ТЕХНОСФЕРА

Методы распределенной обработки больших объемов данных в Hadoop

Лекция 8: NoSQL, Hbase, Cassandra

Москва 2014

NoSQL

Scaling Up

- Проблемы с масштабированием вверх при росте объемов данных
- RDBMS изначально не были предназначены быть распределенными
- Решения на основе *multi-node database*
- Известно как *‘scaling out’* или *‘horizontal scaling’*
- Разные подходы
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

- Все операции записи идут на *Master nodes*
- Все операции чтения выполняются с реплицированных *Slave nodes*
- Критические операции чтения могут быть некорректны, т.к. данные еще неотреплицировались
- Большие объемы данные приносят большие проблемы из-за больших объемов репликации

Scaling RDBMS - Sharding

- Хорошо масштабируется для операций чтения и записи
- Не всегда прозрачно для приложения (*partition-aware application*)
- Больше нельзя иметь *relationships/joins* между шардами
- Потеря общей интеграции между шардами

Что такое NoSQL?

- **N**ot **O**nly **SQL**
- Класс *non-relational data storage systems*
- Обычно не требуется фиксированная схема таблицы и не используется концепция *join*
- Все NoSQL решения пренебрегают одним из свойств ACID (*Atomicity, Consistency, Isolation, Durability*)
  - CAP-теорема

Dynamo и BigTable

- Три основные статьи на эту тему зародили NoSQL
  - BigTable (Google)
    - *http://labs.google.com/papers/bigtable.html*
  - Dynamo (Amazon)
    - *http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html*
    - *http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf*
  - CAP Theorem

CAP Theorem

- Три основные свойства системы
  - ***Consistency*** - во всех вычислительных узлах в один момент времени данные не противоречат друг друг
  - ***Availability*** - любой запрос к распределённой системе завершается корректным откликом
  - ***Partitionability*** - расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций
- Можно иметь только два из трех свойств в любой *shared-data* системе
- Для горизонтального масштабирования надо применять *partitioning.* Это заставляет выбрать либо *consistency,* либо *availability*
  - Почти во всех случаях выбирается *availability* вместо *consistency*

Consistency Model

- *Consistency Model* определяет правила показа и порядка видимости обновлений данных.
- К примеру:
  - Строка X реплицируется на ноды M и N
  - Клиент A пишет строку X на ноду N
  - Проходит некоторый период времени *t*
  - Клиент B читает строку X с ноды M
  - **Видит ли клиент записанные данные от клиента A?**
  - Consistency - это континуум с некоторыми компромиссами
  - Для NoSQL ответ будет “*может быть*”
  - CAP теорема говорит: *Strict Consistency* не может быть достигнута одновременно с *availability* и *partition-tolerance*

Eventual Consistency

- Пока не происходит никаких операций обновлений данных какое-то долгое время, то все ноды будут в консистентном состоянии
- Для данного *accepted update* и данной ноды в итоге либо *update* достигнет ноды, либо нода будет удалена из кластера
- Известно как *BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency)*
  - *Basically Available* – система доступна все время
  - *Soft State* – данные не должны быть консистентны все время
  - *Eventually Consistent* – становятся консистентными через какое-то время
  - Прямо противоположно ACID

Типы NoSQL

- NoSQL решения обычно бывают двух типов
  - **Key/Value** или ‘*the big hash table*‘
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
  - **Schema-less**, который разделяется на *column-based, document-based* или *graph-based*
    - Cassandra (column-based)
    - CouchDB (document-based)
    - Neo4J (graph-based)
    - HBase (column-based)

Key/Value

- Плюсы
  - Очень быстрые
  - Хорошо масштабируемые
  - Простая модель данных
  - Горизонтально масштабируема
- Минусы
  - Многие структуры данных не так просто смоделировать в виде пар *key/value*

Schema-Less

- Плюсы
  - Модель данных более богатая, чем *key/value*
  - Eventual consistency
  - Распределенная
  - Также имеет хорошую производительность и масштабируемость
- Минусы
  - Обычно нет ACID транзакций и Joins


H·BASE

HBase

- Column-Oriented хранилище данных (Hadoop Database)
- Поддержка рандомных realtime операций CRUD (в отличии от HDFS)
- Распределенная система изначально спроектированная для поддержки больших таблиц
  - Миллиарды строк и миллионы колонок
- Работает на кластере из commodity hardware
- Open-source, написана на Java
- NoSQL
  - Не предоставляет SQL-доступ
  - Не предоставляет реляционной модели (лишь ограниченная часть)

HBase

- Основана на идеях Google BigTable
  - http://labs.google.com/papers/bigtable.html
- Также, как и BigTable работает поверх GFS, HBase реализована поверх HDFS
- Горизонтально масштабируется
  - Автоматическое шардирование
- Операции чтения и записи *strongly consistent*
- Автоматический fail-over
- Простой Java API
- Интеграция с MapReduce
- Thrift, Avro и REST-ful Web-services

Когда нужно использовать HBase

- Хорошо подходит для больших объемов данных
  - Если данных мало, то они все будут лежать на одной или нескольких нодах -> плохая утилизация кластера
- Паттерн доступа к данным
  - Выборка значений по заданному ключу
  - Последовательный скан в диапазоне ключей
- Свободная схема данных
  - Строки могу существенно отличаться по своей структуре
  - В схеме может быть множество колонок и большинство из них будет равно *null*

Когда НЕ нужно использовать HBase

- Традиционный доступ к данным в стиле RDBMS
  - Приложения с транзакциями
  - Реляционная аналитика
    - *'group by', 'join‘ и 'where column like‘* и т.д.
    - Через MapReduce
- Плохо подходит для доступ к данным на основе текстовых запросов

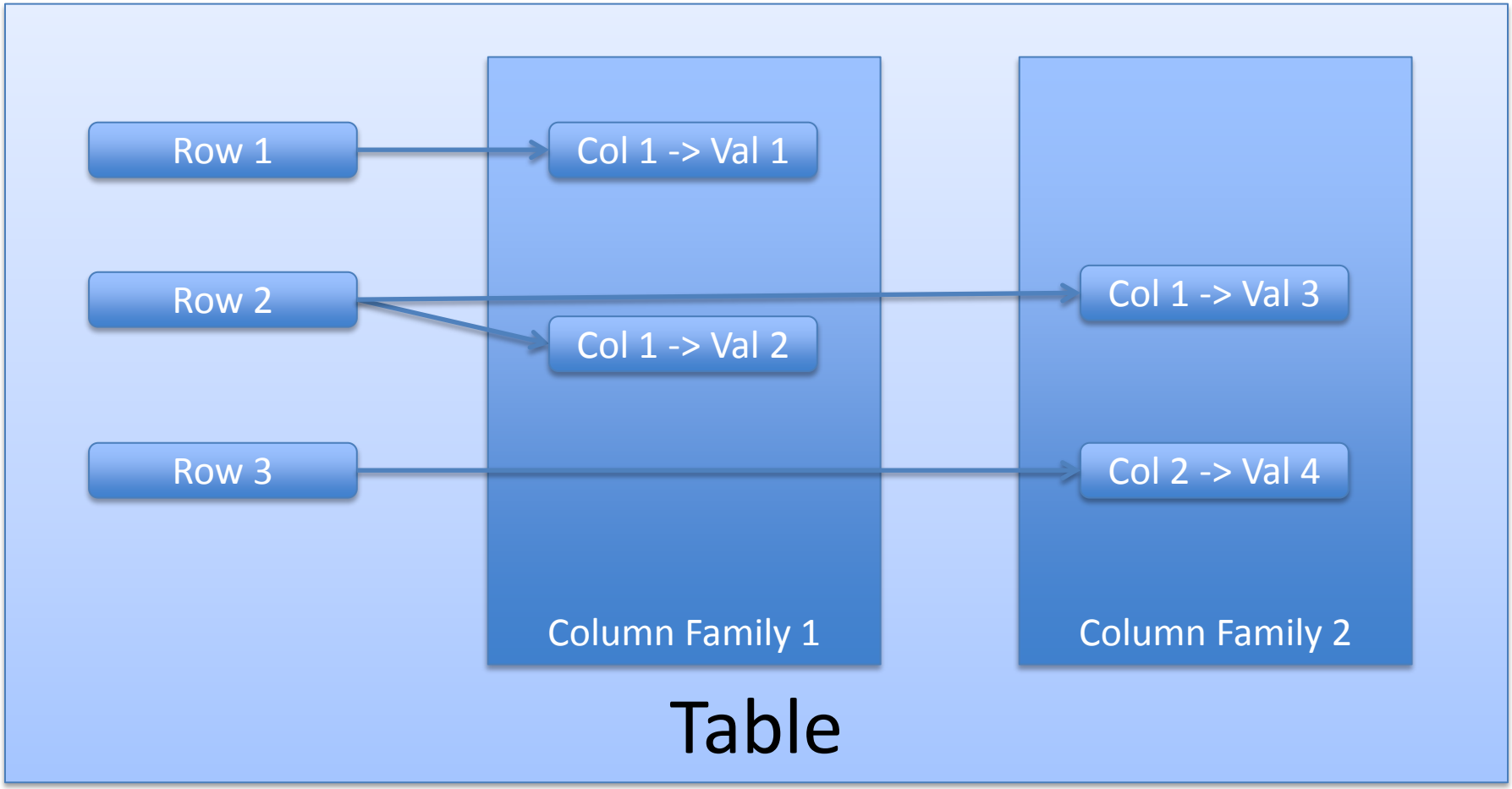
HBase Data Model

- Данные хранятся в таблицах (*table*)
- Таблицы содержат строки (*row*)
  - Доступ к строке осуществляется по уникальному ключу
    - Ключ – это байтовый массив
    - Все может быть ключом
  - Строки отсортированы в лексиграфическом порядке по ключам
- Строки группируются по колонкам в *column families*
- Данные хранятся в ячейках (*cells*)
  - Доступ к ячейке как *row : column-family : column*
  - Данные хранятся также в виде байтового массива

HBase Column Families

- Колонки группируются в C*olumn Families (CF)*
  - Именуются как *family : column*
  - Средство организации данных
  - Различные функции применяются к *column families*
    - Сжатие
    - Опция хранения только в памяти (In-memory)
    - Хранится в одном файле (или нескольких)
      - HFile/StoreFile
- Конфигурация *CF* статична
  - Задается в процессе создания таблицы
    - Можно изменить, но потребуется выключить/включить таблицу
  - Количество *CF* ограничено небольшим числом
- Колонки наоборот НЕ статичны
  - Создаются в runtime
  - Кол-во может исчислять сотнями тысяч для одной CF

HBase Column Families



HBase Timestamps

- Значения в колонках версионные
  - Хранится несколько версий значений
    - Настраивается в конфигурации CF
    - 3 по-умолчанию
  - Дополнительное измерение для данных
  - Timestamp
    - Задается неявно при записи RegionServer’ом
    - Или явно указывается клиентом
  - Версии хранятся в убывающем порядке ts
    - Последнее значение читается первым
- *Value = Table + RowKey + Family + Column + Timestamp*

Hbase Cells

Row Key	Timestamp	CF: “Data”		CF: “Meta”		
		Url	Html	Size	Date	Log
row1	t1	Mail.Ru				Log text 1
	t2				123456	Log text 2
	t3		<html>…	1234		Log text 3
	t4		<HTML>…	2345		Log text 4
row2	t1	OK.Ru			123765	Log text 1
	t2					Log text 2

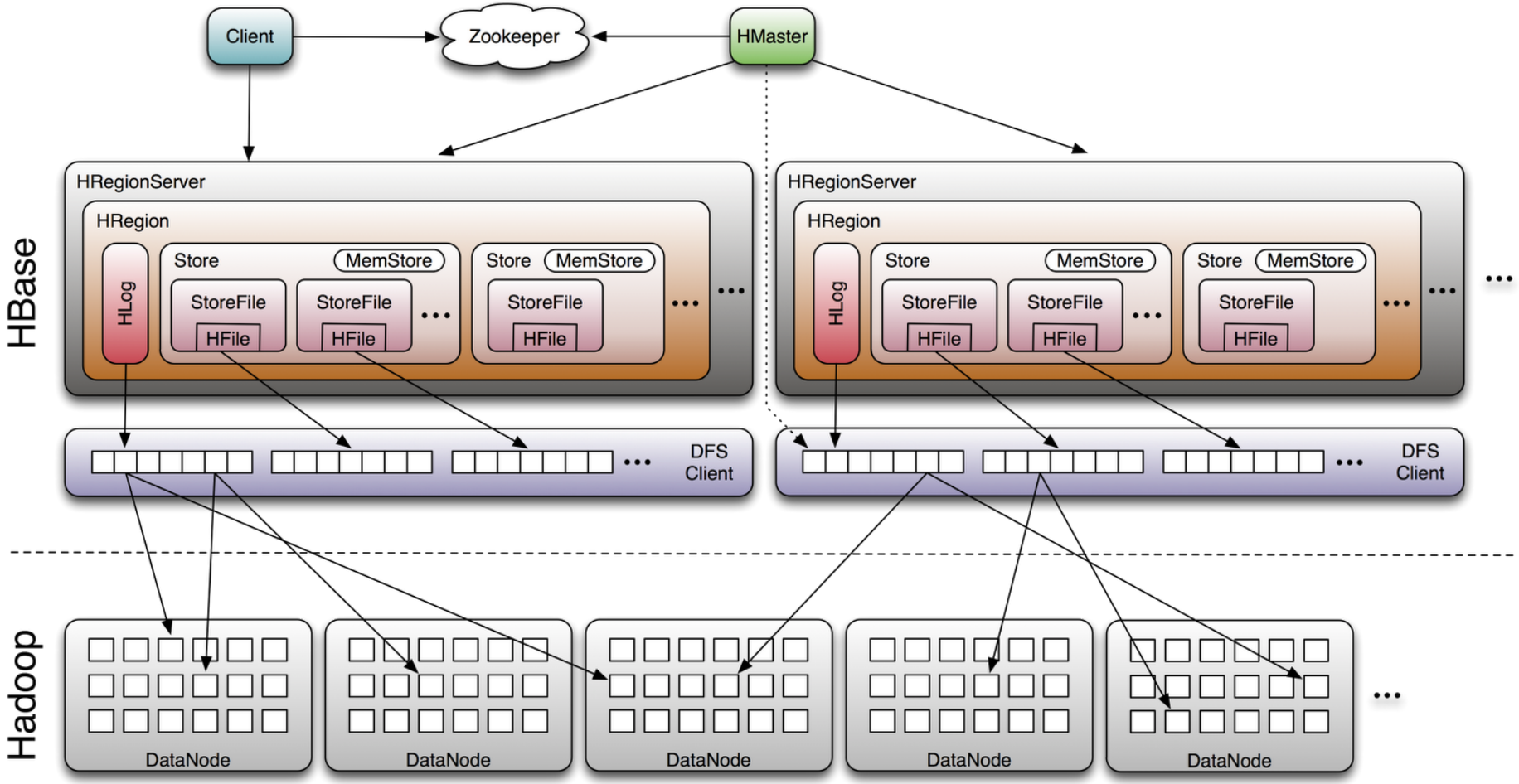
Hbase Cells

Row Key	Timestamp	CF: “Data”		CF: “Meta”		
		Url	Html	Size	Date	Log
row1	t1	**Mail.Ru**				Log text 1
	t2				**123456**	Log text 2
	t3		<html>…	1234		Log text 3
	t4		**<HTML>…**	**2345**		**Log text 4**
row2	t1	**OK.Ru**			**123765**	Log text 1
	t2					**Log text 2**

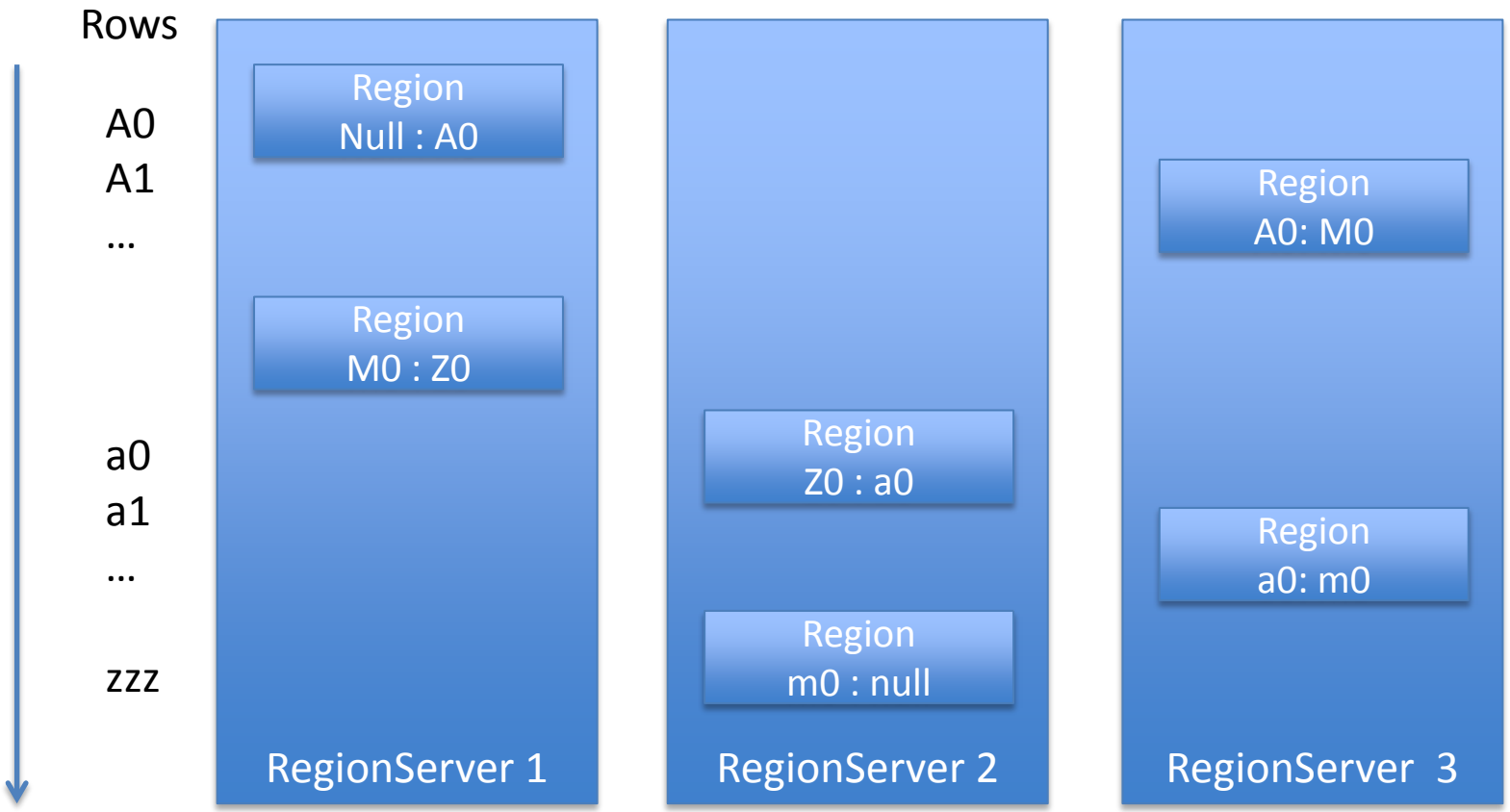
Архитектура Hbase

- **Таблица** делится на регионы
- **Регион** – это группа строк, которые хранятся вместе
  - Единица шардинга
  - Динамически делится пополам, если становится большим
- **RegionServer** – демон, который управляет одним или несколькими регионами
  - Регион принадлежит только одному RS
- **MasterServer (HMaster)** – демон, который управляет всеми RS
- Hbase сохраняет данные в HDFS
  - high availability и fault-tolerance features в HDFS
- Hbase использует ZooKeeper для распределенной координацией действий

Компоненты Hbase



Распределение ключей в RegionServer



Hbase Regions

- **Регион** - это диапозон ключей
  - *Start Key -> Stop Key*
    - *Start Key* включается в регион
    - *Stop Key* не включается
- Добавление данных
  - Изначально есть только один регион
    - Можно предварительно задать кол-во регионов
  - Записывая новые данные размер региона превысит лимит и он разбивается (*split*) на 2 региона по среднему ключу
- Количество регионов на RegionServer зависит от железа. Обычно
  - От 10 до 1000 регионов
  - Размер региона до 2-3 Gb

Hbase Regions

- Сплит региона позволяет
  - Быстрое восстановление если регион зафейлился
  - Баланс нагрузки на RS
  - Split – это быстрая операция
    - Происходит в бэкграунде автоматически
    - Прозрачна для пользователей
  - На больших инсталляциях лучше производить в ручную

Data Storage

- Данные хранятся в файлах *Hfiles (StoreFiles)*
  - Файлы хранятся в HDFS
- *HFile* это *key-value map*
  - Ключи отсортированы в лексиграфическом порядке
- Когда данные добавляются они помещаются сначала в *WAL (Write Ahead Log)* и сохранятся в памяти RS (*memstore*)
- Когда размер данных в памяти превышает лимит, то данные сбрасываются (*flush*) в Hfile
  - Данные, записанные в HFile могут быть удалены из WAL

Data Storage

- Помним, что файлы в HDFS нельзя обновлять in-place
  - Нельзя удалить key-value из HFile
  - Со временем становится все больше и больше HFile’ов
- При удалении добавляется *delete marker* который обозначает, что запись удалена
  - Маркеры используются для фильтрации удаленных записей в runtime

Data Storage

- Для контроля количества файлов и балансировки кластера HBase выполняются операции компакта данных
  - Minor Compaction
    - Маленькие HFile’ы объединяются в бОльшие файлы (n-way merge)
    - Быстрая операция
    - Маркеры удаления не применяются
  - Major Compaction
    - Для каждого региона все файлы в рамках одной CF объединяются в один файл
    - Используются маркеры удаления для того, чтобы не включать удаленные записи

HBase Master

- Отвечает за управление регионами и их распределением
  - Назначает регион на RS
  - Перебалансирует для распределения нагрузки
  - Восстанавливает регион, если он становится недоступен
  - Использует ZooKeeper
    - *http://zookeeper.apache.org*
- Не хранит данные
  - Клиент взаимодействует напрямую с RS
  - Обычно не сильно нагружен
- Отвечает за управление схемой таблиц и ее изменений
  - Добавляет/удаляет таблицы и CF

HBase и Zookeeper

- Zookeeper, основные моменты
  - Написан на Java, очень простое API
  - Операции над директориями и файлами (называются *Znodes*)
  - CRUD Znodes и подписка на *updates*
    - Поддержка *PERSISTENT* и *EPHERMAL Znodes*
  - Клиенты соединятся с Zookeeper через механизм сессий
    - Сессии поддерживаются через *heartbeat*
    - Если клиент не отвечает, сессия истекает и EPHERMAL ноды удаляются
    - Клиенты, которые подписаны на update, уведомляются при удалении и создании нод

HBase и Zookeeper

- Каждый RS создает EPHERMAL ноду
  - Мастер мониторит эти ноды чтобы понимать, что все ноды доступны
  - Мастер отслеживает ноды, на которых происходят fail’ы
- Zookeeper используется также для того, чтобы гарантировать, что только один мастер зарегистрирован

Доступ к HBase

- *HBase Shell*
- *Native Java API*
- *Avro Server*
  - *Cross-language schema compiler*
  - *http://avro.apache.org*
  - Требует для запуска *Avro Server*
- *PyHBase*
- *REST Server*
- *Thrift*
  - *Cross-language schema compiler*
  - *http://thrift.apache.org*
  - Требует для запуска *Thrift Server*

Column Family – Storage Unit

- *Column Family* определяет, как данные будут разделены
- Для каждой CF ячейки данных (*cells*) хранятся в одном (или нескольких) файлах в HDFS
- *Cells,* которые, не имеют значений, не будут записаны и хранится
- *Rows* хранятся как множество *cells*
- Сортированы по RowId, затем по *column qualifier*

Запрос данных из HBase

- По RowId или набору RowIds
  - Отдает только те строки, которые соответствуют заданным id
  - Если не заданы дополнительные критерии, то отдаются все *cells* из всех *CF* заданной таблицы
    - Это означает слияние множества файлов из HDFS
- По *ColumnFamily*
  - Уменьшает количество файлов для загрузки
- По *Timestamp/Version*
  - Может пропускать полностью Hfile’ы, которые не содержат данный диапазон timestamp’ов

Запрос данных из HBase

- По *Column Name/Qualifier*
  - Уменьшает объем передаваемых данных
- По *Value*
  - Можно пропускать ячейки используя фильтры
  - Самый медленный критерий выбора данных
  - Будет проверять каждую ячейку
- Фильтры могут применяться для каждого критерия выбора
  - *rows, families, columns, timestamps u values*
- Можно использовать множественные критерии выбора данных


Cassandra

Cassandra

- Изначально разработана в Facebook
- Следует модели Google BigTable
  - column-oriented
- Использует модель *Dynamo Eventual Consistency*
- Написана на Java
- Open-source, проект Apache
- Использует Apache Thrift для API
  - Cross-language, service-generation framework
  - Binary Protocol (по типу Google Protocol Buffers)
  - Компилируется в C++, Java, PHP, Ruby, Erlang, Perl…

Типичный NoSQL API

- Basic API
  - *get(key)*
  - *put(key, value)*
  - *delete(key)*
  - *execute(key, operation, parameters)*

Data Model

- В Cassandra доступ к данным следующим способом:
  - **Column:** наименьший элемент данных, пара имя и значение
  - **ColumnFamily:** структура для группировки *Columns* и *SuperColumns*
    - *SuperColumns* – это *Columns* внутри *Columns*
  - **Key:** постоянное имя записи
  - **Keyspace:** самый внешний уровень организации данных (имя базы данных)

Data Model

- Опционально
  - **SuperColumn:** именованный список. Содержит стандартные *Column*
    - Users имеет различные поля. Запрос *(:Users, '12345')* может вернуть:
      {‘User' => {'name' => ‘Blah-Blah', ..}, ‘foo': {...}, ‘bar': {...}, ‘whatever': {...}}
    - foo, bar, whatever – имена *SuperColumns.*
      - Определяются «на лету»
      - Может быть любое количество в строке
      - *:Users* – это имя для *SuperColumn Family*
  - *Column* и *SuperColumn* оба являются парой name/value. Ключевое отличие в том, что
    - Стандартное значение для *Column* является string,
    - Для *SuperColumn* значение – это *Map of Columns*

Cassandra и Consistency

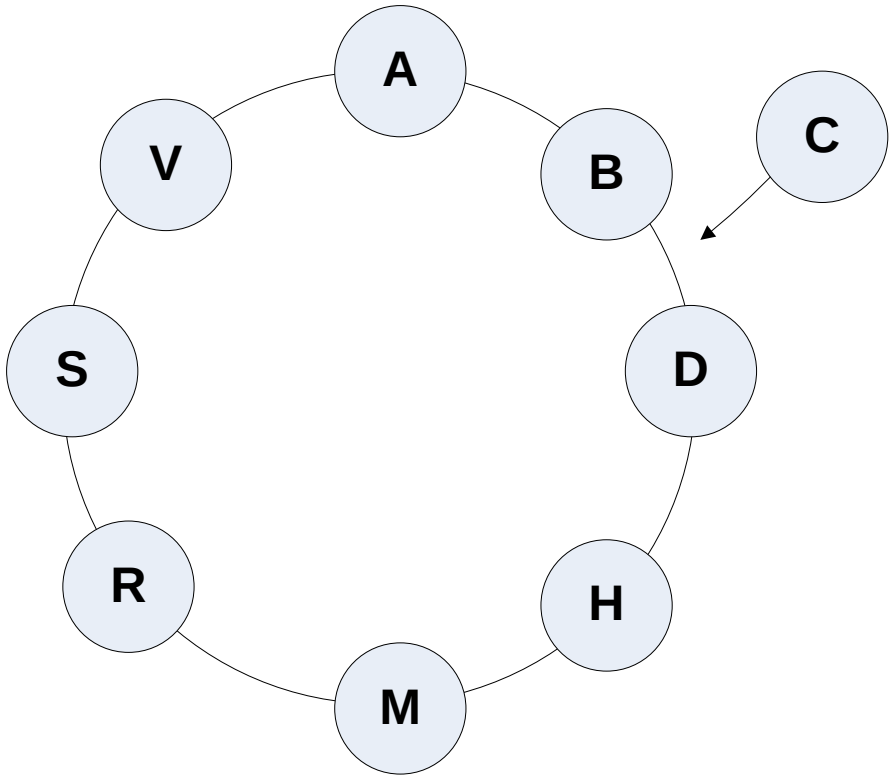
- Eventual consistency
- Cassandra имеет программируемый *read/writable consistency*
- *Read*
  - **One:** возвращается ответ от самой первой ноды, которая отвечает
  - **Quorom:** Запрос ко всем нодам и ответ от ноды, которая имеет самый последний таймстемп, когда большинство ответили
  - **All:** Запрос ко всем нодам и ответ от ноды, которая имеет самый последний таймстемп, когда все ответили

Cassandra и Consistency

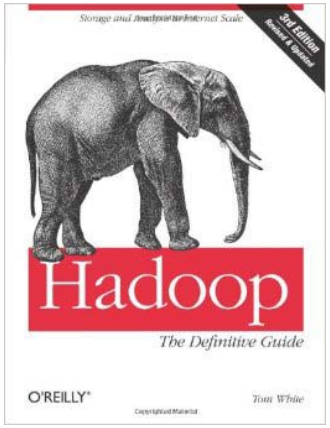
- *Write*
  - **Zero:** Ничего не гарантируется. Асинхронная запись в background
  - **Any:** Гарантируется запись на, как минимум, одну ноду
  - **One:** Гарантируется запись, как минимум, в один *commit log* и в *memory table*
  - **Quorom:** Гарантируется, что запись будет выполнена на N/2 + 1 нод
  - **All:** Гарантируется, что запись дойдет до всех нод

Consistent Hashing

- Партиционирование используя *consistent hashing*
  - Ключи хешируются на точку в *fixed circular space*
  - Круг разделяется на набор упорядоченных слотов и ключи хешируются между ними
- Ноды занимают позиции на круге
- **A**, **B** и **D** существуют
  - **B** отвечает за диапазон **AB**
  - **D** отвечает за диапазон **BD**
  - **A** отвечает за диапазон **VA**
- Добавляется **C**
  - Диапазон **BD** сплитится
  - **C** получает **BC** от **D**



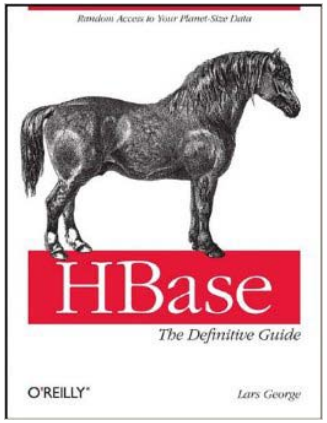
Ресурсы



Hadoop: The Definitive Guide
Tom White (Author)
O'Reilly Media; 3rd Edition

Chapter 13 Hbase

HBase: The Definitive Guide
Lars George (Author)
O'Reilly Media; 1 edition



Вопросы?