Big Data Analytics and Reasoning - Practice 04

Giuseppe Mazzotta

HBase

HBase is a NoSQL database that use a columnar data model

Supports massively parallelized processing via MapReduce for using HBase as both source and output.

Automatic RegionServer failover

Data versioning and Auto-Sharding

Is not the best choice for every context

- Suitable for store huge amount of heterogeneous data, no sparsity
- Missing features: typed columns, secondary indexes, triggers, and advanced query languages, etc.

Data Model

The most basic unit is a column

A column is a pair of the form:

column_family:column_qualifier

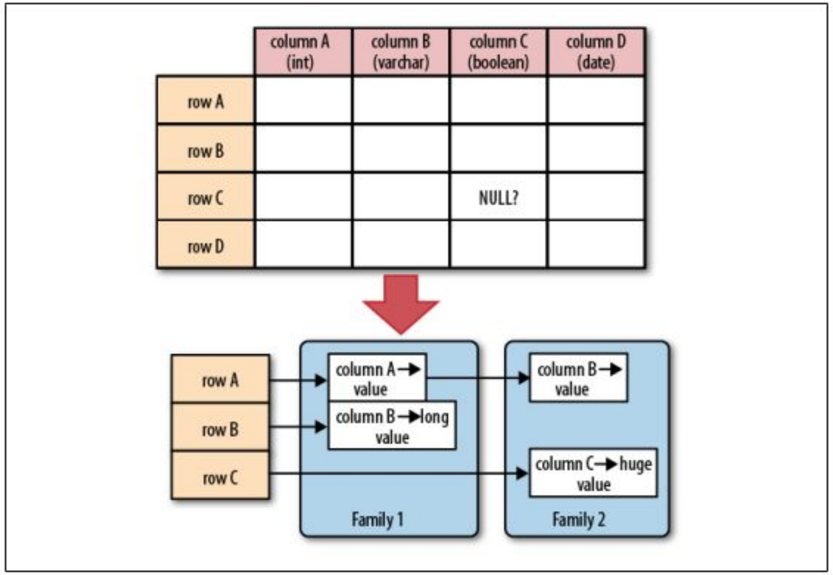
Column family group together a set of qualifiers - Semantical and Performance reason

One or more columns form a row

Each row is identified by a row_key

A set of rows form a table

A namespace is a collection of tables



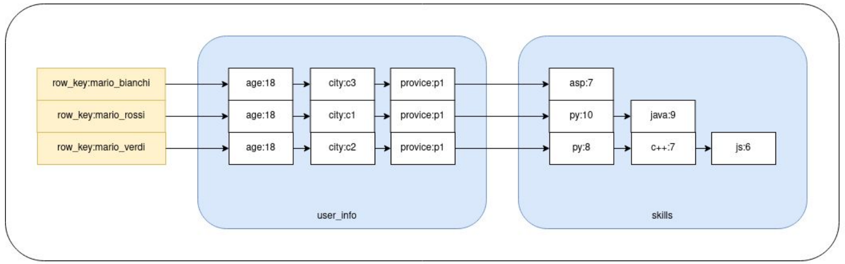
(Table, RowKey, Family, Column, Timestamp) → Value

Data Model - Table Example

For each employee we are interested in:

- Name
- Surname
- Age
- City
- Province
- List<Skill>
    - Skill represent a programming language and a confidence level

Note qualifier can be used to store information



1. HBase Structure

- **HMaster**
  Master service of an HBase cluster
  Assigns regions to regionservers
  Balances data load among regionservers
  Exposes interface for all metadata changes
- **RegionServers**
  Slave services of an HBase cluster
  Serving and managing regions and eventually split them
- **Zookeeper**
  Used to elect a cluster master and to keep the metadata of the cluster

HBase Architecture

HBase tables are divided into regions

Regions store a subset of rows and are hosted by regionserver

Each region is stored in different HFiles according to column families

HFile are stored into the HDFS

RegionServers use Write-Ahead-Log to keeps track of operation not stored permanently

RegionServers are able to perform compaction of multiple HFiles

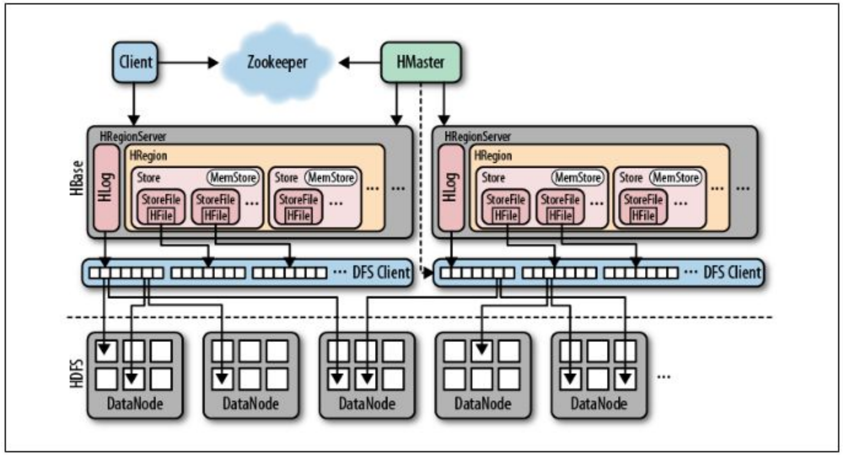


Table hbase:meta

Catalog table used for storing metadata information about other tables

Structure:

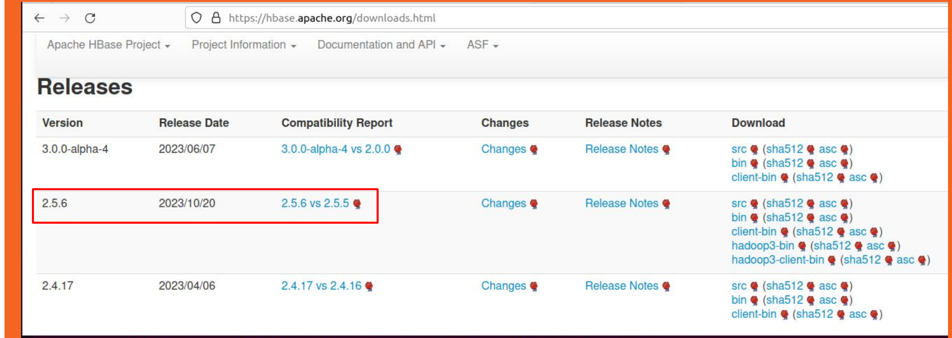
- Row key - Region key of the form (table, region start key, region id)
- info:regioninfo stores a serialized object containing region information
- info:server contains the machine and port for the particular region
- info:serverstartcode stores the start-time of the region server the store the particular region

Download and Install HBase

Download the binary archive of the hbase distribution from the official website in the master machine

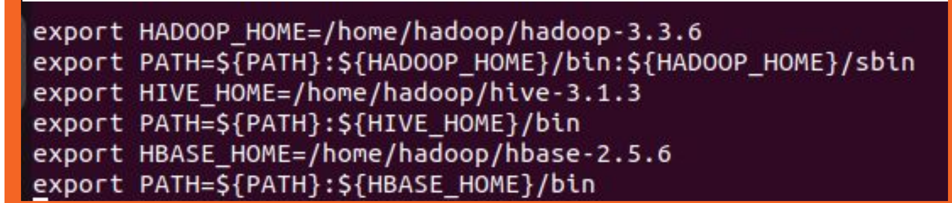
Unfold the archive and export HBASE_HOME environment variable into .bashrc

Add to the PATH variable the bin folder of hive : ${HBASE_HOME}/bin



Releases

Version	Release Date	Compatibility Report	Changes	Release Notes	Download
3.0.0-alpha-4	2023/06/07	3.0.0-alpha-4 vs 2.0.0	Changes	Release Notes	src (sha512 asc) bin (sha512 asc) client-bin (sha512 asc)
2.5.6	2023/10/20	2.5.6 vs 2.5.5	Changes	Release Notes	src (sha512 asc) bin (sha512 asc) client-bin (sha512 asc) hadoop3-bin (sha512 asc) hadoop3-client-bin (sha512 asc)
2.4.17	2023/04/06	2.4.17 vs 2.4.16	Changes	Release Notes	src (sha512 asc) bin (sha512 asc) client-bin (sha512 asc)



```
export HADOOP_HOME=/home/hadoop/hadoop-3.3.6
export PATH=${PATH}:${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin
export HIVE_HOME=/home/hadoop/hive-3.1.3
export PATH=${PATH}:${HIVE_HOME}/bin
export HBASE_HOME=/home/hadoop/hbase-2.5.6
export PATH=${PATH}:${HBASE_HOME}/bin
```

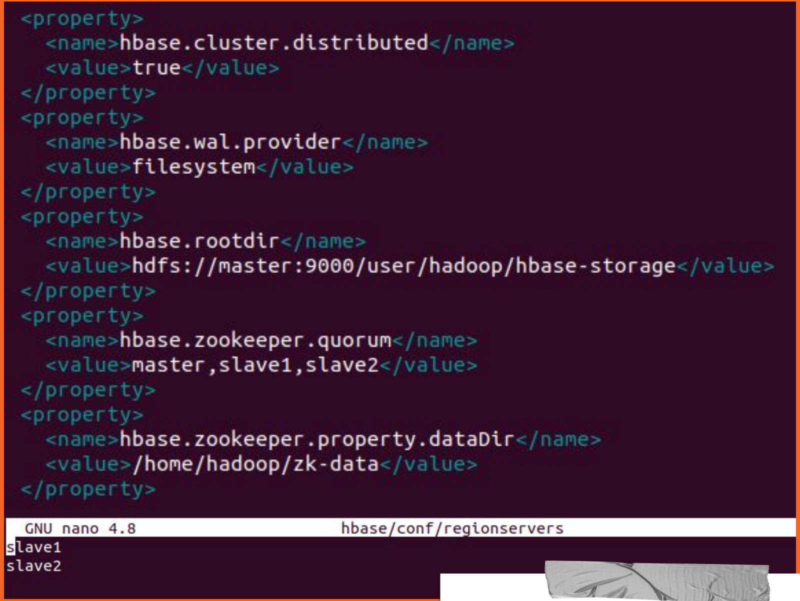
Configure HBase

HBase configuration files are located into ${HBASE_HOME}/conf

Main configuration:

- HBase storage location on HDFS
- Regionservers hostnames
- Zookeeper configuration

HBase has both master and slave services then it has to be installed on each machine of the cluster



```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
<property>
  <name>hbase.wal.provider</name>
  <value>filesystem</value>
</property>
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://master:9000/user/hadoop/hbase-storage</value>
</property>
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>master,slave1,slave2</value>
</property>
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/home/hadoop/zk-data</value>
</property>
```

```
  GNU nano 4.8                    hbase/conf/regionservers
slave1
slave2
```

Tip

Remember: HBase configuration must be repeated in each machine

PER RUNNARE MAPREDUCE CON HBASE E COSE COSI

Aggiun

HADOOP-3.3.6/etc/hadoop/hadoop-env.sh

```
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:${HBASE_HOME}/lib/*
```

Cambiare anche la java_home nel file hbase-env.sh

2. Java API

HBase provide a Java Client API

➔ **CRUD operation on HBase**

Create, **R**ead, **U**pdate and **D**elete HBase tables directly from Java

Main Java classes:

- **Admin** - Administrative operation as create, delete and more
- **Get** - Read row from HBase
- **Result** - Encodes table rows
- **Put** - Write row in HBase
- **Scan** - Read HBase table rows

2. Java API

➔ **Filters**

Read operations (Scan or Get) admit filters:

- **Comparison operators**:
  LESS, EQUAL, GREATER_OR_EQUAL, and more
- **Comparators**
  BinaryComparator, SubstringComparator, RegexStringComparator
- **Available Filters**
  RowFilter, FamilyFilter, QualifierFilter, SingleColumnValueFilter, PrefixFilter
- **FilterList**
  Combines multiple filters: MUST_PASS_ONE, MUST_PASS_ALL

2. Java API

➔ **Map Reduce**

HBase can be used as input or output of mapreduce applications
Main Java classes:

- **TableMapReduceUtil** - Allow us to initialize Map/Reduce job to read/write HBase tables
- **TableMap** - Mapper interface that receive hbase row as **Result** object
- **TableReduce** - Reducer interface that send data to store as **Put** object
- **TableInputFormat / TableOutputFormat** - Used to specify input/output format of the mapreduce format

Let’s practice with the hbase API ...