KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 4 урок > Apache HBase. Масштабируемая колоночная БД.

> Оглавление

> Введение: как хранятся данные в HBase

Нереляционные базы данных делятся на четыре типа:

- **Key-Value хранилища**, они хранят базы данных в виде ключа и значения. В каждой её ячейке хранятся данные произвольного типа, а каждому значению присвоен уникальный ключ, по которому это значение можно найти.
- **Колоночные БД**, куда относится и **Apache HBase**. Данные группируются для хранения не по строкам, а по столбцам. Если строго придерживаться позиционирования, то **HBase,** это БД ориентированная на семейство колонок.
- **Документоориентированные БД**, в них данные хранятся в виде иерархических структур (документов) с произвольным набором полей и их значений. Документы объединяются в коллекции. Яркий представитель - MongoDB.
- **Гра́фовые базы данных** — разновидность баз данных с реализацией сетевой модели в виде графа и его обобщений. Это **Neo4j** или InfiniteGraph.

Наиболее быстрыми и удобными для доступа к записи в БД по ключу являются хранилища типа Key-Value (ключ-значение). Для таких БД характерна сложность алгоритма доступа к элементу равная O(1), т.е. условно-константная.

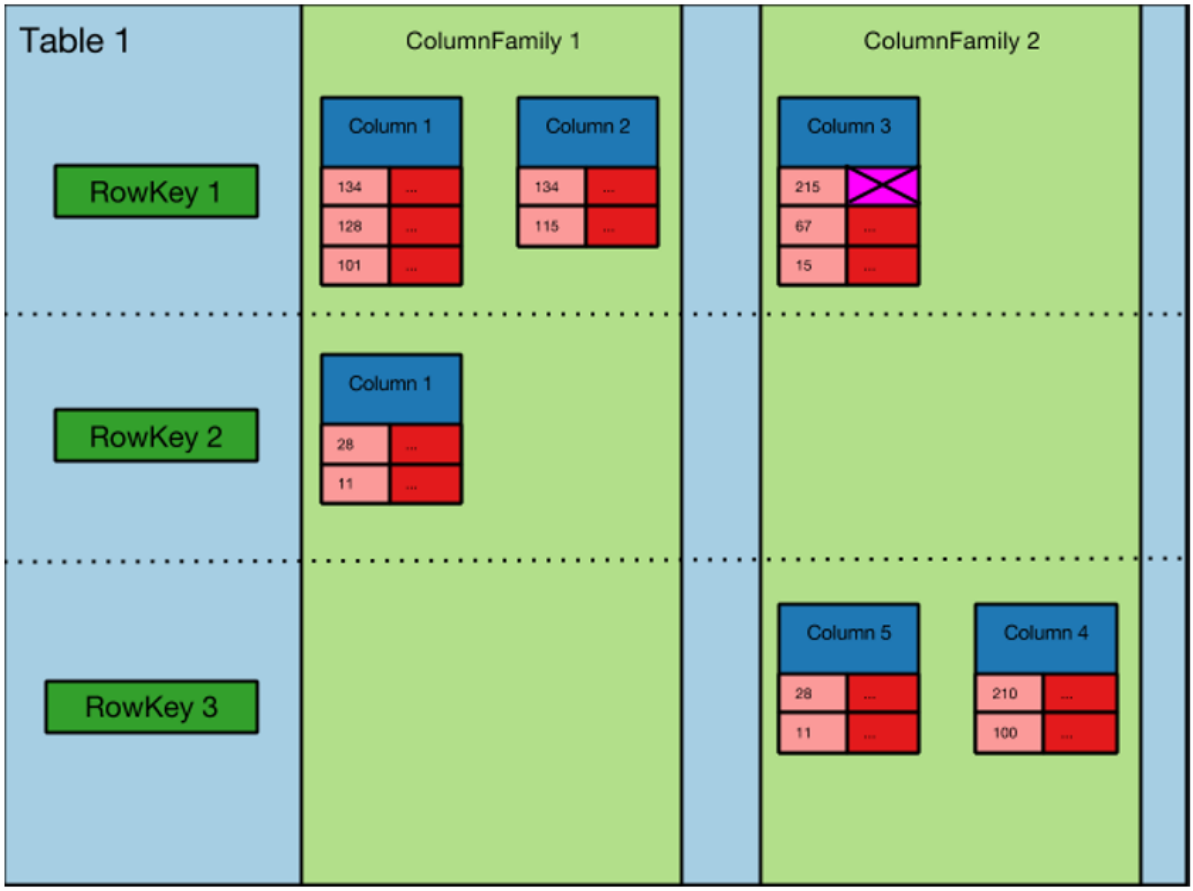
Но для таких БД неудобно создавать batch-обработки, т.к. они созданы для максимально быстрого доступа к единичным записям. Для таких обработок больше подходят хранилища на HDFS, например, Apache Hive.

Особенность **Apache HBase:**

- эффективный способ доступа по ключу;
- поддерживает Scan (доступ к диапазонам ключей).

> Модель данных

- Каждая запись в таблице проиндексирована при помощи первичного ключа, который называется RowKey.
- Для каждого ключа RowKey может храниться неограниченное количество атрибутов, размещаемых в колонках (Column).
- Колонки группируются в группы колонок (ColumnFamily). Это контейнер для полей, который мы можем определять в момент вставки. На момент создания таблицы колонки не определяются, а определяются только ColumnFamily. Они объединены или бизнес-логикой, или близостью данных, которые мы хотим хранить вместе. Например, персональная информация и адреса.
- Колонки не определяются схемой и могут быть добавлены «на лету». Внутри ColumnFamily мы можем создавать колонки и указывать их (колонки) только в момент вставки в БД. Это дает гибкость в хранении и использовании данных, избегая дополнительных накладных расходов.
- Таблицы являются разреженными. Пустые значения не требуют дополнительного места для хранения.



- В HBase нет такого понятия — "тип данных", все по умолчанию определяется как массив byte. Мы сами должны знать какие данные и в каком виде мы записываем в таблицу.
- Для каждого атрибута может храниться несколько версий. Каждая версия имеет свой Timestamp. На уровне ColumnFamily мы можем настраивать дополнительную атрибутику по управлению количеством версий полей.
- Записанное в Hbase значение не может быть изменено. Вместо этого необходимо добавить новую версию с более свежим timestamp’ом.
- Есть возможность выставлять TTL (диапазон времени), который позволяет нам автоматически удалять устаревшие записи. Стоит использовать с осторожностью.
- Для удаления записи помечаются специальным маркером о совершении данной операции. Удаление данных из таблицы в HBase не происходит физически, а производится на другой стадии.
- Hbase является распределенной системой. Гарантируется, что данные соответствующие одному значению и группе колонок хранятся вместе.

Краткая модель данных:

- Распределенное, многомерное, разреженное, сортированное отображение
- (Table, RowKey, ColumnFamily, Column, Timestamp) -> value

> Иерархия хранения данных

- Table - SortedMap `<RowKey, Row>` Таблица представляет из себя сортированный массив, который состоит из ключа (RowKey) и значения (Row - сама строка).
- Row - List `<ColumnFamily>` Строка является списком контейнеров под колонки (ColumnFamily).
- ColumnFamily - SortedMap `<Column, List<Entry>>` Семейство колонок представляет из себя сортированный массив, состоящий из ключа (колонки) и значения (список значений колонки).
- Entry - Tuple `<Timestamp, Value>` Значения поля является парой timestamp-а и самого значения.

Такая чёткая иерархия позволяет точно производить навигацию по структуре данных без дополнительных накладных расходов на считывание всей строки и отсечения ненужных полей (как это происходит в обычных реляционных БД).

> Базовые операции и HBase shell

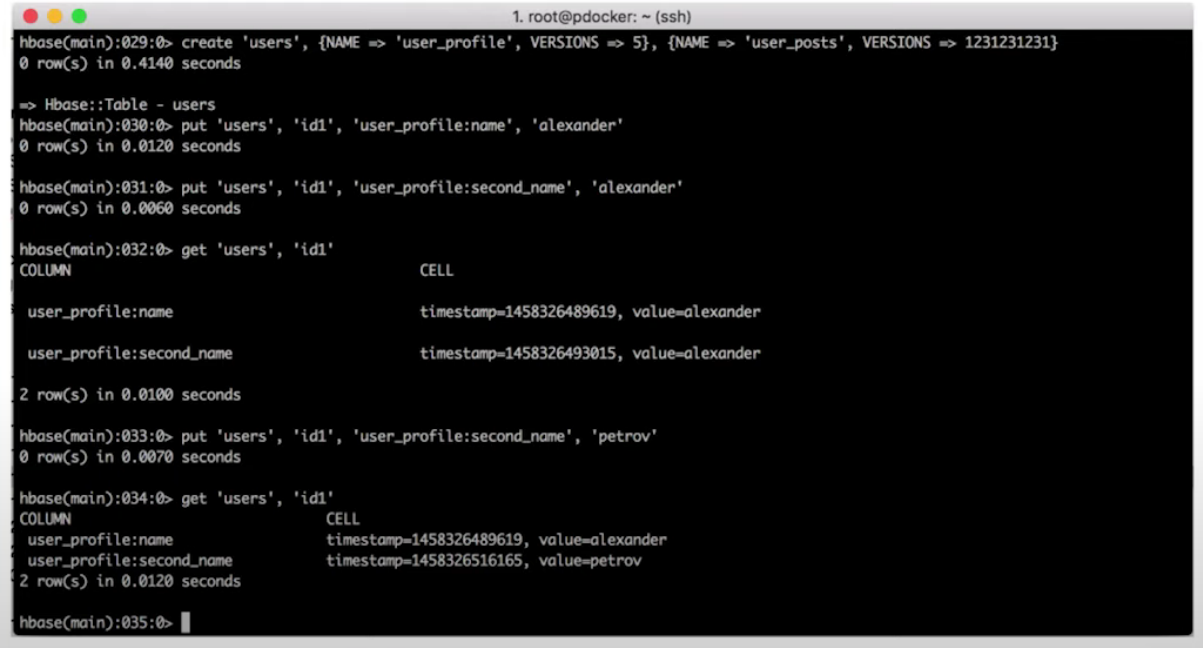
> Базовые операции

- **Get** - получить все атрибуты для заданного ключа.
- **Put** - добавить новую запись в таблицу (если записи не было) или обновить (если запись была). Timestamp этой записи может быть задан вручную, иначе он будет установлен автоматически, как текущее время.
- **Scan** - позволяет итерироваться по диапазону ключа. Можно указать запись с которой начинается чтение, запись до которой нужно прочитать, количество записей которые необходимо считать, **Column Family** из которой будет производиться чтение и максимальное количество версий для каждой записи.

- **Delete** - позволяет пометить запись как удаленную. **HBase** не удаляет данные сразу, а ставит маркер "могильный камень" - tombstone. Физическое удаление произойдет на **Major Compaction.**

> Утилита HBase shell

HBase shell – командная оболочка HBase. Доступна сразу после установки **Hbase** на любой ноде кластера. **Hbase shell** представляет из себя jruby-консоль со встроенной поддержкой всех основных операций по работе с **Hbase** (создание БД, таблиц, добавление данных, сканы и т.д.). Разработчики как правило не используют HBase shell в работе, в отличие от администраторов.



На данном примере создаётся таблица `users`, и в фигурных скобках указываются ColumnFamily. Создаются 2 ColumnFamily: `user_profile` (храним только последние 5 изменений) и `user_posts` (храним довольно большое количество изменений).
Далее вносится первые записи в таблицу `users` с помощью операции `put`. Сначала указывается RowKey, затем ColumnFamily с названием колонки и далее само значение.
Т.к. HBase не поддерживает операцию изменения внесённых данных, поэтому для изменения значения выполняем ещё одну операцию `put` в том же поле с тем же значением RowKey.
Операция `get` без дополнительных параметров покажет актуальное состояние значений указанных полей.

> Архитектура HBase

HMaster – главный сервер, который мониторит, управляет и координирует процессы в кластере. Управляет распределением регионов по Region Server'ам, ведет реестр регионов, управляет запусками регулярных задач. В ранних версиях был один, далее их стало несколько.

Zookeeper – инструмент для поддержки информации о конфигурации, системных настройках и обеспечения синхронизации. Хранит информацию о расположении метатаблицы в HBase, в которой описана вся топология кластера, а также настройки подключения к HMaster-у. Является точкой входа для клиента (приложения).

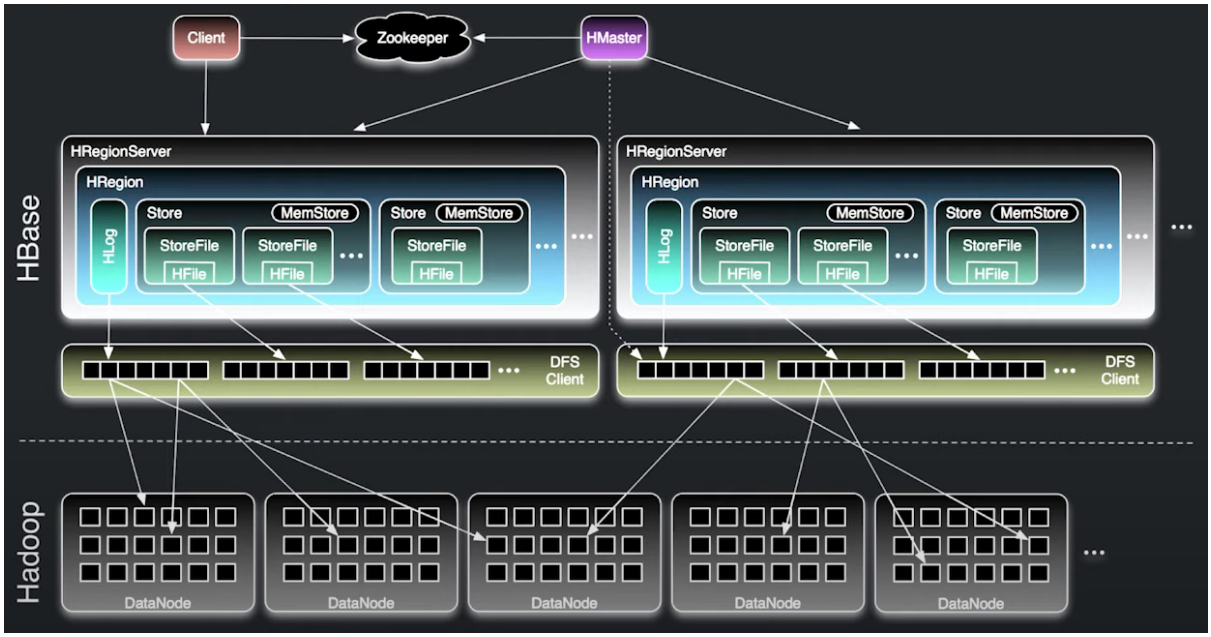
Т.е. клиент сперва обращается к Zookeeper-у, чтобы узнать топологию и как подключаться к кластеру HBase.

Все операции клиент производит не через HMaster, а напрямую с серверами, которые называются Region Server. Region Server представляет из себя либо сервер, либо виртуальную машину, которая управляет данными, которые хранятся в HBase. Обслуживает, хранит и управляет одним или несколькими регионами. Регион — это диапазон записей в таблице соответствующих определенному диапазону подряд идущих RowKey.

Каждый регион содержит:

- Write Ahead Log (WAL) или HLog — последовательный append-only лог произведённых операций. Так как данные при записи попадают в Memstore, существует некоторый риск потери данных из-за сбоя. Для того чтобы этого не произошло все операции перед осуществлением манипуляций попадают в специальный лог-файл. Это позволяет восстановить данные после любого сбоя.
- MemStore — буфер на запись, хранилище данных, которые поступают в таблицу. Данные в MemStore накапливаются и хранятся некоторое время. При наполнении MemStore до некоторого критического значения данные записываются в новый файл HFile, а MemStore очищается.

HBase не работает с конкретными датанодами на HDFS, но HBase работает с клиентом HDFS, и не управляет тем, где физически хранятся данные.

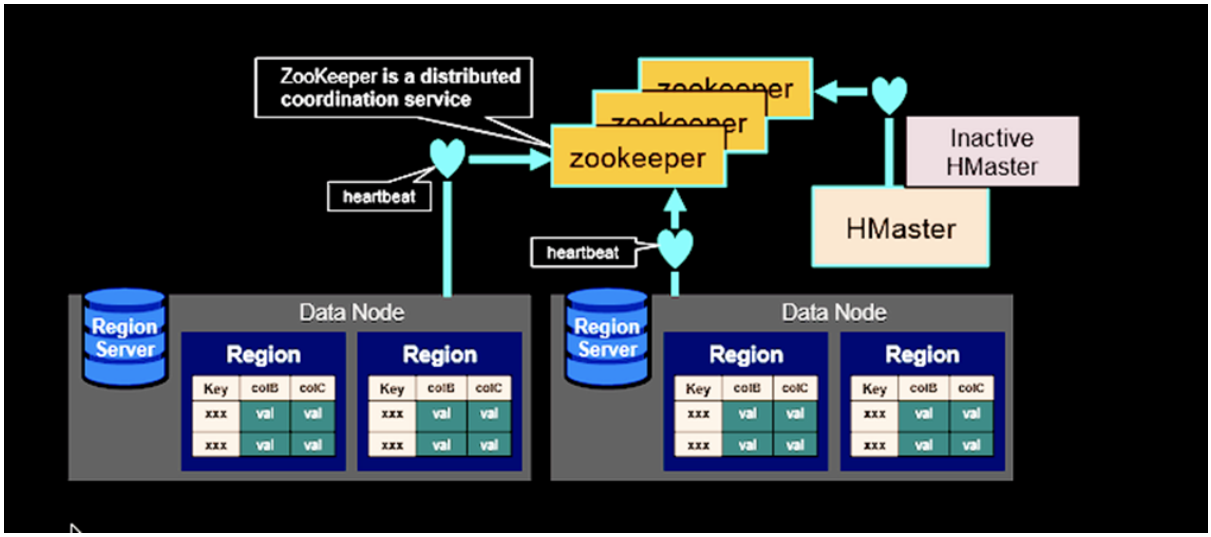


Архитектура HBase

> Zookeeper и HBase Master

> Zookeeper

- Специальный сервис, используемый для координации сервисов. Представляет из себя очень «живучую» key-value базу данных, поддерживающая механизмы Pub/Sub (публикация и подписка на изменения по ключу).
- Каждый Region Server и HMaster Server периодически отправляют heartbeat в Zookeeper и он проверяется статус. В случае потери инициирует сообщения о необходимости восстановления.
- Активный HMaster отравляет сообщения в Zookeeper, inactive HMaster следит за активным. В случае падения сам становится активным.
- Если Region Server не отправляет уведомления HMaster запускает процесс восстановления.
- Zookeeper обслуживает путь до .META таблицы.



На схеме имеется отказоустойчивый кластер из 3 нод zookeeper-а, в который все компоненты кластера направляют сообщения о своём состоянии (heartbeat). Если не пришло такое сообщение от активного HMaster-а, то происходит переключение на standby HMaster. Если теряются heartbeat-ы от region server-а, то запускается его восстановление.

> HBase Master

- Главный процесс в HBase, ведет реестр всех активных регионов.
- Обслуживает DDL операции (create and delete tables), распределяет регионы по всем серверам.
- Управляет и координирует работу Region Server (как NameNode управляет DataNode в HDFS).
- Назначает Region в Region Server при восстановлении и load balancing (перебалансировка Region-ов между оставшимися RegionServer).
- Мониторит Region Server (используя Zookeeper) и производит восстановление Region Server в случае падения.

> HBase Region Server

Обслуживает один или несколько Region.

Region

- отсортированная часть строк таблицы.

WAL (HLog)

- журнал записей. Любая операция должна быть записана в WAL перед записью на диск,
- append only.

MemStore

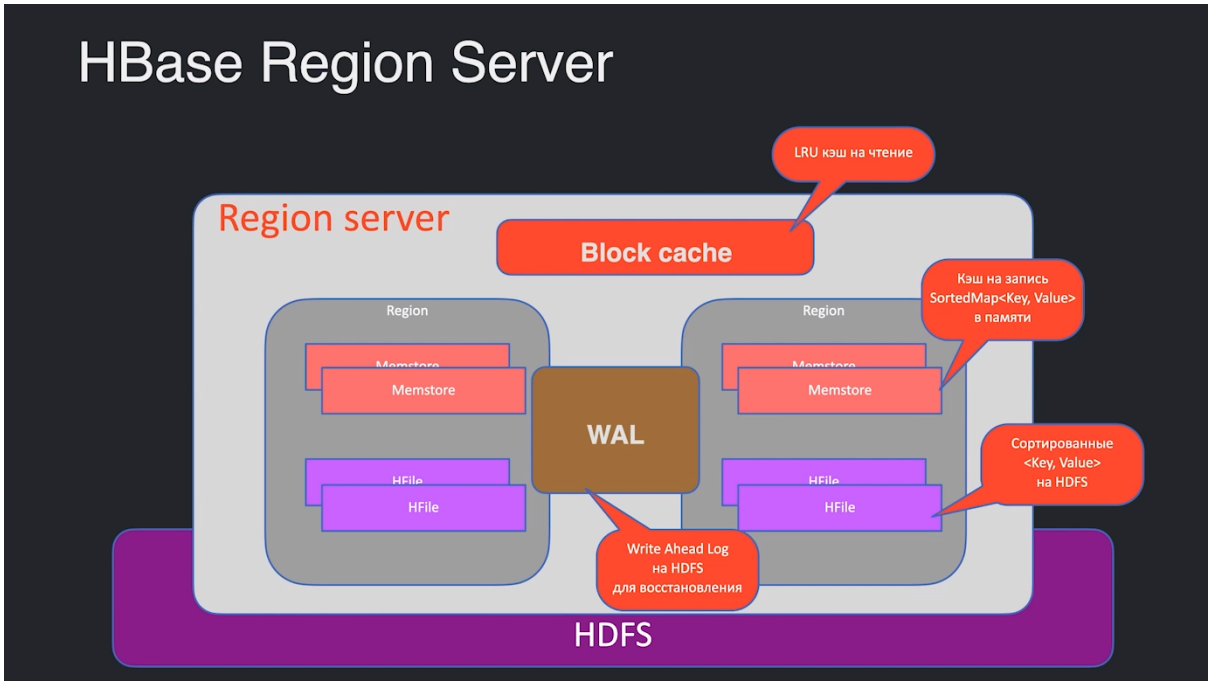
- кэш на запись. В MemStore хранятся которые еще не были записаны на диск (HFile),
- записи сортируются перед записью на диск,
- 1 MemStore приходится на пару (Column Family, Region),
- представляет из себя отсортированный словарь, в котором в качестве ключа – значение ключа из таблицы, в качестве значения – конкретное значение поля и timestamp.

BlockCache

- LRU кэш на чтение. В нем хранятся часто используемые данные, вытесняются старые.

HFile

- свой формат хранения,
- отсортированный по RowKey набор значений в определенной ColumnFamily (Memstore дамп),
- «твёрдая» копия данных, хранящихся в MemStore.



> Запись данных в HBase

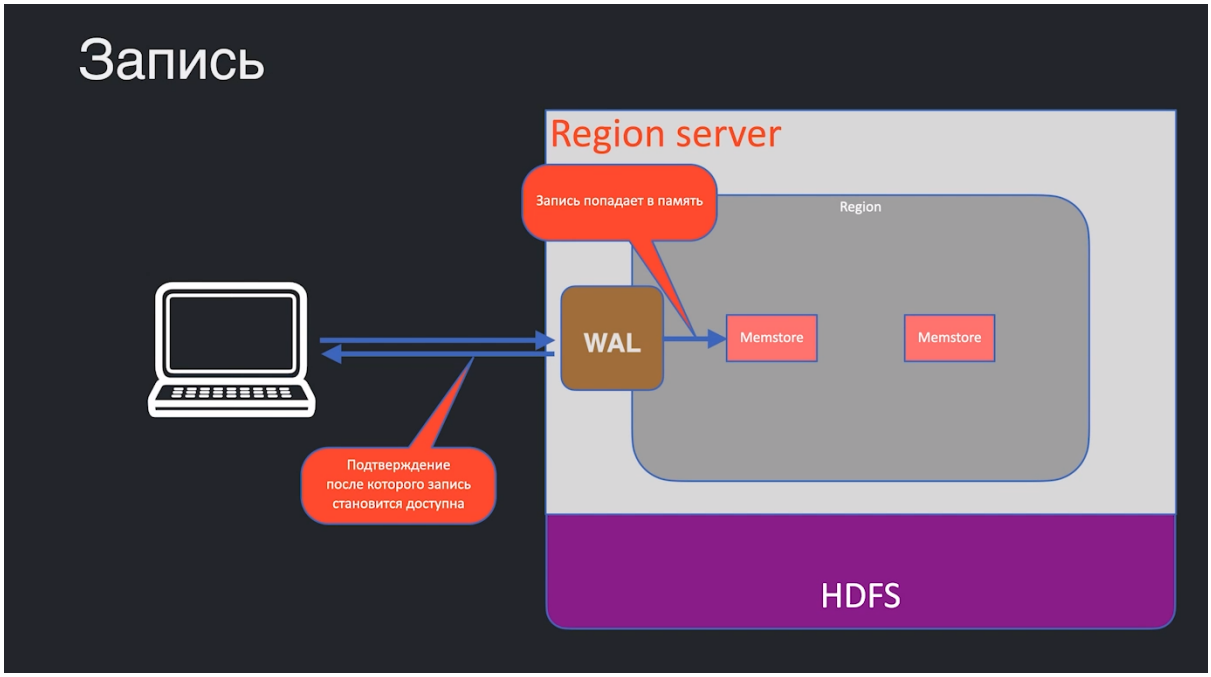
Клиентское приложение при подключении к кластеру Hbase считывает информацию о кластере из служебной таблицы .META. Таким образом оно знает, на какой Region Server нужно прийти, чтобы вставить запись в таблицу.

Также мы знаем ключ, который хотим вставить и можем сопоставить с диапазонами значений ключей, которые обслуживаются в каждом из регионов.

Зная, в какой регион должна попасть запись, мы приходим к определенному Region Server, чтобы совершить операцию Put. В первую очередь эта операция попадает в WAL, который хранится на HDFS, что позволяет не потерять данные. Затем данные попадают в MemStore (кэш на запись). После того, как данные записаны в Memstore происходит отправка пакета, подтверждающего что данные успешно вставлены в таблицу.

Теперь эта запись будет доступна для всех остальных клиентов.

После заполнения MemStore произойдёт сброс данных в HFile, которые запишутся на HDFS. До этого момента информация хранится в оперативной памяти Region Server-а и как журнал операций в WAL.

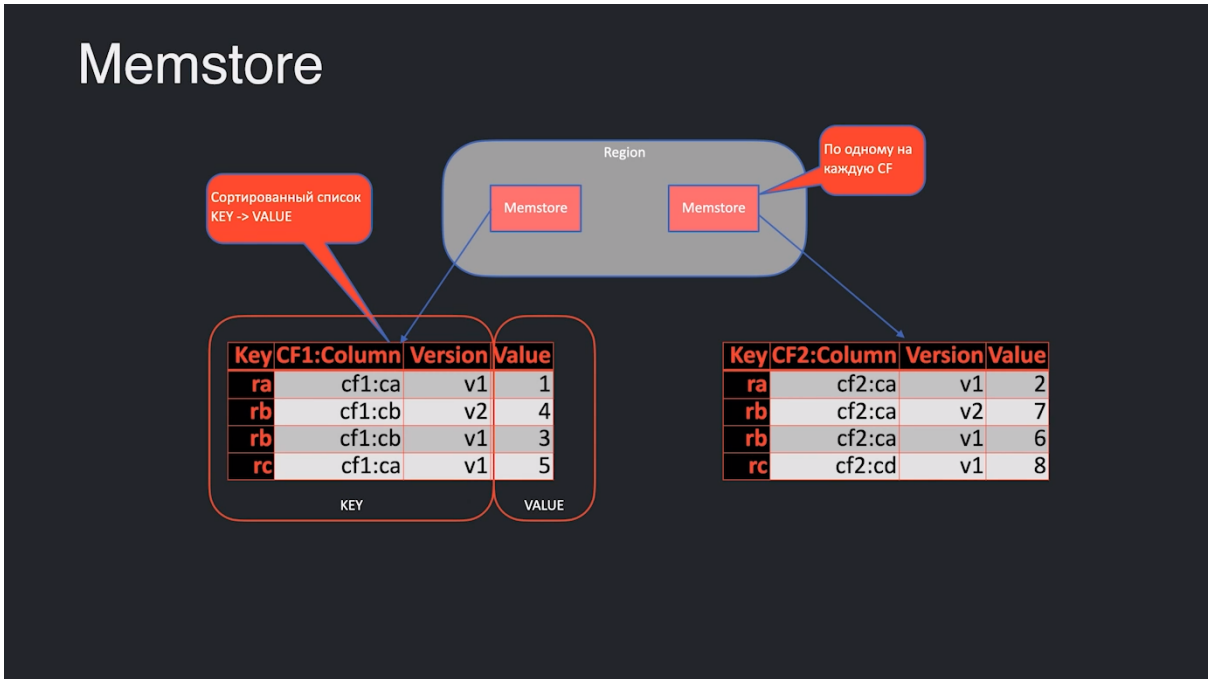


> Memstore и Server flush

> Memstore

В регионе количество MemStore соответствует количеству ColumnFamily в таблице. MemStore устроен как отсортированный список ключей и значений.

Для каждой колонки есть значение, версия этого значения и системное время вставки в базу данных (timestamp).



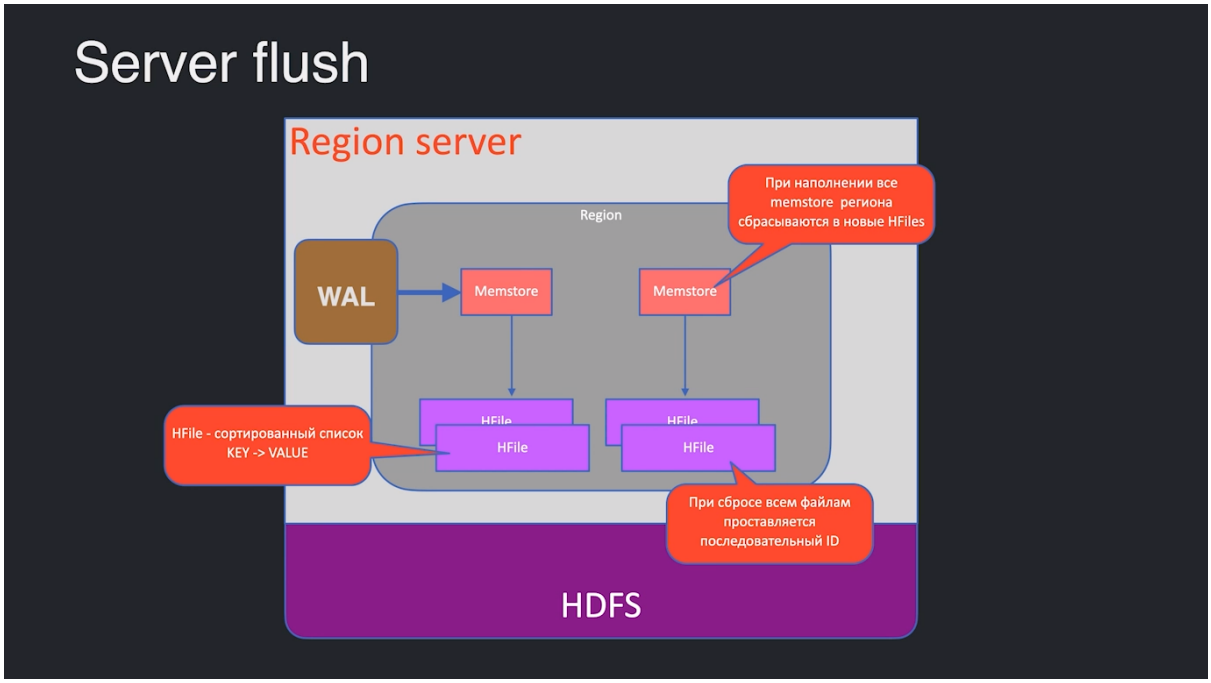
В ColumnFamily под названием CF1 есть колонка Column. Для ключа rb имеется 2 значения: первая версия значения (value) равное 3 и вторая версия значение равная 4.

> Server flush

Memstore имеет ограниченное системным параметром значение (объем памяти). И есть threshold — отсечка (задаётся системным параметром), после которой нужно производить сброс того, что хранится на Memstore на HDFS. При наполнении одного из Memstore до такой отсечки инициируется операция flush (сброс данных на диск).

Данные из Memstore преобразуются в другой формат (отсортированный map из memstore преобразуется в отсортированный list), записываются в HFile и попадает на HDFS. В каждом HFile есть служебный идентификатор операции, который она получает при попадании в WAL.

С помощью HFile и WAL можно высчитать дельту, которая была в памяти и не успела попасть в HFile.



> Minor и Major compaction

Как происходит управление данными, записанными на HDFS. Для каждого региона, при наполнении одного из Memstore происходит синхронный сброс содержимого Memstore в виде HFile на HDFS.

Это приводит к тому, что в этом Memstore HFile получается относительно нормального размера (блок или больше). При этом остальные незаполненные Memstore преобразуются в HFile и эти HFile имеют небольшой размер. Это приводит к появлению т.н. "проблемы мелких файлов на HDFS".

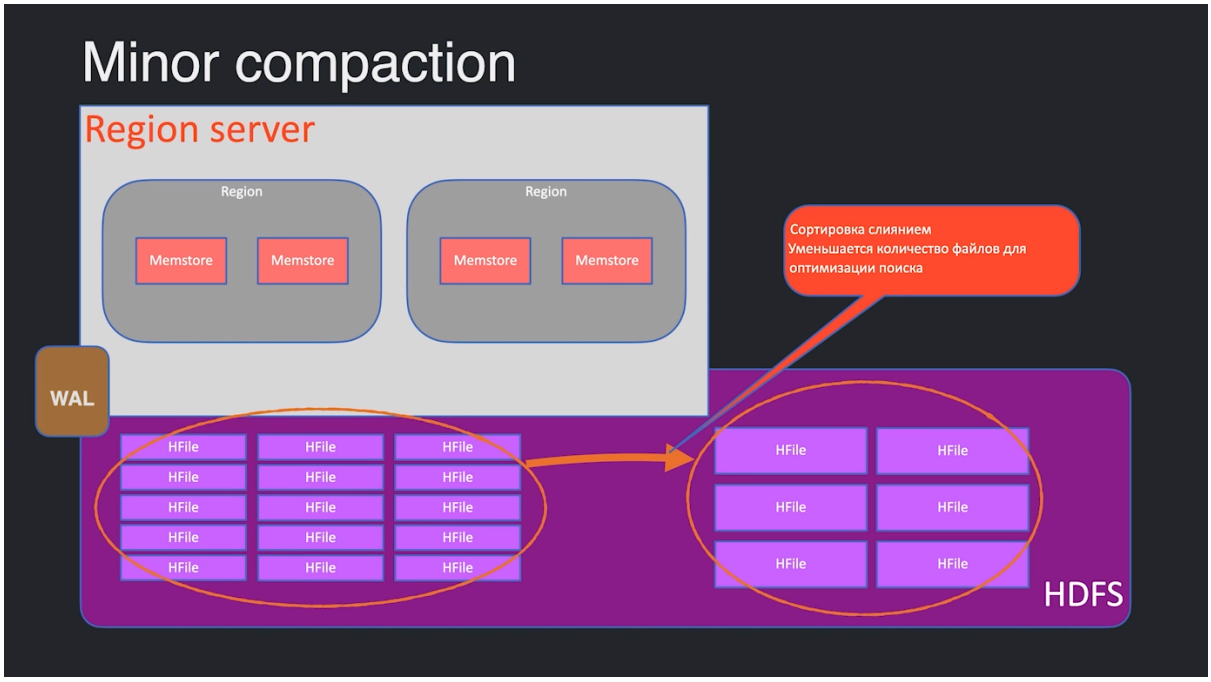
При интенсивной работе HBase у нас будет появляться все больше и больше мелких файлов. Это влияет на скорость получения информации из HFile - скорость падает.

Для решения этой проблемы родилась идея Compaction. Compaction - это объединение небольших HFile в HFile большого размера. Они разделяются на два вида:

> Minor compaction

Minor compaction - запускается автоматически, работает в фоновом режиме и объединяет мелкие HFile. Имеет низкий приоритет по сравнению с другими

операциями.



HFile являются отсортированными по ключу, каждому Memstore соответствуют собственные HFile. Для того, чтобы слить два HFile, которые соответствуют одному и тому же Memstore используется алгоритм сортировки слиянием, и образуется HFile большего размера.

Данная операция не требует большого количества оперативной памяти. Этот процесс постоянно происходит в фоновом режиме на Region Server.

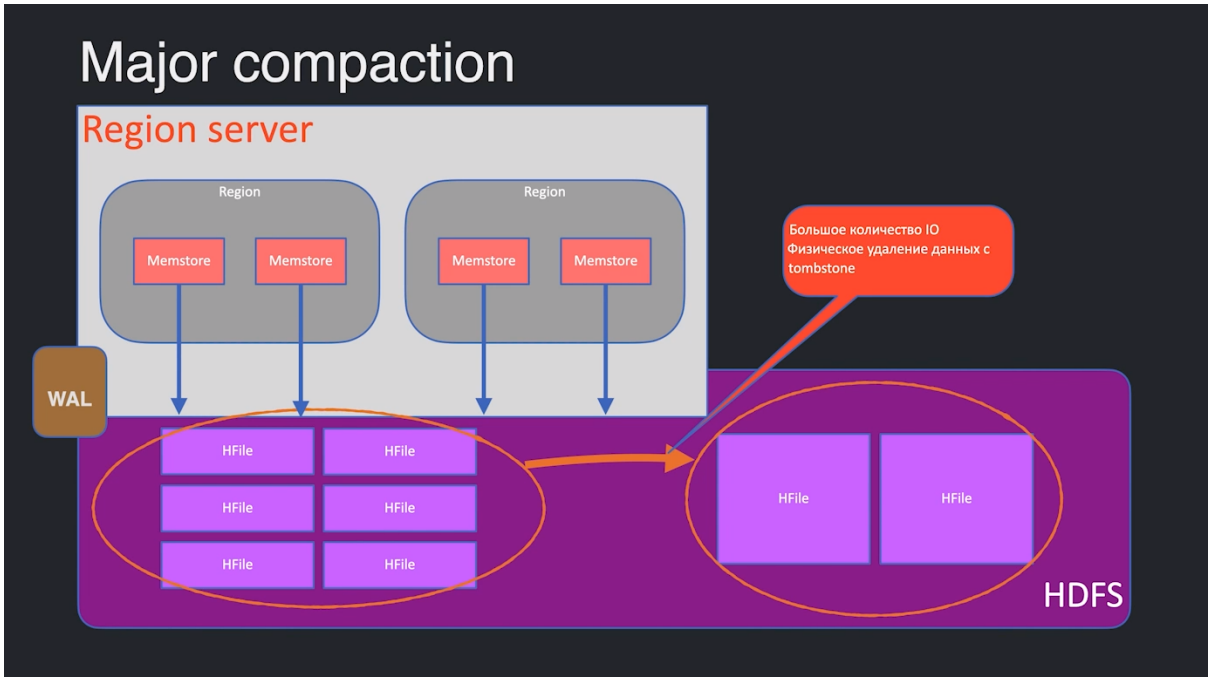
> Major compaction

По сравнению с Minor compaction не просто объединяет мелкие файлы большие, но и производит удаление записей, которые были помечены на удаление меткой tombstone.

Это запускаемый вручную или в случае наступления определенных условий (триггеров), например, срабатывание по таймеру.

Имеет высокий приоритет и может существенно замедлить работу кластера. Во время выполнения Major Compaction также происходит физическое удаление данных, ранее помеченных соответствующей меткой tombstone.

Эту операцию рекомендуется выполнять при невысокой нагрузке на кластер. Например по ночам или на выходных.



> Итоги по Compaction

Compaction - механизм слияния данных в HBase, при котором HFile сливаются в один файл большего размера.

Minor Compaction:

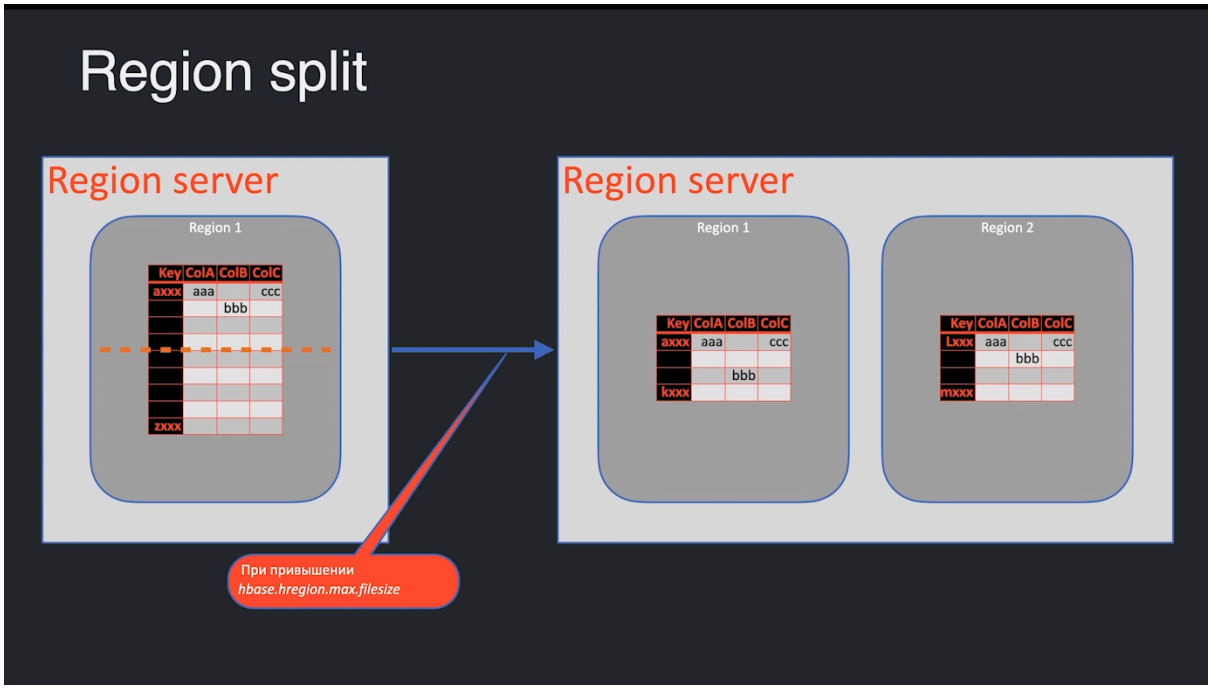
- Происходят постоянно автоматически в фоне
- Почти не снижают производительность
- Не удаляют записи, только сливают несколько маленьких HFile в один большего размера

Major Compaction:

- Запускаются вручную или по расписанию
- Значительное снижение производительности
- Записи с tombstone удаляются физически

> Region split

Region split — это стратегия разделения таблиц на регионы, для повышения производительности работы вашей базы данных.



У нас есть таблица, которая состоит из одного региона. Мы начинаем добавлять в нее данные. Один первоначальный регион обслуживает **Region server**. С накоплением данных, превышающий системный параметр `hbase.hregion.max.filesize` происходит **Region split.**

Наш **Region server** понимает, что он обслуживает регион размера большего, чем должен. поэтому он берет у этого региона начальные и конечные значения ключей, находит условную середину и разделят его на два региона.

В этом случае **Region server** обслуживает два региона одной и той же таблицы. При ребалансировке второй регион может быть переназначен **HMaster'ом** на другой **Region server.**

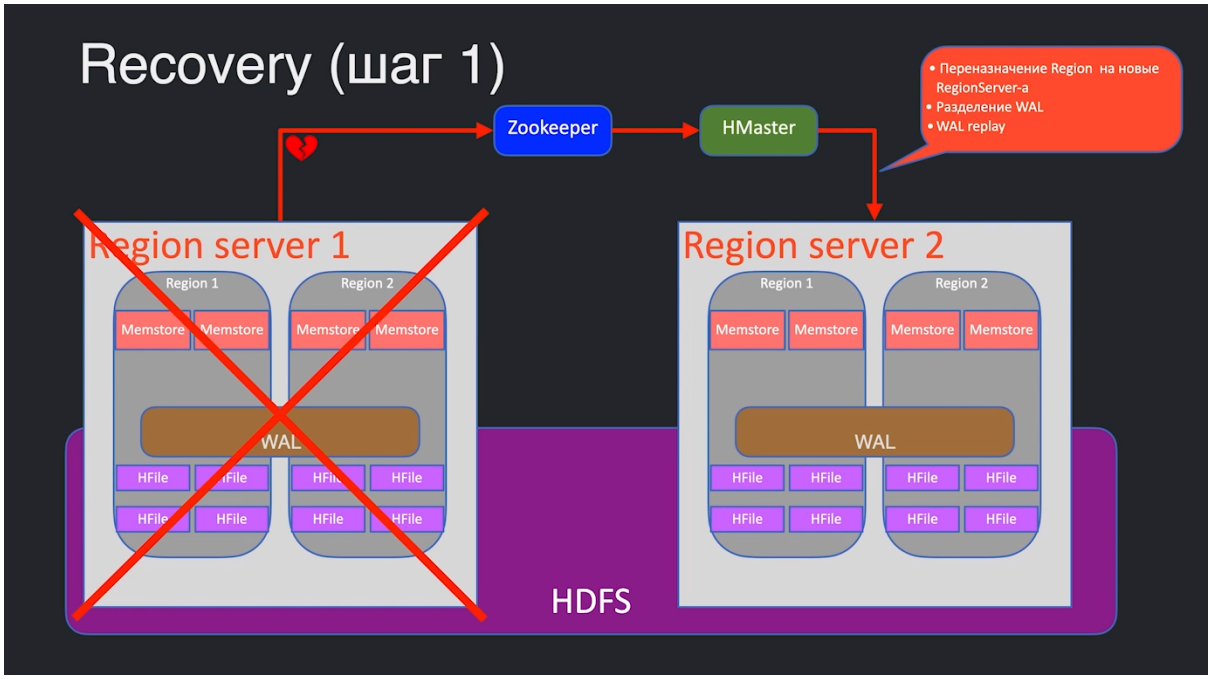
Надо помнить, что обслуживание региона на новом **Region server** не будет максимально эффективным пока не произойдет **Compaction.**

> Recovery

Recovery - восстановление после сбоев. Оно происходит в два этапа.

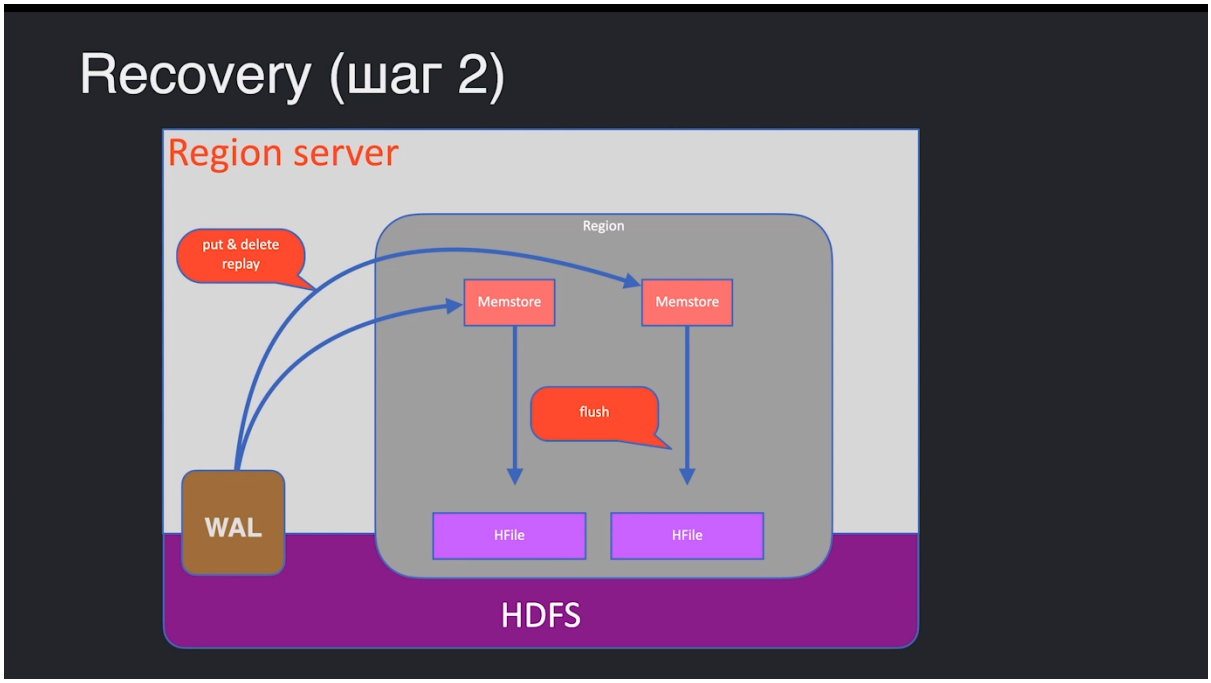
Все **Region server** и **HMaster** постоянно отправляют heartbeat в **Zookeeper**. Если в **Zookeeper** не приходит heartbeat из какого-то Region server-а, **Zookeeper** сообщает **HMaster,** что **Region server** вышел из строя.

Если это продолжается несколько циклов, **HMaster** принимает решение переназначить регионы с одного **Region server** на другой.



Данные (HFile и WAL) хранятся на HDFS доступны всем нодам HBase кластера.

Новый Region server имеет WAL от старого Region server, а также все HFIle от регионов на потерянном Region server. Далее происходит восстановление — запускается новый регион на новом Region server. Из WAL получаем список операций (put, delete) связанных с данным регионом, которые начинают последовательно выполняться – происходит replay с момента последнего flush-а в HFile.



После выполнения всех операций Memstore восстанавливается до состояния, которое предшествовало падению Region server. После наполнения Memstore всеми операциями, происходит принудительный спрос содержания Memstore в виде HFIle на HDFS.

> Особенности HBase

- Нет типов данных (только массив байт)
- Колоночная структура (ColumnFamily хранятся и обрабатываются независимо)
- При разрастании Region производится его разделение (split)
- Гибкие параметры хранения для ColumnFamily (количество версий, сжатие, TTL)
- Служебная таблица .META с данными о топологии кластера. Расположение таблицы хранится в Zookeeper
- Данные надежно хранятся на HDFS, что позволяет проводить переназначение регионов и восстановление после сбоев

> RowKey

- По факту является первичным ключом и ключом сортировки
- Алгоритм формирования обычно выбирается один раз при заведении таблицы
- RowKey должен быть равномерным
- При неправильном выборе RowKey может возникнуть ”Hot Region” – регион, нагрузка на который на порядок выше чем на другие регионы
- Очень сложно поменять RowKey в таблице с сотнями терабайт данных (по факту требуется перезаливка всех данных)