Обзор Apache Accumulo.

Apache Accumulo — это отсортированное распределенное хранилище ключ-значение, основанное на дизайне Google BigTable. Accumulo построен на основе Apache Hadoop, Zookeeper и Thrift. Он включает в себя несколько новых улучшений дизайна BigTable в виде меток доступа на уровне ячеек и механизма программирования на стороне сервера, который может изменять пары ключ/значение на различных этапах процесса управления данными.

Согласно рейтингу DB-Engines, Accumulo является третьим по популярности хранилищем столбцов NoSQL после Apache Cassandra и HBase и 67-м по популярности механизмом базы данных любого типа (полным) по состоянию на 2018 год.

**История развития**

Google опубликовал дизайн BigTable в 2006 году. Несколько других проектов с открытым исходным кодом реализовали аспекты этого дизайна, включая HBase, CloudStore и Cassandra. Accumulo начала свою разработку в 2008 году.

Данная СУБД предоставляет гибкое, высокопроизводительное распределенное хранилище ключей/значений, с детализированными метками доступа. Сообщества, заинтересованные в таком проекте, — это правительство, здравоохранение и другие отрасли, где конфиденциальность является проблемой.

Accumulo находится в разработке с весны 2008 года. Основная кодовая база состоит из 200 000 строк кода (в основном Java) и сотен страниц документации. Есть также несколько проектов, созданных поверх Accumulo, которые могут быть добавлены к его вкладу в будущем. К ним относятся поддержка Hive, Matlab, YCSB и обработка графов.

**Дополнительные программы для запуска Accumulo**

Для запуска Accumulo понадобятся следующие программы:

* Zookeeper
* Hadoop
* Thrift
* Maven

Для простого запуска можно использовать:

* [Uno](https://github.com/apache/fluo-uno) устанавливает Accumulo на одной машине для тестирования
* [Muchos](https://github.com/apache/fluo-muchos) настраивает Accumulo в кластере (опционально запускается в Amazon EC2 и Microsoft Azure VM)

**Модель данных**

Accumulo предоставляет более богатую модель данных, чем простые хранилища ключей и значений, но не является полностью реляционной базой данных. Данные представлены в виде пар ключ-значение, где ключ и значение состоят из следующих элементов:

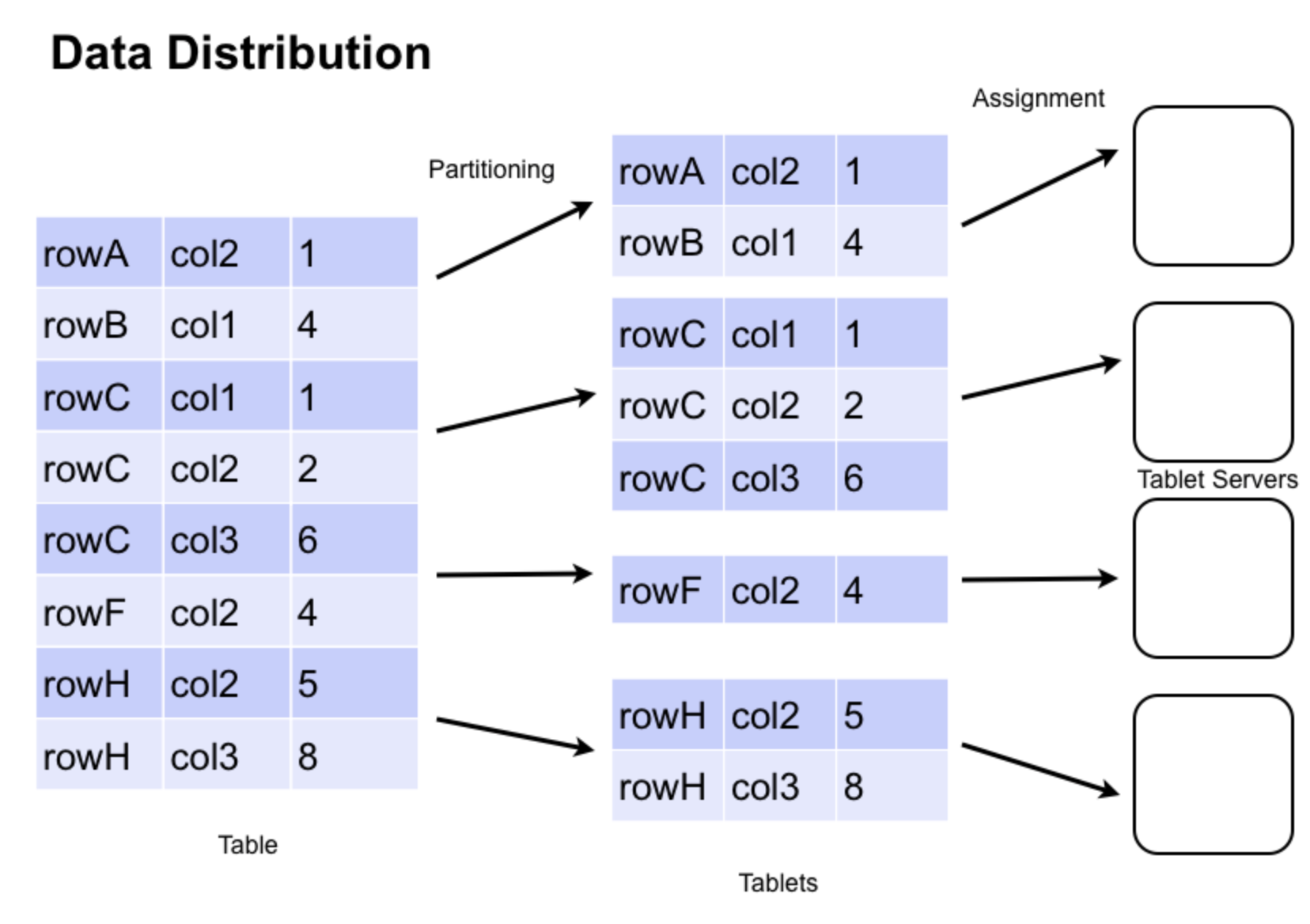


Все элементы ключа и значения представлены в виде массивов байтов, кроме метки времени, которая является Long. Accumulo сортирует ключи по элементам и лексикографически в порядке возрастания. Временные метки сортируются в порядке убывания, так что более поздние версии того же ключа появляются первыми при последовательном сканировании. Таблицы состоят из набора отсортированных пар ключ-значение.

Таблицы Accumulo сортируются по Row ID. Поиск по Row ID может быть выполнен быстро, путем двоичного поиска сначала по всем планшетам, а затем внутри одного планшета.

**Разделение данных**

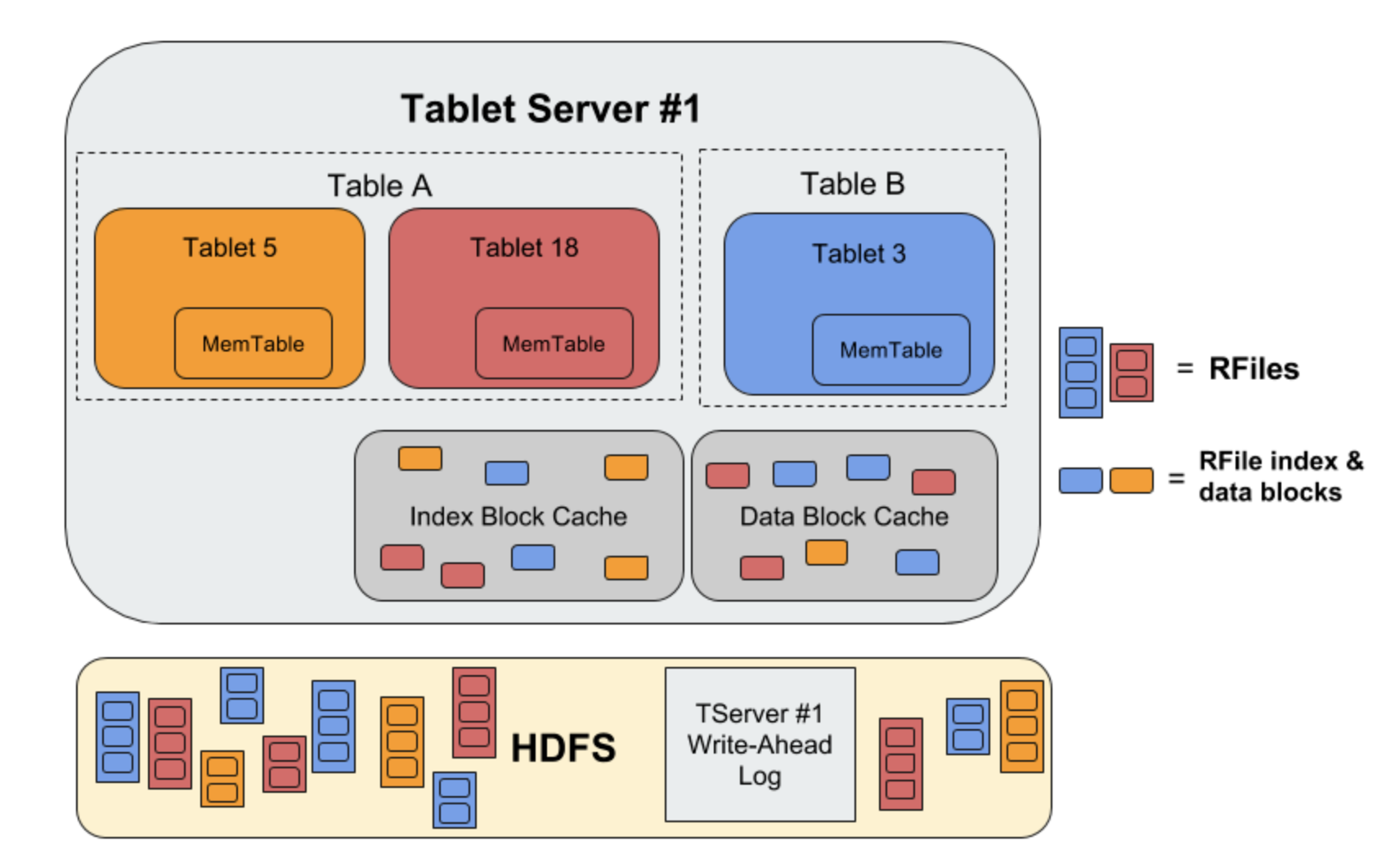
Accumulo хранит данные в таблицах, которые разделены на планшеты. Таблицы разделены по границам строк(raw), так что все столбцы и значения для конкретного Row Id находятся вместе в одной таблице. Мастер одновременно назначает планшеты одному TabletServer. Это позволяет выполнять транзакции на уровне строк без использования распределенной блокировки или какого-либо другого сложного механизма синхронизации. Когда клиенты вставляют и запрашивают данные, а также когда компьютеры добавляются и удаляются из кластера, мастер переносит планшеты, чтобы гарантировать, что они останутся доступными, а нагрузка по приему и запросу будет сбалансирована в кластере.



**Tablet Server**

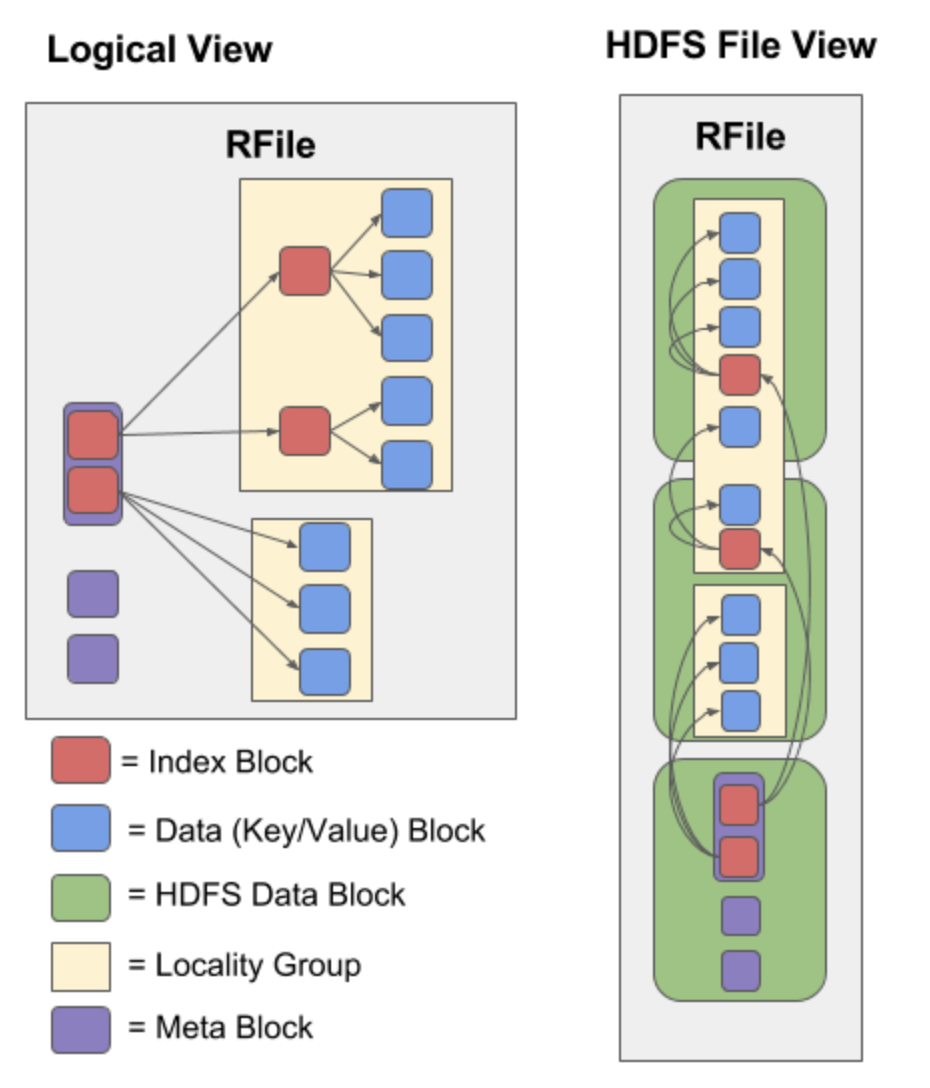
Когда запись поступает на TabletServer, она записывается в Write-Ahead Log, а затем вставляется в отсортированную структуру данных в памяти, называемую MemTable. Когда MemTable достигает определенного размера, TabletServer записывает отсортированные пары ключ-значение в файл в HDFS, называемый RFile). Этот процесс называется незначительным уплотнением. Затем создается новая таблица MemTable, и факт уплотнения записывается в Write-Ahead Log.

Когда на TabletServer поступает запрос на чтение данных, TabletServer выполняет бинарный поиск в MemTable, а также в блоках индекса, связанных с каждым RFile, чтобы найти соответствующие значения. Если клиенты выполняют сканирование, несколько пар ключ-значение возвращаются клиенту по порядку из MemTable и блоков данных RFiles путем выполнения sorted merge по мере их чтения. Если для таблицы включено кэширование, любой индекс или блок данных сохраняются в кеше блоков, чтобы ускорить будущие сканирования.



**RFile**

RFile (Relative Key File) — это файл, который содержит отсортированные пары «ключ-значение» Accumulo. Файл записывается в HDFS планшетными серверами во время незначительного сжатия. RFiles организованы с использованием метода индексного последовательного доступа (ISAM). RFiles состоят из блока данных (ключ/значение), индексных блоков (которые используются для поиска блока данных) и метаблоков (которые содержат метаданные для фильтров Блума и сводной статистики). Данные в файле RFile разделяются по группам местоположения. На приведенной ниже диаграмме показано логическое представление и представление файла HDFS для файла RFile.



**Уплотнение данных и разбиение**

Чтобы управлять количеством файлов на планшете, TabletServer периодически выполняет основные уплотнения файлов внутри планшета, при которых некоторый набор RFiles объединяется в один файл. Предыдущие файлы в конечном итоге будут удалены сборщиком мусора. Это также дает возможность безвозвратно удалить удаленные пары ключ-значение, исключив пары ключ-значение, перекрытые записью удаления при создании нового файла.

Когда создается таблица, она имеет один планшет. По мере роста таблицы ее первоначальный планшет в конечном итоге разделяется на два планшета. Вполне вероятно, что один из этих планшетов переместится на другой сервер планшетов. По мере роста таблицы ее планшеты будут продолжать разделяться и перемещаться. Решение об автоматическом разделении планшета основывается на размере файлов планшета. Порог размера, при котором планшет разделяется, настраивается для каждой таблицы. В дополнение к автоматическому разделению пользователь может вручную добавить точки разделения в таблицу для создания новых таблиц. Ручное разбиение новой таблицы может распараллелить чтение и запись, обеспечивая лучшую начальную производительность без ожидания автоматического разбиения. По мере удаления данных из таблицы планшеты могут уменьшаться. Со временем это может привести к появлению маленьких или пустых планшетов. Для решения этой проблемы в Accumulo 1.4 было введено их объединение.

**Компоненты**

Экземпляр Accumulo включает в себя множество TabletServers, один процесс сборщика мусора, один главный сервер и множество клиентов.

**TabletServer**

Сервер TabletServer управляет некоторым подмножеством всех планшетов (разделов таблиц). Это включает в себя прием записей от клиентов, сохранение записей в журнале записи, сортировку новых пар ключ-значение в памяти, периодическую очистку отсортированных пар ключ-значение в новые файлы в HDFS, а также ответ на чтение от клиентов, формируя отсортированное представление слияния всех ключей и значений из всех созданных файлов и отсортированного хранилища в памяти. Серверы TabletServers также выполняют восстановление планшетов, которые ранее находились на сервере, вышедшем из строя, повторно применяя все записи, найденные в журнале write-ahead, к планшетам.

**Сборщик мусора**

Процессы Accumulo совместно используют файлы, хранящиеся в HDFS. Периодически сборщик мусора будет определять файлы, которые больше не нужны ни одному процессу, и удалять их.

**Мастер**

Мастер Accumulo отвечает за обнаружение и реагирование на отказ TabletServer. Он пытается сбалансировать нагрузку на TabletServer, тщательно распределяя планшеты и давая указания TabletServers разгрузить планшеты, когда это необходимо. Мастер обеспечивает назначение всех планшетов на один TabletServer, а также обрабатывает запросы клиентов на создание, изменение и удаление таблиц. Мастер также координирует запуск, завершение работы и восстановление изменений в журналах с опережающей записью при выходе из строя серверов Tablet.

**Monitor**

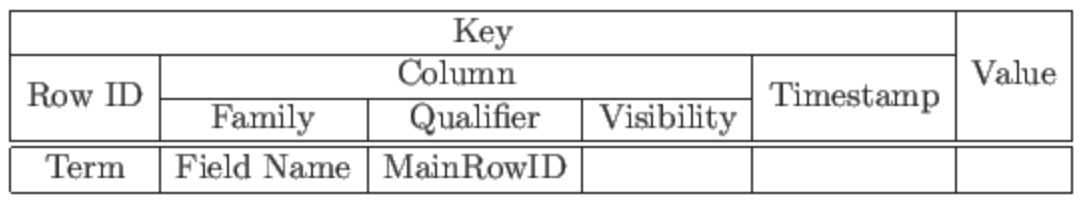
Монитор Accumulo - это веб-приложение, которое предоставляет обширную информацию о состоянии экземпляра. Монитор отображает графики и таблицы, содержащие информацию о скорости чтения/записи, количестве обращений/пропусков к кэшу, а также информацию о таблице Accumulo, такую как скорость сканирования и активные/очередные сжатия. Кроме того, Монитор всегда должен быть первой точкой входа при попытке отладки проблемы Accumulo, поскольку он показывает проблемы высокого уровня в дополнение к агрегированным ошибкам со всех узлов кластера.

**Клиент**

Accumulo имеет клиентскую библиотеку, которая может быть использована для написания приложений, записывающих и считывающих данные в/из Accumulo.

**Индексация данных**

Как было сказано выше, данные можно считать проиндексированными по Row ID. Для поддержки поиска по более чем одному атрибуту объекта могут быть построены дополнительные индексы. Однако, поскольку таблицы Accumulo могут поддерживать любое количество столбцов без их предварительного указания, часто бывает достаточно одного дополнительного индекса для поддержки поиска записей в основной таблице. Здесь индекс имеет в качестве row ID value или term из основной таблицы, семейства столбцов одинаковы, а квалификатор столбца индексной таблицы содержит row ID из основной таблицы.



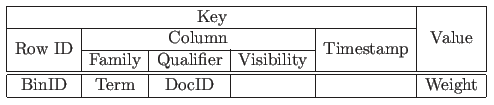
RowID хранится в классификаторе столбца, а не в Value, чтобы иметь более одного rowID, связанного с определенным Term в индексе. Если бы мы хранили его в Value, мы бы увидели только одну из строк, в которой появляется значение, поскольку Accumulo по умолчанию настроен на возврат одного последнего значения, связанного с ключом. Затем поиск может быть выполнен путем сканирования индексной таблицы на предмет вхождения нужных значений в указанные столбцы, что возвращает список идентификаторов строк из основной таблицы. Затем эти данные могут быть использованы для извлечения каждой совпадающей записи полностью или подмножества столбцов из основной таблицы. Для поддержки эффективного поиска нескольких идентификаторов строк из одной таблицы клиентская библиотека Accumulo предоставляет BatchScanner. Пользователи задают набор диапазонов BatchScanner, который выполняет поиск в нескольких потоках на нескольких серверах и возвращает итератор всех полученных строк. Возвращаемые строки НЕ сортируются в порядке, как в случае с базовым интерфейсом Scanner.

**Индексация данных по множеству условий**

Использование простого индекса, как описано выше, хорошо работает при поиске записей, соответствующих одному из набора заданных критериев. При поиске записей, которые одновременно соответствуют более чем одному критерию, например, при поиске документов, содержащих все слова «the», «white» и «house», возникает несколько проблем.

Во-первых, набор всех записей, соответствующих одному из условий поиска, должен быть отправлен клиенту, что влечет за собой большой сетевой трафик. Вторая проблема заключается в том, что клиент несет ответственность за выполнение пересечения множества возвращенных наборов записей, чтобы исключить все записи, кроме тех, которые соответствуют всем условиям поиска. Память клиента может быть легко перегружена во время этой операции.

По этим причинам Accumulo включает поддержку схемы, известной как сегментированное индексирование, в которой эти операции с наборами могут выполняться на TabletServers, а решения о том, какие записи включать в набор результатов, могут приниматься без сетевого трафика. Это достигается путем разделения записей на ячейки, каждая из которых находится не более чем на одном TabletServer, а затем создания индекса терминов для каждой записи в каждой ячейке.



Документы или записи сопоставляются с корзинами определяемым пользователем приложением загрузки. Сохраняя BinID в качестве RowID, мы гарантируем, что вся информация для конкретной корзины содержится в одном планшете и размещается на одном TabletServer, поскольку Accumulo никогда не разделяет ряды между планшетами. Хранение Terms в виде семейств столбцов обеспечивает быстрый поиск всех документов в этой подборке, содержащих данный Term. Наконец, мы выполняем операции на TabletServer с помощью специального итератора, называемого Intersecting Iterator. Поскольку документы разделены на множество корзин, поиск всех документов должен выполняться во всех корзинах. Мы можем использовать BatchScanner для параллельного сканирования всех корзин.

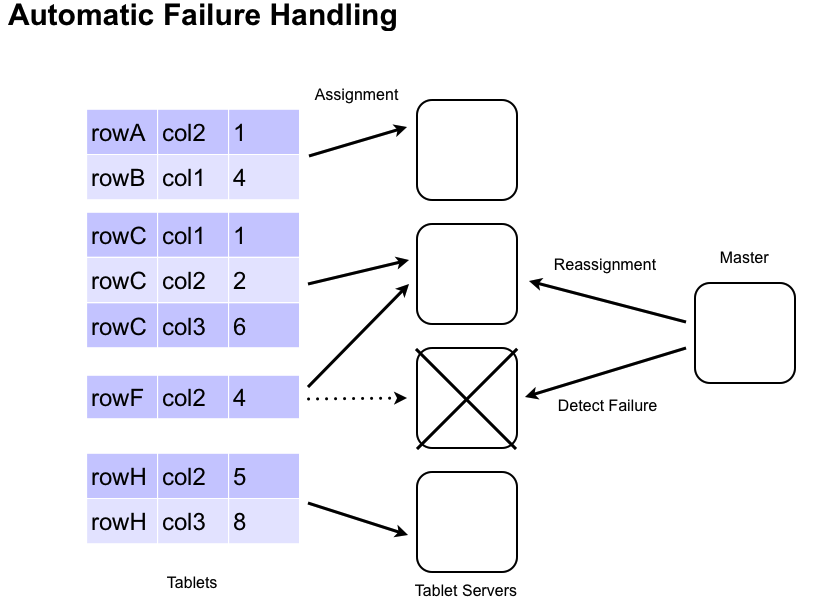
**Транзакции (FATE)**

Транзакции в Accumulo реализованы в виде FATE (**Fa**ult **T**olerant **E**xecutor). FATE это платформа для выполнения операций отказоустойчивым образом. До FATE, если главный процесс умирал в середине создания таблицы, он мог оставить систему в несогласованном состоянии. С этой новой структурой, если мастер умирает в середине создания таблицы, он продолжит работу при перезапуске. Кроме того, клиент, запрашивающий операцию создания таблицы, никогда не узнает, что что-то произошло. Платформа сериализует работу в Zookeeper, прежде чем пытаться выполнить работу. Клиенты начинают транзакцию FATE, заполняют ее работой и затем ждут ее завершения. Большинство операций с таблицами выполняются с использованием этого фреймворка. Постоянные блокировки чтения-записи для каждой таблицы создаются в Zookeeper для синхронизации операций при сбоях процессов.

FATE состоит из двух основных компонентов: повторяемой постоянной операции (REPO), уровня хранения для REPO и системы выполнения для запуска REPO. Accumulo использует ZooKeeper в качестве уровня хранения для FATE, а Accumulo Master выступает в качестве исполнительной системы для запуска REPO. Важной характеристикой REPO является то, что они реализованы идемпотентно: каждая операция должна иметь возможность отменить или воспроизвести частичное выполнение самой себя. Требование реализации операции для поддержки этого функционала значительно упрощает выполнение этих операций. Это свойство также является гарантией безопасности в условиях отказа.

**Методы восстановления после падения.**

Если Tablet Server выходит из строя, Master обнаруживает это и автоматически переназначает планшеты, назначенные от отказавшего сервера, другим серверам. Любые пары ключ-значение, которые находились в памяти на момент сбоя TabletServer, автоматически повторно применяются из Write Ahead Log (WAL), чтобы предотвратить потерю данных. Tablet Servers записывают свои WAL непосредственно в HDFS, поэтому журналы доступны для всех планшетных серверов для восстановления. Чтобы сделать процесс восстановления эффективным, обновления в журнале сгруппированы по планшетам. Tablet Servers могут быстро применять изменения из отсортированных журналов, предназначенных для планшетов, которые им сейчас назначены.

****

**Шардирование**

В Accumulo данные автоматически шардируются, так как происходит их сохранение на разные планшеты.

**Безопасность и Шифрование**

Accumulo имеет следующие функции безопасности:

* Только аутентифицированные пользователи могут получить доступ к Accumulo.
  + Kerberos может быть включен для замены стандартной аутентификации Accumulo на основе пароля.
* Пользователи могут выполнять действия, только если им дано разрешение.
* Пользователи могут просматривать только те маркированные данные, на которые у них есть разрешение.
* Данные могут быть зашифрованы на диске и over-the-wire.

После аутентификации пользователя с помощью Authenticator, он получает доступ к другим действиям в Accumulo. Все действия в Accumulo контролируются ACL, проверка ACL обрабатывается Permission Handler. Permission Handler управляет всеми разрешениями, которые разделены на уровне системы и на уровне таблицы. Если пользователь выполняет действие, требующее авторизации, Authorizor запрашивается, чтобы определить, какие авторизации есть у пользователя.

**Выполнение и планирование запросов.**

В Accumulo отсутствует собственный план запросов. Разработчику приложения нужно самому его реализовать, опираясь на специфику данных и запросы. Запросы передаются через Accumulo Client. Поддержки SQL нет. Результаты сканирования отправляются клиенту партиями автоматически определяемого размера. Пакетная обработка может привести к задержке в несколько секунд, прежде чем клиент получит первый результат. Более того, в Accumulo API нет возможности, подобной оператору SQL LIMIT. Таким образом, без пакетных запросов пользователю может потребоваться дождаться буферизации большого количества результатов на планшетных серверах, даже если требуется лишь небольшое количество результатов.

**Работа с Accumulo**

Есть 2 способа взаимодействия с Accumulo: Accumulo Client и Accumulo Shell.

1. Accumulo Shell

Подключение выполняется с помощью команды:

accumulo shell -u [username]

Стандартные команды:



1. Accumulo Client -  можно подсоединиться к БД используя maven библиотеку для Java org.apache.accumulo:accumulo-core:2.0.1 Для Альтернативой может послужить библиотека pyaccumulo - для работы с Accumulo на Python.

**OLAP с Accumulo.**

Accumulo не реализует реляционную алгебру. Accumulo предоставляет гарантии ACID, но на более ограниченной основе. Единственными транзакциями, разрешенными Accumulo, являются вставки, удаления или обновления нескольких значений в одной строке. Эти транзакции являются атомарными, последовательными, изолированными и устойчивыми. Но набор обновлений нескольких строк в одной таблице или строк в разных таблицах не дает этих гарантий.

Возможность эффективно выполнять single-row update позволяет использовать Accumulo для OLAP. Стоит отметить однако, что для этого данные требуется денормализовать.

**Community и участие в разработке**

Accumulo поддерживается Apache Accumulo community. Связь с community через [Slack](https://s.apache.org/slack-invite).

Любой может поучаствовать в разработке Accumulo: выбрать одну из нерешенных [проблем](https://github.com/apache/accumulo/issues?q=is%3Aissue+is%3Aopen+label%3A%22good+first+issue%22), посмотреть открытые пулл реквесты, обновить документацию, сообщить о багах или протестировать новые релизы.

**Ссылки**

* <https://accumulo.apache.org/docs/2.x/getting-started/design>
* <https://cwiki.apache.org/confluence/display/incubator/AccumuloProposal>
* <https://www.oreilly.com/library/view/accumulo/9781491947098/ch01.html>