HBASE

Before looking into HBASE, lets look into what is NOSQL.

**Intro to NoSQL:**

* A NoSQL originally referring to non SQL or non relational is a database that provides a mechanism for storage and retrieval of data.
* NoSQL databases are used in real-time web applications and big data and their use are increasing over time. NoSQL systems are also sometimes called Not only SQL to emphasize the fact that they may support SQL-like query languages.
* It is not only managed with RDBMS as a backend, rather we have dynamic schema, auto sharding, reliable, and open source solution.

**Categories of NoSQL:**

* Columnar
* HBase
* cassandra
* Document based
* MongoDB
* Elastic Search
* Solar
* Key value based
* Accumulo
* Graph
* Neo4j, Giraph

**Characteristics of NoSQL:**

* Dynamic schema
* Auto Sharding
* Replications
* Integrated caching
* No joins, Low cost.

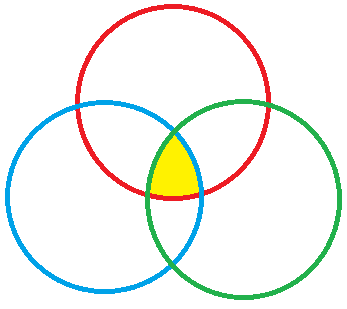
**CAP theorem:**

CA = SQL/RDBMS

* SQL
* Oracle
* MySQL

CP = NoSQL/Column

* Hadoop
* Big table
* HBase
* Mem Cache DB
* Graph

****

AP = NoSQL/Document/Key value

* Dynamic DB
* Couch DB
* Cassandra

**Availability**

**Partitioning**

**Consistenc**y

**HBASE**

HBase is hadoop’s database which provides real time random read/write on huge datasets

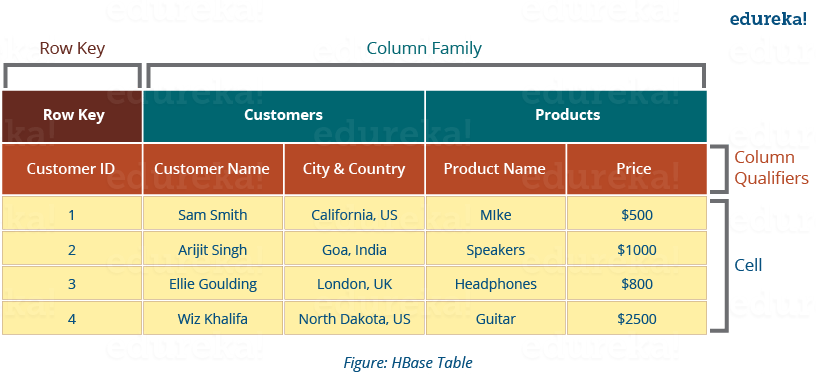
(ie) OLTP on top of OLAP data

Which used HDFS as a base storage in the form of Hfiles.

Difference:

|  |  |
| --- | --- |
| **RDBMS** | **HBase** |
| * Row oriented * Join are possible * Narrow table * OLTP * Aggregation and compression are costly * No Dynamic schema * No flexible * No Auto Sharding | * Column oriented * Joins are not possible * Wide table * OLTP to OLAP * Aggregation and compression are less costly * Its dynamic schema format * Its flexible * Auto sharding(auto scaling up) |

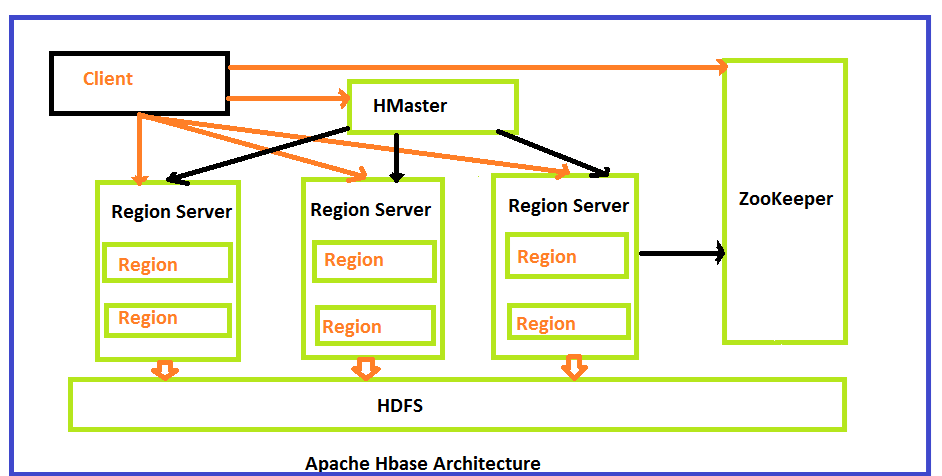
Storage hierarchy:



* Column family
* Collection of columns
* Tables
* Collection of column families
* Row
* Spread across column families
* Row key
* Row identification by unique ID
* Columns
* Collection of key value pair
* Cell
* Each value of the column
* Timestamp
* Version of a cell.

|  |
| --- |
| Concept:   * Each table will have one unique key. * Each table can have multiple column family. * Each column family is dynamic in schema. * The unique key is common throughout the column families. * For one unique key, if we need to maintain multiple column values, we need to have versions. |

Architecture:

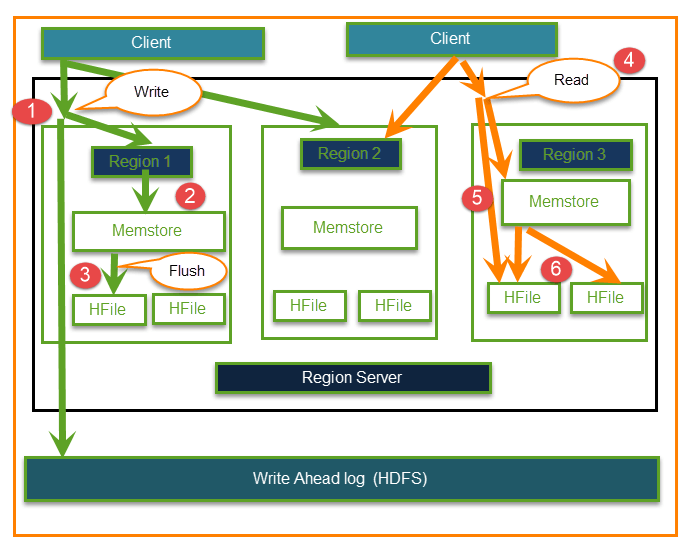


HBase architecture consists mainly of four components

* HMaster
* HMaster in HBase is a process which helps to assign the regions to region servers.
* It balances the loads by assigning the regions.
* It manages the Hadoop clusters
* Helps to create, modify, and delete tables in the database.
* It also cares about different tasks when the client wants to changes the schema or metadata.
* HRegionserver
* The region servers are the main working nodes. It handles the read, write, modify requests for the clients. The region server runs on every node in the hadoop cluster.
* When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where the actual column family resides.
* It has a read cache called **block cache**, read data are stored in the read cache, and when the cache is full, recently used data is removed.
* Another cache is present here called MemStore. It is the write cache. It stored new data that is not yet stored in the disks. Each column-family has different write cache in it.
* It has the actual storage file called HFile. It stores the actual data on a disk.
* HRegions
* HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.
* Zookeeper
* This is an Open-Source server which enables the reliable distributed coordination. Zookeeper is a centralized service that maintains the configuration information. It also maintains distributed synchronization etc.
* Zookeeper service keeps track on all region servers in HBase.

HBase Read and Write Data Explained

The Read and Write operations from Client into Hfile can be shown in below diagram.



**Step 1)** Client wants to write data and in turn first communicates with Regions server and then regions

**Step 2)** Regions contacting memstore for storing associated with the column family

**Step 3)**First data stores into Memstore, where the data is sorted and after that, it flushes into HFile.

The main reason for using Memstore is to store data in a Distributed file system based on Row Key.

Memstore will be placed in Region server main memory while HFiles are written into HDFS.

**Step 4)**Client wants to read data from Regions

**Step 5)**In turn Client can have direct access to Mem store, and it can request for data.

**Step 6) Client** approaches HFiles to get the data. The data are fetched and retrieved by the Client.

Memstore holds in-memory modifications to the store. The hierarchy of objects in HBase Regions is as shown from top to bottom in below table.

Memstore holds in-memory modifications to the store. The hierarchy of objects in HBase Regions is as shown from top to bottom in below table.

|  |  |
| --- | --- |
| Table | HBase table present in the HBase cluster |
| Region | HRegions for the presented tables |
| Store | It stores per ColumnFamily for each region for the table |
| Memstore | Memstore for each store for each region for the table |
| It sorts data before flushing into HFiles |
| Write and read performance will increase because of sorting |
| StoreFile | StoreFiles for each store for each region for the table |
| Block | Blocks present inside StoreFiles |

HBase vs. HDFS

HBase runs on top of HDFS and Hadoop. Some key differences between HDFS and HBase are in terms of data operations and processing.

|  |  |
| --- | --- |
| **Hbase** | **HDFS** |
| Low latency operations | High latency operations |
| Random reads and writes | Write once Read many times |
| Accessed through shell commands , client API in Java, REST, Avro or Thrift | Primarily accessed through MR (Map Reduce) jobs |
| Storage and process both can be perform | It's only for storage areas |

**Lets see how to install HBASE:**

1. Go to the below path. cd /home/hduser/install/
2. Extract the tarball,

HBase installation

---------------------------

tar xvzf hbase-0.98.4-hadoop2-bin.tar.gz

sudo mv hbase-0.98.4-hadoop2 /usr/local/hbase sudo chown -R hduser:hadoop /usr/local/hbase

Zookeeper installation

---------------------------

tar xvzf zookeeper-3.4.6.tar.gz

sudo mv zookeeper-3.4.6 /usr/local/zookeeper sudo chown -R hduser:hadoop /usr/local/zookeeper

1. Zookeeper config

cd /usr/local/zookeeper/conf mv zoo\_sample.cfg zoo.cfg

vi zoo.cfg dataDir=/usr/local/zookeeper/data

Create the below dir for zookeeper mkdir /usr/local/zookeeper/data

1. Start Zookeeper by running below command:

cd /usr/local/zookeeper/bin

./zkServer.sh start

1. Edit hbase environment script cd

/usr/local/hbase/conf

echo 'export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_71' >> hbase-env.sh echo 'export HBASE\_MANAGES\_ZK=false' >> hbase-env.sh

1. Edit the hbase-site.xml to set hbase distribution, hbase root data dir, zookeeper quorum and zk port.

vi /usr/local/hbase/conf/hbase-site.xml

<configuration>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.rootdir</name>

<value>hdfs://localhost:54310/user/hduser/hbase</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>localhost</value>

</property>

<property>

<name>hbase.zookeeper.property.clientPort</name>

<value>2181</value>

</property>

<property>

<name>hbase.zookeeper.property.dataDir</name>

<value>/usr/local/zookeeper/data</value>

</property>

</configuration>

**This property is to add at the time of configuring phoenix. (this is later when we work with phoenix)**

*<property>*

*<name>hbase.regionserver.wal.codec</name>*

*<value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>*

*</property>*

1. Once completed the above steps start the hbase daemon start-hbase.sh

Type jps and see if zookeeper and hbase is running

4240 NodeManager

14129 ResourceManager

13761 DataNode

13475 QuorumPeerMain 🡪 zookeeper

16436 Jps

13973 SecondaryNameNode

15285 Main

13656 NameNode

14968 HRegionServer

8572 RunJar

14844 HMaster

1. To get into the hbase interactive shell type the below command hbase shell
2. Type 'list' to see if hbase is working properly

**HBASE WORKOUTS**

Creating a table "Patient" with the column Families (Personal and Medical)

hbase(main):010:0> **create 'Patient','Personal','Medical'**

0 row(s) in 2.7740 seconds

=> Hbase::Table - Patient

Inserting a record into the table

put 'Patient','001','Personal:pname','Ramesh'

put 'Patient','002','Personal:pname','saravanan'

put 'Patient','002','Personal:filenum','100'

put 'Patient','003','Personal:pname','gowtham' put 'Patient','004','Personal:pname','amudhan' put 'Patient','005','Personal:pname','alex'

put 'Patient','002','Personal:age','24'

put 'Patient','105','Personal:pname','alex' put 'Patient','202','Personal:age','24'

put 'Patient','202','Personal:filenum','101'

put 'Patient','202','Personal:addr','3 first ave,NJ'

put 'Patient','001','Medical:history','Anemic'

put 'Patient','105','Medical:history','General check'

put 'Patient','102','Medical:history','Arthritis'

put 'Patient','102','Medical:oldhistory','Ostophenia'

Check whether the below put works with the column family used as Medical1 instead of Medical

**put 'Patient','102','Medical1:oldhistory','Ostophenia'**

ERROR: Failed 1 action: Patient: 1 time,

Here is some help for this command:

Put a cell 'value' at specified table/row/column and optionally

timestamp coordinates. To put a cell value into table 'ns1:t1' or 't1'

at row 'r1' under column 'c1' marked with the time 'ts1', do:

hbase> put 'ns1:t1', 'r1', 'c1', 'value'

hbase> put 't1', 'r1', 'c1', 'value'

hbase> put 't1', 'r1', 'c1', 'value', ts1

hbase> put 't1', 'r1', 'c1', 'value', {ATTRIBUTES=>{'mykey'=>'myvalue'}}

hbase> put 't1', 'r1', 'c1', 'value', ts1, {ATTRIBUTES=>{'mykey'=>'myvalue'}}

hbase> put 't1', 'r1', 'c1', 'value', ts1, {VISIBILITY=>'PRIVATE|SECRET'}

The same commands also can be run on a table reference. Suppose you had a reference

t to table 't1', the corresponding command would be:

hbase> t.put 'r1', 'c1', 'value', ts1, {ATTRIBUTES=>{'mykey'=>'myvalue'}}

Scan/select all the data from the table

hbase(main):044:0> **scan 'Patient'**

ROW COLUMN+CELL

001 column=Medical:history, timestamp=1557035960386, value=Anemic

001 column=Personal:pname, timestamp=1557035959592, value=Ramesh

002 column=Personal:age, timestamp=1557035960118, value=24

002 column=Personal:filenum, timestamp=1557035959825, value=100

002 column=Personal:pname, timestamp=1557035959755, value=saravanan

003 column=Personal:pname, timestamp=1557035959891, value=gowtham

004 column=Personal:pname, timestamp=1557035959959, value=amudhan

005 column=Personal:pname, timestamp=1557035960035, value=alex

102 column=Medical:history, timestamp=1557035960484, value=Arthritis

102 column=Medical:oldhistory, timestamp=1557035962912, value=Ostophenia

105 column=Medical:history, timestamp=1557035960440, value=General check

105 column=Personal:pname, timestamp=1557035960188, value=alex

202 column=Personal:addr, timestamp=1557035960346, value=3 first ave,NJ

202 column=Personal:age, timestamp=1557035960235, value=24

202 column=Personal:filenum, timestamp=1557035960293, value=101

8 row(s) in 0.2170 seconds

Scan/select with rowkey

hbase(main):046:0> **get 'Patient','002'**

COLUMN CELL

Personal:age timestamp=1557035960118, value=24

Personal:filenum timestamp=1557035959825, value=100

Personal:pname timestamp=1557035959755, value=saravanan

3 row(s) in 0.0930 seconds

####### Retrieve more versions#######

**alter 'Patient',{NAME=>'Personal',VERSIONS=>3}**

Updating all regions with the new schema...

0/1 regions updated.

1/1 regions updated.

Done.

0 row(s) in 4.5990 seconds

**put 'Patient1','001','Personal:pname','Ramesh k'**

**put 'Patient1','001','Personal:pname','k Ramesh'**

**put 'Patient','001','Personal:pname','KR'**

without mentioning version.

hbase(main):007:0> **scan 'Patient'**

ROW COLUMN+CELL

001 column=Medical:history, timestamp=1557035960386, value=Anemic

001 column=Personal:pname, timestamp=1557043797584, value=KR

002 column=Personal:age, timestamp=1557035960118, value=24

002 column=Personal:filenum, timestamp=1557035959825, value=100

002 column=Personal:pname, timestamp=1557035959755, value=saravanan

003 column=Personal:pname, timestamp=1557035959891, value=gowtham

004 column=Personal:pname, timestamp=1557035959959, value=amudhan

005 column=Personal:pname, timestamp=1557035960035, value=alex

102 column=Medical:history, timestamp=1557035960484, value=Arthritis

102 column=Medical:oldhistory, timestamp=1557035962912, value=Ostophenia

105 column=Medical:history, timestamp=1557035960440, value=General check

105 column=Personal:pname, timestamp=1557035960188, value=alex

202 column=Personal:addr, timestamp=1557035960346, value=3 first ave,NJ

202 column=Personal:age, timestamp=1557035960235, value=24

202 column=Personal:filenum, timestamp=1557035960293, value=101

8 row(s) in 0.2150 seconds

After mentioning version.

hbase(main):008:0> scan 'Patient',{VERSIONS => 3}

ROW COLUMN+CELL

001 column=Medical:history, timestamp=1557035960386, value=Anemic

001 column=Personal:pname, timestamp=1557043797584, value=KR

001 column=Personal:pname, timestamp=1557043794939, value=k Ramesh

001 column=Personal:pname, timestamp=1557043794866, value=Ramesh k

002 column=Personal:age, timestamp=1557035960118, value=24

002 column=Personal:filenum, timestamp=1557035959825, value=100

002 column=Personal:pname, timestamp=1557035959755, value=saravanan

003 column=Personal:pname, timestamp=1557035959891, value=gowtham

004 column=Personal:pname, timestamp=1557035959959, value=amudhan

005 column=Personal:pname, timestamp=1557035960035, value=alex

102 column=Medical:history, timestamp=1557035960484, value=Arthritis

102 column=Medical:oldhistory, timestamp=1557035962912, value=Ostophenia

105 column=Medical:history, timestamp=1557035960440, value=General check

105 column=Personal:pname, timestamp=1557035960188, value=alex

202 column=Personal:addr, timestamp=1557035960346, value=3 first ave,NJ

202 column=Personal:age, timestamp=1557035960235, value=24

202 column=Personal:filenum, timestamp=1557035960293, value=101

8 row(s) in 0.1080 seconds

#######List only the latest version#######

hbase(main):009:0> **get 'Patient','001'**

COLUMN CELL

Medical:history timestamp=1557035960386, value=Anemic

Personal:pname timestamp=1557043797584, value=KR

2 row(s) in 0.0970 seconds

|  |
| --- |
| **Concept: available formats**  hbase> t.get 'r1'  hbase> t.get 'r1', {TIMERANGE => [ts1, ts2]}  hbase> t.get 'r1', {COLUMN => 'c1'}  hbase> t.get 'r1', {COLUMN => ['c1', 'c2', 'c3']}  hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}  hbase> t.get 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}  hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}  hbase> t.get 'r1', {FILTER => "ValueFilter(=, 'binary:abc')"}  hbase> t.get 'r1', 'c1'  hbase> t.get 'r1', 'c1', 'c2'  hbase> t.get 'r1', ['c1', 'c2'] |

####### delete a specific column from rowkey#######

hbase(main):017:0> get 'Patient','001'

COLUMN CELL

Medical:history timestamp=1557035960386, value=Anemic

Personal:pname timestamp=1557044317905, value=k Ramesh

2 row(s) in 0.0270 seconds

**delete 'Patient1','002','Personal:pname'**

hbase(main):025:0> get 'Patient','001'

COLUMN CELL

Medical:history timestamp=1557035960386, value=Anemic

1 row(s) in 0.0090 seconds

####### delete entire rowkey details#######

It deletes the entire row.

deleteall 'Patient','001'

####### Add column family #######

**alter 'Patient',{NAME=>'Mechanical'**}

put 'Patient','002','Mechanical:filenum','100'

put 'Patient','003','Mechanical:pname','gowtham'

put 'Patient','004','Mechanical:pname','amudhan'

put 'Patient','005','Mechanical:pname','alex'

put 'Patient','002','Mechanical:age','24'

put 'Patient','105','Mechanical:pname','alex'

put 'Patient','202','Mechanical:age','24'

put 'Patient','202','Mechanical:filenum','101'

put 'Patient','202','Mechanical:addr','3 first ave,NJ'

####### drop the column family from the table####### alter 'Patient1',{NAME=>'Medical',METHOD=>'delete'} ####### Execute the hbase commands from file####### hbase shell /home/hduser/install/hbase\_commands.hbase

Example of keyonlyfilter

This filter does not take any arguments. It returns only the key component of each key-value.

hbase(main):038:0> **scan 'Patient',{ FILTER => "KeyOnlyFilter()"}**

ROW COLUMN+CELL

001 column=Medical:history, timestamp=1557035960386, value=

002 column=Mechanical:age, timestamp=1557045191262, value=

002 column=Mechanical:filenum, timestamp=1557045190932, value=

003 column=Mechanical:pname, timestamp=1557045191061, value=

003 column=Personal:pname, timestamp=1557035959891, value=

004 column=Mechanical:pname, timestamp=1557045191109, value=

004 column=Personal:pname, timestamp=1557035959959, value=

005 column=Mechanical:pname, timestamp=1557045191206, value=

005 column=Personal:pname, timestamp=1557035960035, value=

102 column=Medical:history, timestamp=1557035960484, value=

102 column=Medical:oldhistory, timestamp=1557035962912, value=

105 column=Mechanical:pname, timestamp=1557045191310, value=

105 column=Medical:history, timestamp=1557035960440, value=

105 column=Personal:pname, timestamp=1557035960188, value=

202 column=Mechanical:addr, timestamp=1557045193078, value=

202 column=Mechanical:age, timestamp=1557045191385, value=

202 column=Mechanical:filenum, timestamp=1557045191438, value=

202 column=Personal:addr, timestamp=1557035960346, value=

202 column=Personal:age, timestamp=1557035960235, value=

202 column=Personal:filenum, timestamp=1557035960293, value=

8 row(s) in 0.1440 seconds

####### FirstKeyOnlyFilter#######

####### This filter does not take any arguments. It returns only the first key-value from each row.#######

hbase(main):040:0> **scan 'Patient',{ FILTER => "FirstKeyOnlyFilter()"}**

ROW COLUMN+CELL

001 column=Medical:history, timestamp=1557035960386, value=Anemic

002 column=Mechanical:age, timestamp=1557045191262, value=24

003 column=Mechanical:pname, timestamp=1557045191061, value=gowtham

004 column=Mechanical:pname, timestamp=1557045191109, value=amudhan

005 column=Mechanical:pname, timestamp=1557045191206, value=alex

102 column=Medical:history, timestamp=1557035960484, value=Arthritis

105 column=Mechanical:pname, timestamp=1557045191310, value=alex

202 column=Mechanical:addr, timestamp=1557045193078, value=3 first ave,NJ

8 row(s) in 0.0240 seconds

####### prefixfilter: #######

####### This filter takes one argument a prefix of a row key. It returns only those key-values present in a row that starts with the specified row prefix#######

hbase(main):042:0> **scan 'Patient', {FILTER => "(PrefixFilter ('002'))"}**

ROW COLUMN+CELL

002 column=Mechanical:age, timestamp=1557045191262, value=24

002 column=Mechanical:filenum, timestamp=1557045190932, value=100

1 row(s) in 0.0400 seconds

####### ColumnPrefixFilter - This filter takes one argument a column prefix. It returns only those keyvalues present in a column that starts with the specified column prefix. The column prefix must be of the form qualifier#######

hbase(main):051:0> **scan 'Patient', {FILTER => "(PrefixFilter ('004')) AND ColumnPrefixFilter('p')"}**

ROW COLUMN+CELL

004 column=Mechanical:pname, timestamp=1557045191109, value=amudhan

004 column=Personal:pname, timestamp=1557035959959, value=amudhan

1 row(s) in 0.0210 seconds

####### MultipleColumnPrefixFilter - This filter takes a list of column prefixes. It returns key-values that are present in a column that starts with any of the specified column prefixes. Each of the column prefixes must be of the form qualifier#######

hbase(main):053:0> **scan 'Patient',{FILTER => "MultipleColumnPrefixFilter('p=f','a')"}**

ROW COLUMN+CELL

002 column=Mechanical:age, timestamp=1557045191262, value=24

202 column=Mechanical:addr, timestamp=1557045193078, value=3 first ave,NJ

202 column=Mechanical:age, timestamp=1557045191385, value=24

202 column=Personal:addr, timestamp=1557035960346, value=3 first ave,NJ

202 column=Personal:age, timestamp=1557035960235, value=24

2 row(s) in 0.0470 seconds

####### InclusiveStopFilter - This filter takes one argument a row key on which to stop scanning. It returns all key-values present in rows up to and including the specified row. #######

hbase(main):055:0> **scan 'Patient',{FILTER => "InclusiveStopFilter('003')"}**

ROW COLUMN+CELL

001 column=Medical:history, timestamp=1557035960386, value=Anemic

002 column=Mechanical:age, timestamp=1557045191262, value=24

002 column=Mechanical:filenum, timestamp=1557045190932, value=100

003 column=Mechanical:pname, timestamp=1557045191061, value=gowtham

003 column=Personal:pname, timestamp=1557035959891, value=gowtham

3 row(s) in 0.0300 seconds

####### Selecting columns and introducing limit#######

hbase(main):057:0> **scan 'Patient', { COLUMNS => 'Personal:pname', LIMIT => 2}**

ROW COLUMN+CELL

003 column=Personal:pname, timestamp=1557035959891, value=gowtham

004 column=Personal:pname, timestamp=1557035959959, value=amudhan

2 row(s) in 0.0170 seconds

####### drop the table. Table should be disabled to drop. #######

hbase(main):064:0> **drop 'Patient1'**

ERROR: Table Patient1 is enabled. Disable it first.'

Here is some help for this command:

Drop the named table. Table must first be disabled:

hbase> drop 't1'

hbase> drop 'ns1:t1'

if we need to drop a table, first we need to disable it.

####### Disable table#######

hbase(main):065:0> **describe 'Patient'**

DESCRIPTION ENABLED

'Patient', {NAME => 'Mechanical', BLOOMFILTER => 'ROW', VERSIONS => '1', IN\_ME true

MORY => 'false', KEEP\_DELETED\_CELLS => 'false', DATA\_BLOCK\_ENCODING => 'NONE',

TTL => 'FOREVER', COMPRESSION => 'NONE', MIN\_VERSIONS => '0', BLOCKCACHE => '

true', BLOCKSIZE => '65536', REPLICATION\_SCOPE => '0'}, {NAME => 'Medical', BL

OOMFILTER => 'ROW', VERSIONS => '1', IN\_MEMORY => 'false', KEEP\_DELETED\_CELLS

=> 'false', DATA\_BLOCK\_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'N

ONE', MIN\_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICA

TION\_SCOPE => '0'}, {NAME => 'Personal', BLOOMFILTER => 'ROW', VERSIONS => '3'

, IN\_MEMORY => 'false', KEEP\_DELETED\_CELLS => 'false', DATA\_BLOCK\_ENCODING =>

'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN\_VERSIONS => '0', BLOCKCAC

HE => 'true', BLOCKSIZE => '65536', REPLICATION\_SCOPE => '0'}

1 row(s) in 0.0590 seconds

**disable 'Patient'**

####### Describe the table#######

hbase(main):063:0> **describe 'Patient'**

DESCRIPTION ENABLED

'Patient', {NAME => 'Mechanical', BLOOMFILTER => 'ROW', VERSIONS => '1', IN\_ME false

MORY => 'false', KEEP\_DELETED\_CELLS => 'false', DATA\_BLOCK\_ENCODING => 'NONE',

TTL => 'FOREVER', COMPRESSION => 'NONE', MIN\_VERSIONS => '0', BLOCKCACHE => '

true', BLOCKSIZE => '65536', REPLICATION\_SCOPE => '0'}, {NAME => 'Medical', BL

OOMFILTER => 'ROW', VERSIONS => '1', IN\_MEMORY => 'false', KEEP\_DELETED\_CELLS

=> 'false', DATA\_BLOCK\_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'N

ONE', MIN\_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICA

TION\_SCOPE => '0'}, {NAME => 'Personal', BLOOMFILTER => 'ROW', VERSIONS => '3'

, IN\_MEMORY => 'false', KEEP\_DELETED\_CELLS => 'false', DATA\_BLOCK\_ENCODING =>

'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN\_VERSIONS => '0', BLOCKCAC

HE => 'true', BLOCKSIZE => '65536', REPLICATION\_SCOPE => '0'}

1 row(s) in 0.0610 seconds

####### Enable table####### enable 'Patient1'

#######HBase Pig integeration#######

Create the below table in Hbase

create 'PigDataHTable','UserColFamily'

#######Sample data to upload from Pig#######

user\_01|Arun|30|Chennai

user\_02|Bala|40|Madurai

user\_03|Devi|50|Trichy

user\_04|Faizal|60|Chennai

#######Login to pig in local mode####### pig -x local

userdata = load '/home/hduser/pigdata/hbdata' using PigStorage('|') as (id:chararray, name:chararray, age:chararray, place:chararray);

store userdata into 'hbase://PigDataHTable' using org.apache.pig.backend.hadoop.hbase.HBaseStorage ('UserColFamily:name,UserColFamily:age,UserColFamily:place');

here in the above command, we use a different kind of output format which will be used to store the data directly into hbase tables.

hbase(main):067:0> scan 'PigDataHTable'

ROW COLUMN+CELL

user\_01 column=UserColFamily:age, timestamp=1557049054233, value=30

user\_01 column=UserColFamily:name, timestamp=1557049054233, value=Arun

user\_01 column=UserColFamily:place, timestamp=1557049054233, value=Chennai

user\_02 column=UserColFamily:age, timestamp=1557049054252, value=40

user\_02 column=UserColFamily:name, timestamp=1557049054252, value=Bala

user\_02 column=UserColFamily:place, timestamp=1557049054252, value=Madurai

user\_03 column=UserColFamily:age, timestamp=1557049054253, value=50

user\_03 column=UserColFamily:name, timestamp=1557049054253, value=Devi

user\_03 column=UserColFamily:place, timestamp=1557049054253, value=Trichy

user\_04 column=UserColFamily:age, timestamp=1557049054254, value=60

user\_04 column=UserColFamily:name, timestamp=1557049054254, value=Faizal

user\_04 column=UserColFamily:place, timestamp=1557049054254, value=Chennai

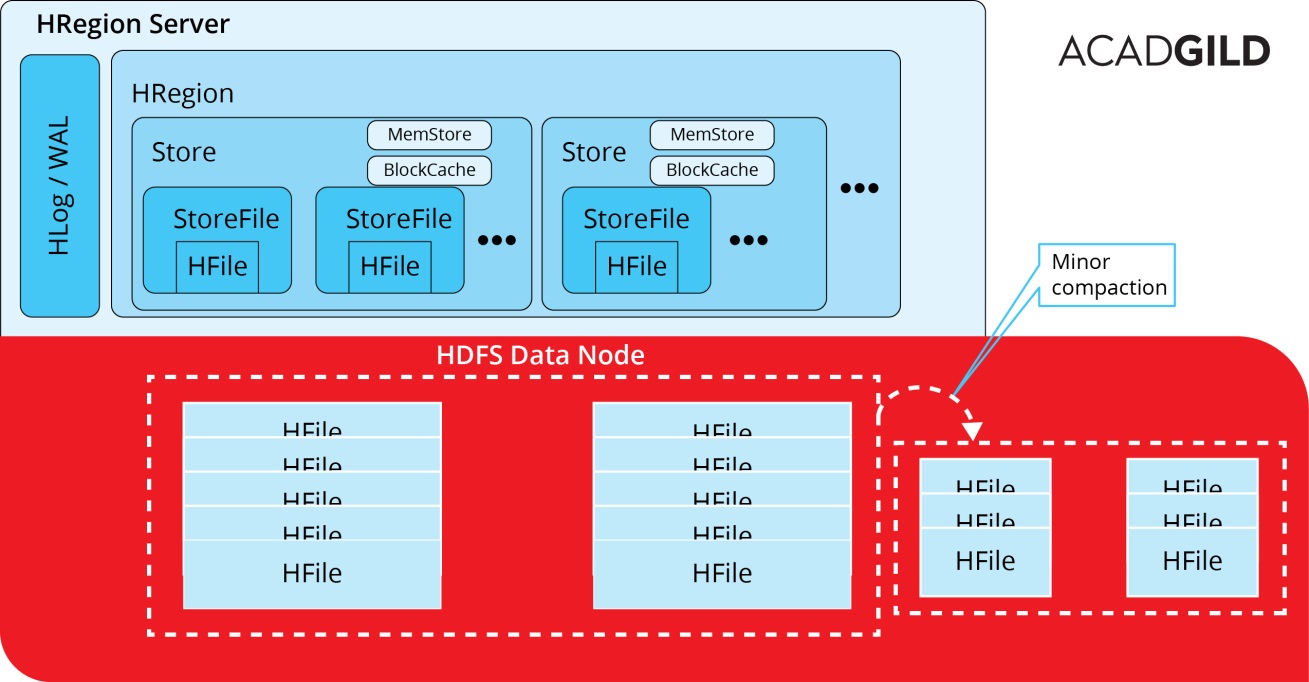
4 row(s) in 0.0600 seconds

############Compaction:#############

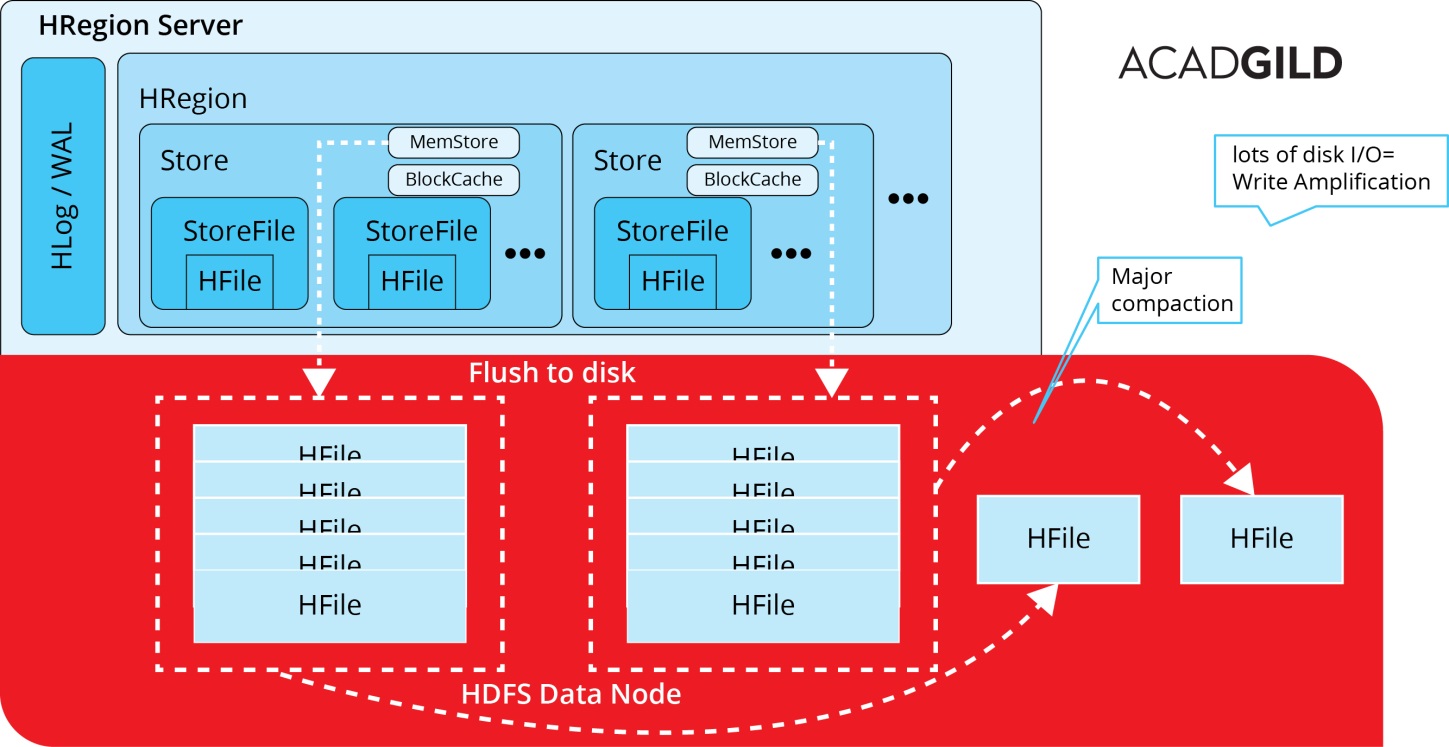
The servers are active all day long and while acting on [Big Data](https://acadgild.com/big-data/big-data-hadoop-administration), HBase hardly gets to write data. Therefore, it breaks the writing process into two parts: Minor Compaction and Major Compaction.  
When the storage area of HBase is all most filled with data, it starts creating compressed files, which occupies less memory.

Here are the various processes involved in Minor Compaction:

* Bigger Hfile are created by combining smaller Hfiles.
* Hfile keeps the deleted file with them.
* Increases space in memory, useful to store more data.
* Merge sorting is used in process.

  
The other way to go around is major compaction

* Data present per column family in one region is accumulated to 1 Hfile.
* During this process, all deleted files or expired cells are deleted permanently
* Increase read performance of newly created Hfile.
* Accepts lots of I/O.
* Possibilities for traffic congestion.
* The Major compaction process is also known as Write Amplification Process.
* This process must be scheduled at a minimum bandwidth of network I/O.



**HBase compaction tuning tips**

**Short Description:**  
How to use some hidden HBase compaction configuration choices to enhance performance and stability of HBase cluster. Below

compact 'Patient2'

major\_compact 'Patient2'

Phoenix

Install Phoenix :

cd /home/hduser/install

tar xvzf phoenix-4.6.0-HBase-0.98-bin.tar.gz

sudo mv phoenix-4.6.0-HBase-0.98-bin /usr/local/phoenix sudo chown -R hduser:hadoop /usr/local/phoenix

# Add below Jar files to hbase lib folder

cd /usr/local/phoenix/

cp phoenix-4.6.0-HBase-0.98-client-minimal.jar /usr/local/hbase/lib/ cp phoenix-core-4.6.0-HBase-0.98.jar /usr/local/hbase/lib/

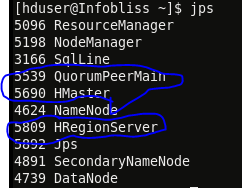
# Before starting Phoenix, make sure HBASE and Zookeeper are running with the below quorums

**from the below pic,**

# **QuorumPeerMain** is the service running for Zookeeper

# **HMaster** is the service for hamster

# **HRegionServer** is the service for Region server.



Use the below command to start the respective services.

* zkServer.sh start – to start
* zookeeper start-hbase.sh – to start hbase

**Phoenix shell**

**To start phoenix command line shell**

sqlline.py localhost

to check tables - !tables

**Phoenix Workouts:**

CREATE TABLE IF NOT EXISTS STOCK\_SYMBOL (SYMBOL VARCHAR NOT NULL PRIMARY KEY, COMPANY VARCHAR);

UPSERT INTO STOCK\_SYMBOL VALUES ('CRM','SalesForce.com');

UPSERT INTO STOCK\_SYMBOL VALUES ('AAPL','APPLE Inc.');

UPSERT INTO STOCK\_SYMBOL VALUES ('CRM','SALESFORCE');

UPSERT INTO STOCK\_SYMBOL VALUES ('GOOG','Google');

UPSERT INTO STOCK\_SYMBOL VALUES ('HOG','Harlet-Davidson Inc.');

UPSERT INTO STOCK\_SYMBOL VALUES ('HPQ','Hewlett Packard');

UPSERT INTO STOCK\_SYMBOL VALUES ('INTC','Intel');

UPSERT INTO STOCK\_SYMBOL VALUES ('MSFT','Microsoft');

UPSERT INTO STOCK\_SYMBOL VALUES ('WAG','Walgreens');

UPSERT INTO STOCK\_SYMBOL VALUES ('WMT','Walmart');

SELECT \* FROM STOCK\_SYMBOL;

CREATE TABLE STOCK\_SYMBOL\_BKP (SYMBOL VARCHAR NOT NULL PRIMARY KEY, COMPANY VARCHAR); UPSERT INTO STOCK\_SYMBOL\_BKP SELECT \* FROM STOCK\_SYMBOL;

Create Updatable View:

create view STOCK\_SYMBOL\_VIEW (SYMBOL VARCHAR, CUR\_DT DATE) AS SELECT \* FROM STOCK\_SYMBOL

WHERE COMPANY LIKE 'W%';

Index Creation:

CREATE INDEX IDX\_STOCK\_SYMBOL ON STOCK\_SYMBOL (SYMBOL) INCLUDE(COMPANY);

Index with Salt bucketing:

CREATE INDEX IDX\_STOCK\_SYMBOL1 ON STOCK\_SYMBOL (SYMBOL DESC, COMPANY) SALT\_BUCKETS=3, DATA\_BLOCK\_ENCODING='NONE';

UPDATE STATISTICS STOCK\_SYMBOL;

# Joins:

select a.SYMBOL ,b. COMPANY

from STOCK\_SYMBOL as a inner join STOCK\_SYMBOL\_BKP as b on a.SYMBOL=b.SYMBOL;

SELECT D.SYMBOL,E.SYMBOL,E.COMPANY

FROM STOCK\_SYMBOL d LEFT OUTER JOIN STOCK\_SYMBOL\_BKP e ON e.SYMBOL = d.SYMBOL;

SELECT D.SYMBOL,E.SYMBOL,E.COMPANY

FROM STOCK\_SYMBOL d RIGHT OUTER JOIN STOCK\_SYMBOL\_BKP e ON e.SYMBOL = d.SYMBOL;

**Non interactive Query execution:**

**Table creation and Bulk loading:**

psql.py localhost /home/hduser/phoenixdata/WEB\_STAT.sql

/home/hduser/phoenixdata/examples/WEB\_STAT.csv

**Perform Webdata analytics:**

psql.py localhost /home/hduser/phoenixdata/WEB\_STAT\_QUERIES.sql