Hbase

Overview

- Hbase Introduction
- Hbase Data Model
- Hbase CURD Operations
- Hbase Architecture
- How Hbase works ?

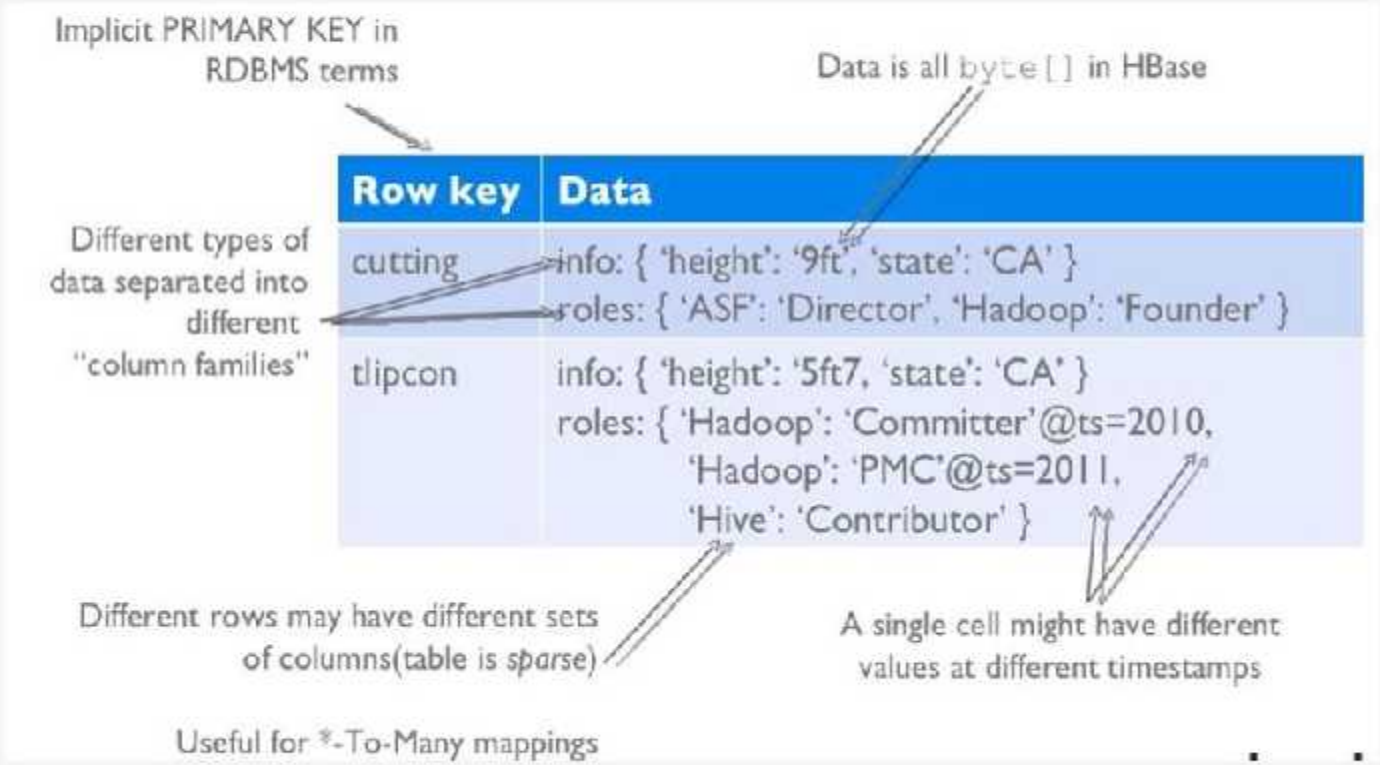
What: HBase is...

Open-source non-relational distributed column-oriented database modeled after Google's BigTable.

- Think of it as a sparse, consistent, distributed, multidimensional, sorted map:
  - labeled tables of rows
  - row consist of key-value cells:

(row key, column family, column, timestamp) -> value

HBase Logical View



HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*

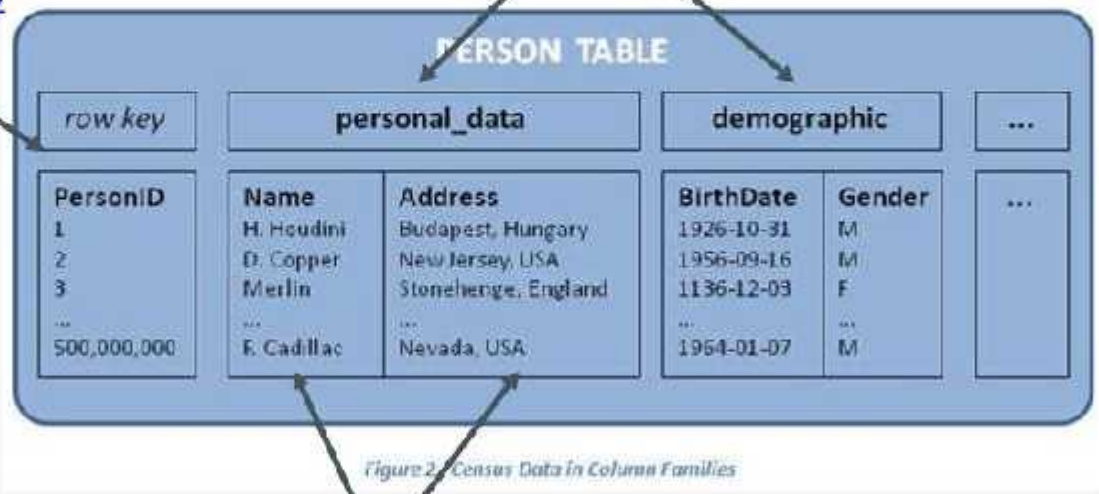
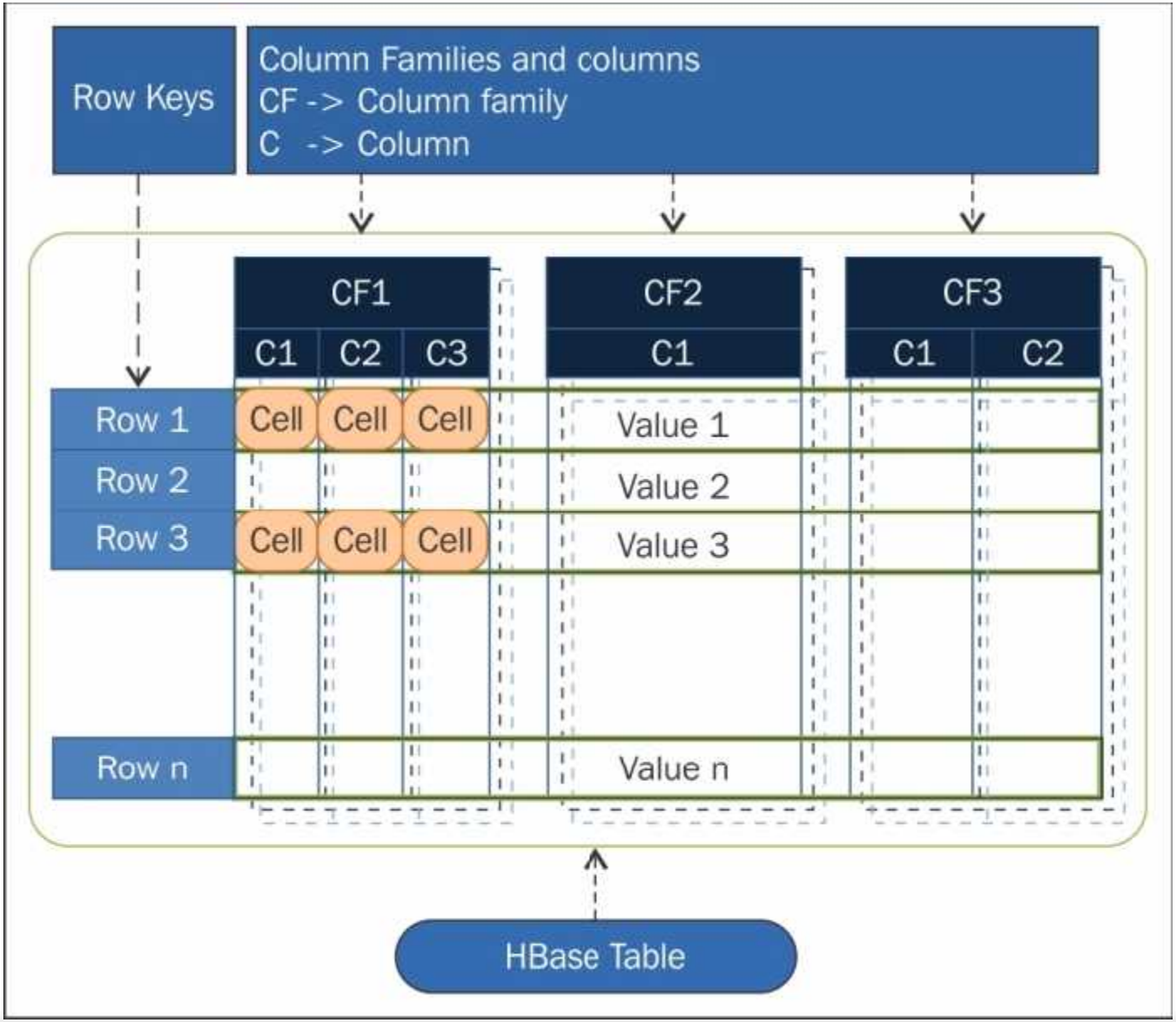
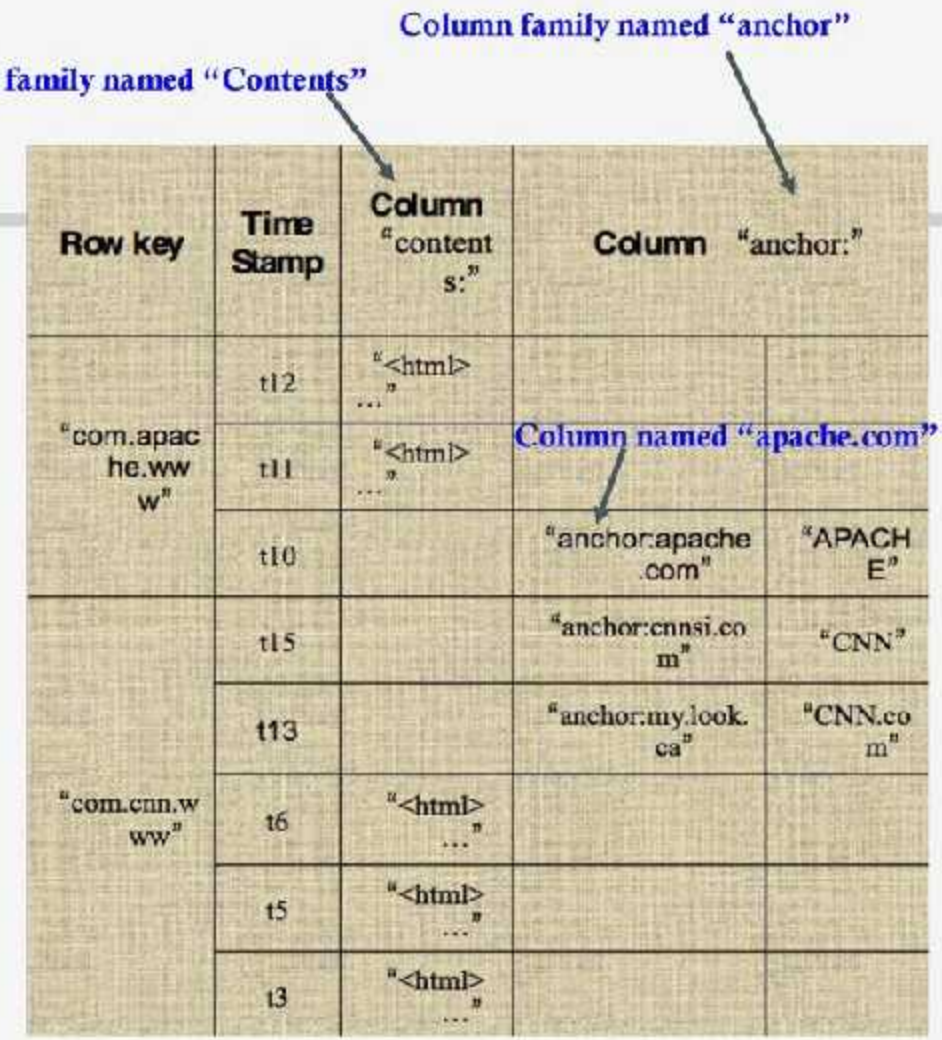


Figure 2: Census Data in Column Families

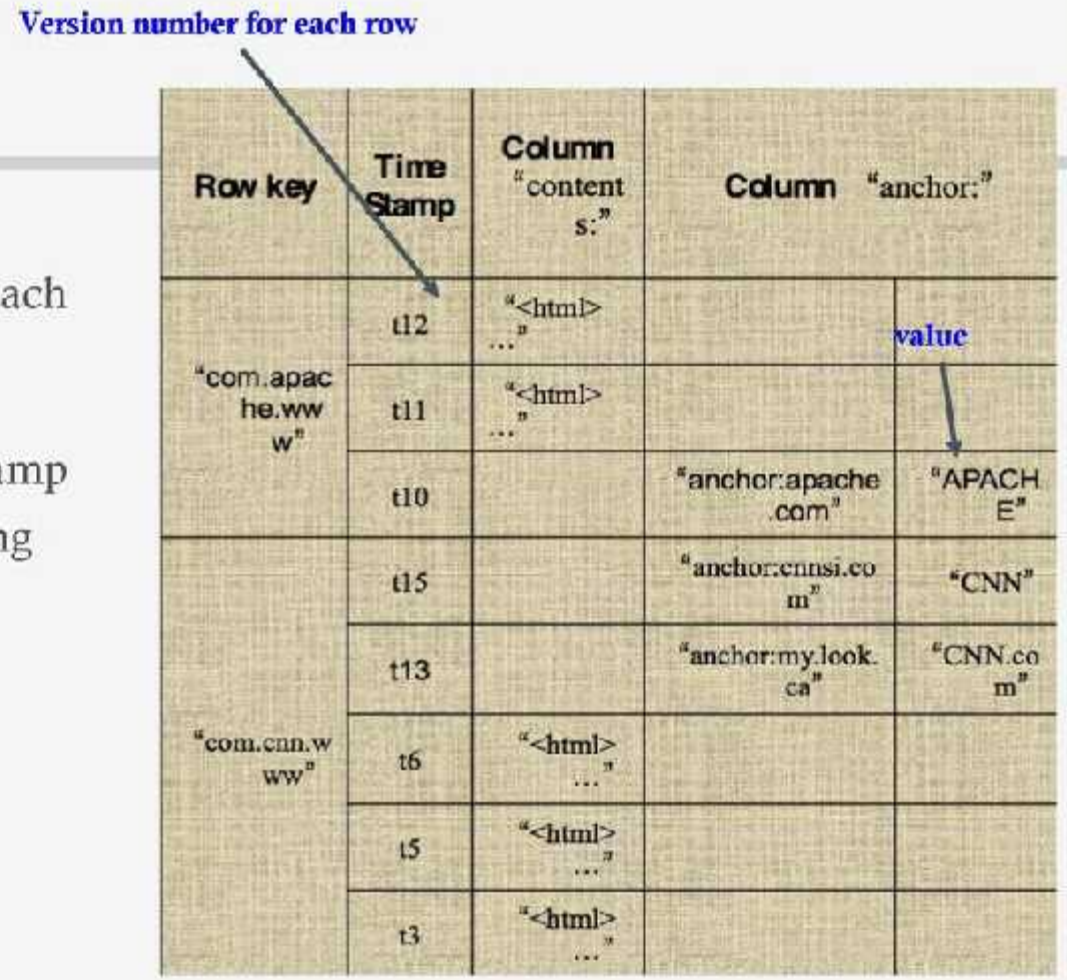
Each column family consists of one or more *Columns*


Row Keys
Column Families and columns
CF -> Column family
C -> Column
CF1
CF2
CF3
C1
C2
C3
C1
C1
C2
Row 1
Cell
Cell
Cell
Value 1
Row 2
Value 2
Row 3
Cell
Cell
Cell
Value 3
Row n
Value n
HBase Table

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - HIas a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - *familyName:columnName*



- **Version Number**
  - Unique within each key
  - By default→ System's timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array



Notes on Data Model

- HBase schema consists of several ***Tables***
- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema
- HBase has ***Dynamic Columns***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

"Roles" column family has different columns in different cells

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Example

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

info Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

roles Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

cloudera

Hbase CURD Operations

Examples in Hbase.

(a) Creating a table called EMPLOYEE with three column families: Name, Address, and Details.

(b) Inserting some in the EMPLOYEE table; different rows can have different self-describing column qualifiers (Fname, Lname, Nickname, Mname, Minit, Suffix, ... for column family Name; Job, Review, Supervisor, Salary for column family Details).

(c) Some CRUD operations of Hbase.

(a) creating a table:

```
create 'EMPLOYEE', 'Name', 'Address', 'Details'
```

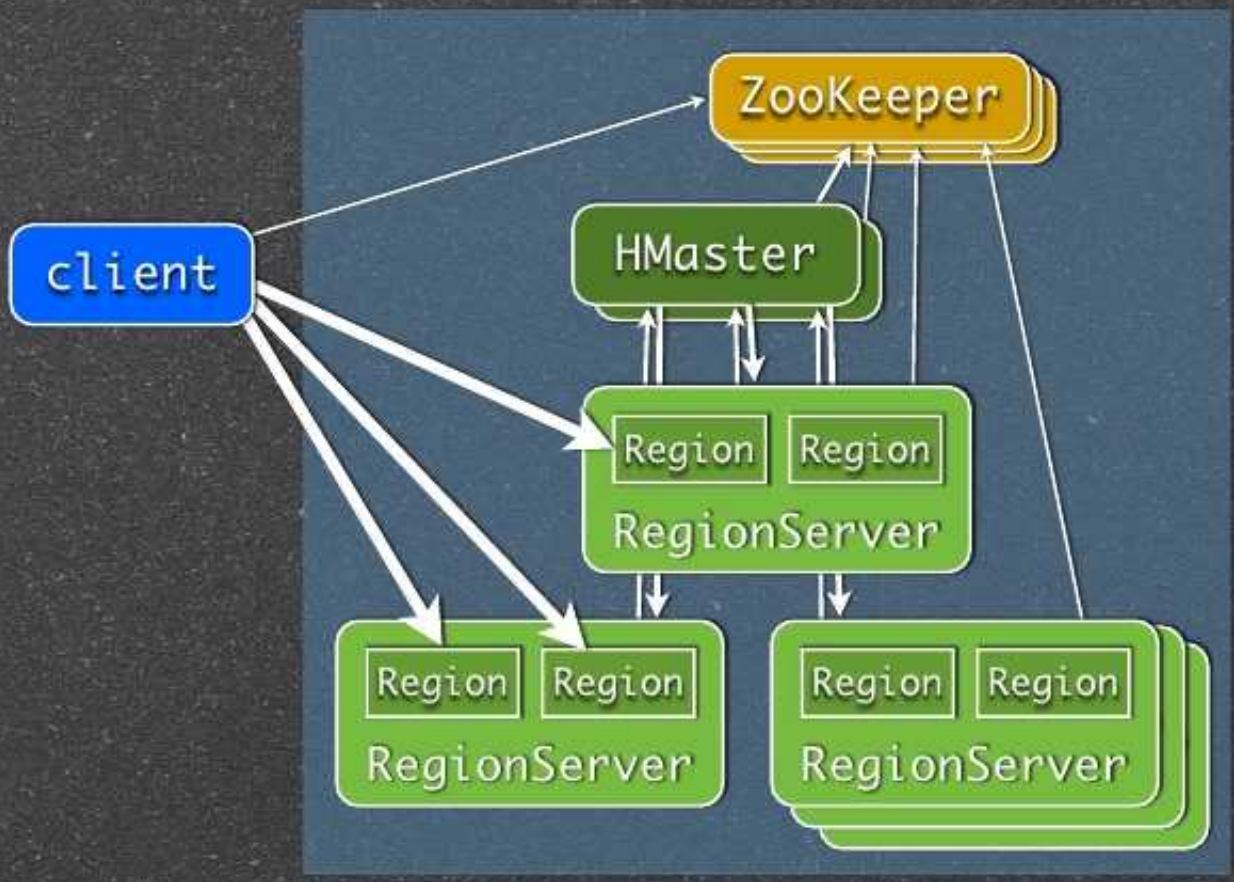
(b) inserting some row data in the EMPLOYEE table:

```
put 'EMPLOYEE', 'row1', 'Name:Fname', 'John'
put 'EMPLOYEE', 'row1', 'Name:Lname', 'Smith'
put 'EMPLOYEE', 'row1', 'Name:Nickname', 'Johnny'
put 'EMPLOYEE', 'row1', 'Details:Job', 'Engineer'
put 'EMPLOYEE', 'row1', 'Details:Review', 'Good'
put 'EMPLOYEE', 'row2', 'Name:Fname', 'Alicia'
put 'EMPLOYEE', 'row2', 'Name:Lname', 'Zelaya'
put 'EMPLOYEE', 'row2', 'Name:MName', 'Jennifer'
put 'EMPLOYEE', 'row2', 'Details:Job', 'DBA'
put 'EMPLOYEE', 'row2', 'Details:Supervisor', 'James Borg'
put 'EMPLOYEE', 'row3', 'Name:Fname', 'James'
put 'EMPLOYEE', 'row3', 'Name:Minit', 'E'
put 'EMPLOYEE', 'row3', 'Name:Lname', 'Borg'
put 'EMPLOYEE', 'row3', 'Name:Suffix', 'Jr.'
put 'EMPLOYEE', 'row3', 'Details:Job', 'CEO'
put 'EMPLOYEE', 'row3', 'Details:Salary', '1,000,000'
```

(c) Some Hbase basic CRUD operations:

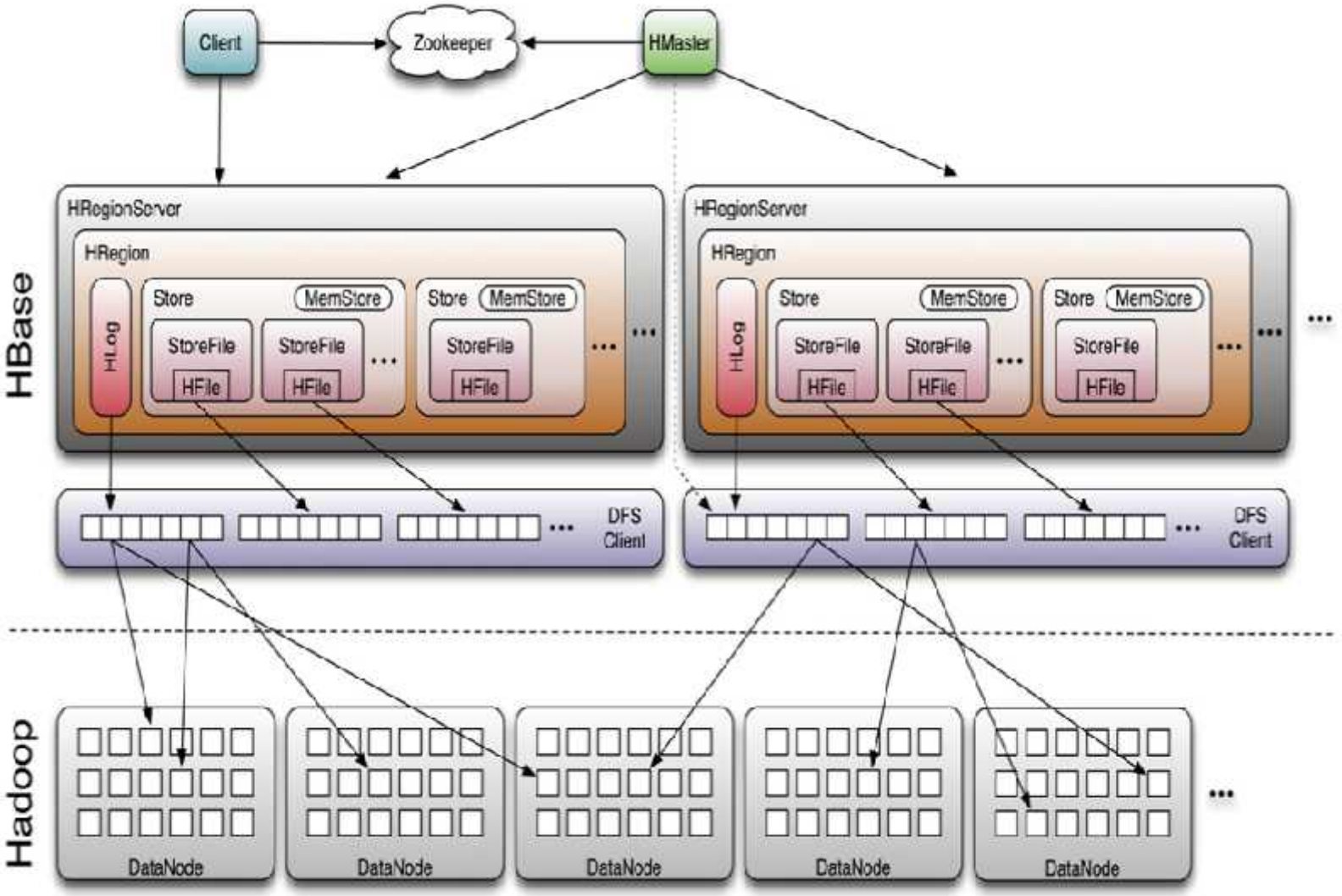
Creating a table: create <tablename>, <column family>, <column family>, ...
Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
Reading Data (all data in a table): scan <tablename>
Retrieve Data (one item): get <tablename>,<rowid>

Logical View: Regions on Cluster

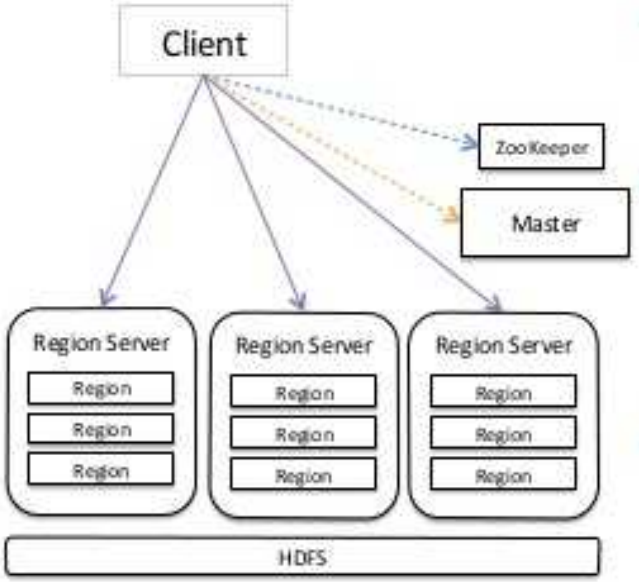


Monday, July 9, 12

Hbase Architecture



Master, Region Servers and Regions



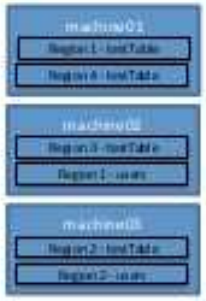
- **Region Server**
  - Server that contains a set of Regions
  - Responsible to handle reads and writes
- **Region**
  - The basic unit of scalability in HBase
  - Subset of the table's data
  - Contiguous, sorted range of rows stored together.
- **Master**
  - Coordinates the HBase Cluster
    - Assignment/Balancing of the Regions
  - Handles admin operations
    - create/delete/modify table, ...

cloudera

Autosharding and .META. table

- A Region is a Subset of the table's data
- When there is too much data in a Region…
  - a split is triggered, creating 2 regions
- The association "Region -> Server" is stored in a System Table
- The Location of .META. Is stored in ZooKeeper

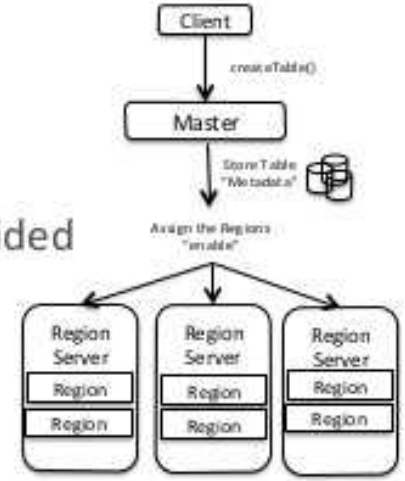
Table	Start Key	Region ID	Region Server
testTable	Key-00	1	machine01.host
testTable	Key-31	2	machine03.host
testTable	Key-65	3	machine02.host
testTable	Key-83	4	machine01.host
…	…	…	…
users	Key-AB	1	machine03.host
users	Key-KG	2	machine02.host



cloudera

The Write Path – Create a New Table

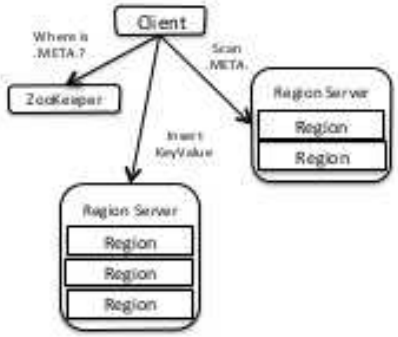
- The **client** asks to the **master** to create a new Table
  - hbase> create 'myTable', 'cf'
- The **Master**
  - Store the Table information ("schema")
  - Create Regions based on the key-splits provided
    - no splits provided, one single region by default
  - **Assign** the **Regions** to the Region Servers
    - The assignment Region -> Server is written to a system table called ".META."



cloudera

The Write Path – "Inserting" data

- table.put(row-key:family:column, value)
- The **client** asks **ZooKeeper** the location of .META.
- The **client** scans .META. searching for the **Region** Server responsible to handle the **Key**
- The **client** asks the **Region Server** to insert/update/delete the specified key/value.

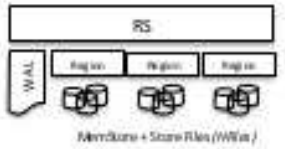


- The **Region Server** process the request and dispatch it to the Region responsible to handle the **Key**
  - The operation is written to a **Write-Ahead Log** (WAL)
  - ...and the KeyValues added to the Store: "**MemStore**"

cloudera

The Write Path – Append Only to Random R/W

- Files in **HDFS** are
  - **Append-Only**
  - **Immutable** once closed
- HBase provides Random Writes?
  - ...not really from a storage point of view
  - KeyValues are **stored** in **memory** and written to disk on pressure
    - Don't worry your data is safe in the WAL!
      - (The Region Server can recover data from the WAL is case of crash)
    - But this allow to **sort** data by **Key** before writing on disk
  - Deletes are like Inserts but with a "remove me flag"

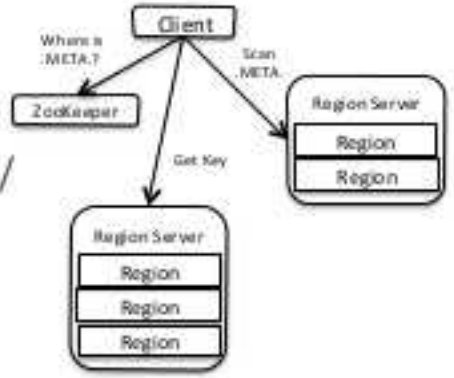


Store Files

cloudera

The Read Path – "reading" data

- The **client** asks **ZooKeeper** the location of .META.
- The **client** scans .META. searching for the **Region** Server responsible to handle the **Key**
- The **client** asks the **Region Server** to get the specified key/ value.
- The **Region Server** process the request and dispatch it to the Region responsible to handle the **Key**
  - **MemStore** and **Store Files** are **scanned** to find the key



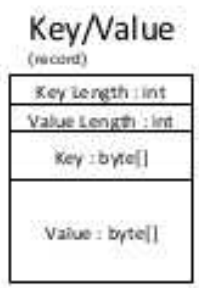
cloudera

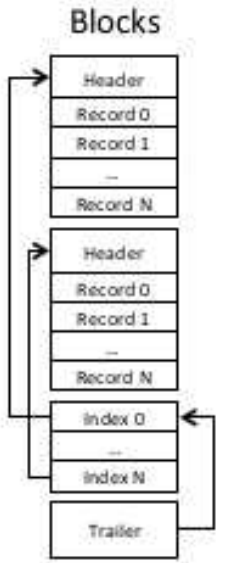
The Read Path – Append Only to Random R/W

- Each **flush** a **new file** is created
- Each file have KeyValues **sorted** by **key**
- Two or more files can contains the same key (updates/deletes)
- To find a Key you need to scan all the files
  - ...with some optimizations
  - Filter Files Start/End Key
  - Having a bloom filter on each file

HFile format

- Only Sequential Writes, *just append(key, value)*
- Large Sequential Reads are better
- Why grouping records in blocks?
  - Easy to split
  - Easy to read
  - Easy to cache
  - Easy to index (if records are sorted)
  - Block Compression (snappy, lz4, gz, ...)





cloudera

Data Block Encoding

- "Be aware of the data"
- Block Encoding allows to compress the Key based on what we know
  - Keys are sorted… prefix may be similar in most cases
  - One file contains keys from one Family only
  - Timestamps are "similar", we can store the diff
  - Type is "put" most of the time…

"on-disk" KeyValue

cloudera

Thank you