfs_object - пара из пути и размера объекта в HDFS
- UInt32 ref_count - счетчик жестких ссылок на наш путь

Создание и перемещение файлов и директорий устроено тривиальным способом, мы лишь меняем локальное расположение метаинформации, так как перекладывать ее внутри HDFS не представляется разумным.

Удаление файла производится в несколько этапов. Во-первых, надо уменьшить количество жестких ссылок на этот файл в его метаинформации. Если больше никто не ссылается на файл, то его следует полностью

удалить с HDFS и локальной системы. Иначе достаточно лишь стереть жесткую ссылку.

Изменения метаинформации файла, такие как время последнего доступа, права на чтение или запись, следует изменять лишь локально с помощью библиотеки Poco для соответсвующих файлов метаинформации.

Для записи файлов в HDFS используется WriteBuffer с собственной памятью. Сначала данными из внешнего источника заполняется собственный рабочий буфер у WriteBuffer. После этого полученные данные при вызове nextImpl сбрасываются в HDFS последовательными вызовами hdfsWrite, до тех пор пока не опустеет рабочий буфер.

Чтение файлов устроено сложнее чем запись, потому что размер файла может значительно превышать размер внутреннего буфера ReadBuffer. Интерфейс IDisk требует, чтобы возвращаемый ReadBuffer был наследником BufferWithOwnMemory<SeekableReadBuffer> в том числе поддерживал операцию смещения seek как и от позиции указателя в начале файла (SEEK_SET), так и от его текущего положения (SEEK_CUR). Но libhdfs3 из-за ограничений HDFS протоколов предоставляет только SEEK_SET операцию. Для решения этой проблемы обернем итоговый ReadBuffer в еще один класс, который будет хранить абсолютную позицию указателя в файле, а также ссылку на текущий ReadBuffer. Если результат seek находится в пределах хранимого буфера, то передвинем указатель напрямую в нем, иначе вычислим новую абсолютную позицию указателя и пересоздадим ReadBuffer с таким отступом и продолжим работу с ним.

4 Заключение

В ходе данной выпуской квалификационной работы был произведен обзор текущих решений по работе с распределенными файловыми системами, в основном с упором на выполнении сложных аналитических

запросов. Было предложено решение для использования HDFS в качестве распределенной файловой системы для ClickHouse. Для добавления такой возможности, во-первых, была подробно изучена архитектура HDFS, а также практические аспекты работы с ней, это ее развертывание, поддержка. Во-вторых, тщательно разобрано внутреннее устройство ClickHouse, особенно разделы, которые относятся к работе с вводом и выводом, а также интерфейсы, отвечающие за взаимодействие с интерфейсом файловой системы. В результате данной работы появляется возможность для многочисленных пользователей ClickHouse, так и продуктов Hadoop использовать преимущества обеих систем в задачах по хранению и обработке данных.

5 Список литературы

Список литературы

[1] Ghemawat S., Gobioff H., Leung S. T. The Google file system //Proceedings of the nineteenth ACM symposium on Operating systems principles. – 2003. – С. 29-43.

[2] Apache Hadoop. http://hadoop.apache.org/. Accessed 20 May 2020.

[3] Shvachko K. et al. The hadoop distributed file system //2010 IEEE 26th symposium on mass storage systems and technologies (MSST). – Ieee, 2010. – С. 1-10.

[4] Performance Comparison of Database Management Systems. https://clickhouse.tech/benchmark/dbms/. Accessed 20 May 2020.

[5] Bharath Kumar Reddy Vangoor, Vasily Tarasov, and Erez Zadok. To FUSE or Not to FUSE: Performance of User-space File Systems. In Proceedings of the 15th Usenix Conference on File and Storage Technologies, FAST'17, Santa clara, CA, USA, 2017.

[6] Sangaline E., Lauret J. Experience, use, and performance measurement of the Hadoop File System in a typical nuclear physics analysis workflow //Journal of Physics: Conference Series. – IOP Publishing, 2014. – Т. 523. – №. 1. – С. 012006.

[7] White, Tom. Hadoop: The Definitive Guide. Fourth edition, O'Reilly, 2015.

[8] Distinctive Features - ClickHouse Documentation. https://clickhouse.tech/docs/en/introduction/distinctive-features/. Accessed 20 May 2020.

[9] Haines, Steven. Introduction to HBase, the NoSQL Database for Hadoop. 2014. informIT database, https://www.informit.com/articles/article.aspx?p=2253412.

[10] "Amazon Web Services (AWS) - Cloud Computing Services." Amazon Web Services, Inc., https://aws.amazon.com/. Accessed 20 May 2020.

[11] "Amazon DynamoDB - Overview." Amazon Web Services, Inc., https://aws.amazon.com/dynamodb/. Accessed 20 May 2020.

[12] "Amazon FSx for Lustre | File Storage | AWS." Amazon Web Services, Inc., https://aws.amazon.com/fsx/lustre/. Accessed 20 May 2020.

[13] Overview of ClickHouse Architecture - ClickHouse Documentation. https://clickhouse.tech/docs/en/development/architecture/. Accessed 20 May 2020.

[14] Rosen, Brett. Bdrosen96/Libhdfs3. 2016. 2020. GitHub, https://github.com/bdrosen96/libhdfs3.

[15] Apache Hadoop 3.2.1 – C API Libhdfs. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/LibHdfs.html. Accessed 20 May 2020.