Assignment 10

Hbase

1.What is NoSQL data base?

Ans: NoSQL is an approach to database design that can accomodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stand for "not only SQL," is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data.

NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

Examples of NoSQL DataBase: Cassandra,MongoDB, HBase

2.How does data get stored in NoSQl database?

Ans: Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

- **Key-value data stores:** Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned *and replicated* across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.
- **Document stores:** Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.
- **Wide-column stores:** Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

- **Graph stores:** A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

3.What is a column family in HBase?

Ans: **Columns** in Apache **HBase** are grouped into **column families**. All **column**members of a **column family** have the same prefix. The colon character **(:)** delimits the column family from the column family prefix must be composed of printable characters. The qualifying tail, the column family qualifier, can be made of any arbitrary bytes. Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up and running.
Physically, all column family members are stored together on the filesystem. Because tunings and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics.

For example, the **columns** courses:history and courses:math are both members of the courses **column family**.

Physically, all **column family** members are stored together on the filesystem.

Example:

Row Key	**Column Family: {Column Qualifier:Version:Value}**
00001	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'} ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}
00002	CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe', ContactInfo: { 'SA': 1383859185577:'7 HBase Ave, CA 22222'}

The table shows two column families: CustomerName and ContactInfo. When creating a table in HBase, the developer or administrator is required to define one or more column families using printable characters.

Generally, column families remain fixed throughout the lifetime of an HBase table but new column families can be added by using administrative commands.

4.How many maximum number of columns can be added to HBase table?

Ans: The official recommendation for the number of column families per table is three or less.

There is one MemStore(Its a write cache which stores new data before writing it into Hfiles) per Column Family, when one is full, they all flush.

The more we add column families there will be more MemStore created and Memstore flush will be more frequent. It will degrade the performance.

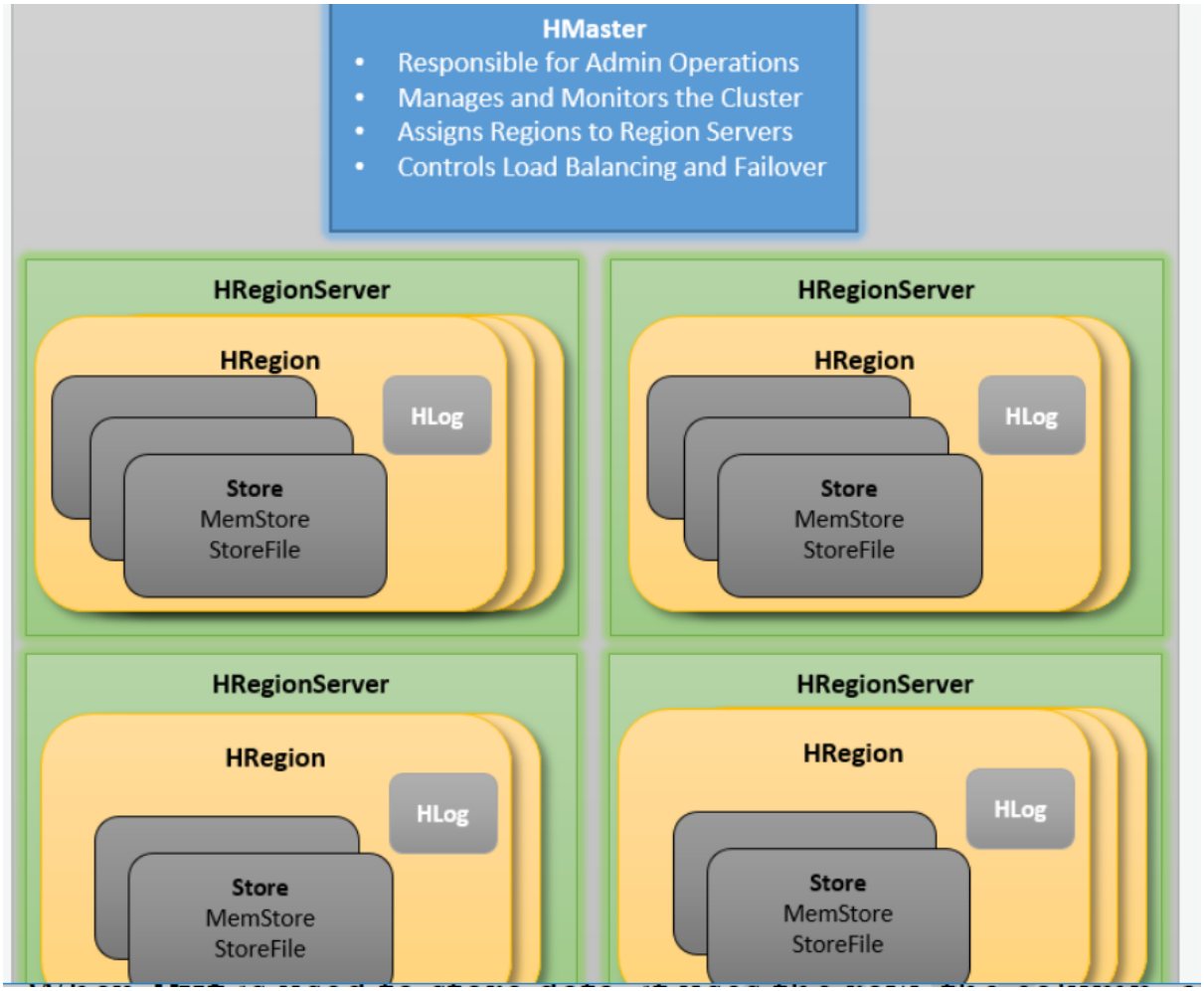
5.Why columns are not defined at the time of table creation in HBase?

Ans: HBase has dynamic schema. It uses query -first schema design. All possible queries are identified first and the schema model is designed accordingly. We are adding columns on need basis at run time unlike in traditional database where we predefine the coulmns. Hence there is a flexibility in HBase that whenever any new requirement comes up we can design the database and add columns according to our current requirement without changing the complete model of the database.

6.How does data get managed in HBase?

Ans: **HBase** stores **data** in a form of a distributed sorted multidimensional persistence maps called Tables. ...**HBase data** model consists of tables containing rows.**Data is organized** into column families grouping columns in each row. Just like in a Relational Database, data in HBase is stored in Tables and these Tables are stored in Regions. When a Table becomes too big, the Table is partitioned into multiple Regions. These Regions are assigned to Region Servers across the cluster. Each Region Server hosts roughly the same number of Regions.

The HBase Physical Architecture consists of servers in a Master-Slave relationship as shown below. Typically, the HBase cluster has one Master node, called HMaster and multiple Region Servers called HRegionServer. Each Region Server contains multiple Regions – HRegions.



7.What happens internally when new data gets inserted into HBase table?

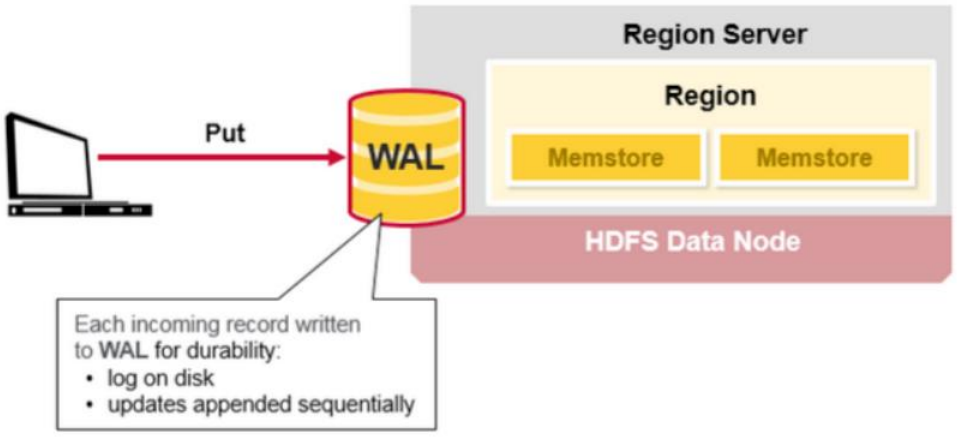
Ans: When we put data into HBase, a timestamp is required. The timestamp can be generated automatically by the RegionServer or can be supplied by you. The timestamp must be unique per version of a given cell, because the timestamp identifies the version. To modify a previous version of a cell, for instance, we issue a Put with a different value for the data itself, but the same timestamp.

HBase Write Steps (1)

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

- Edits are appended to the end of the WAL file that is stored on disk.
- The WAL is used to recover not-yet-persisted data in case a server crashes.

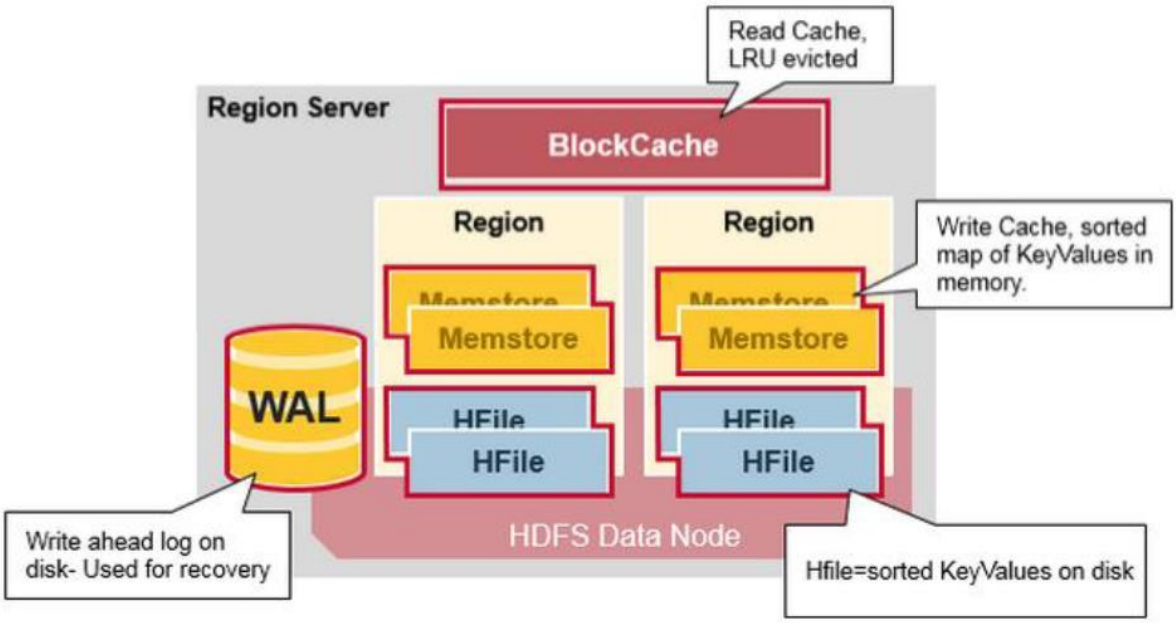
HBase Write Steps (2)



Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.

The READ-WRITE data in the HBASE is managed in the following ways:
•**WAL**: Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.
•**BlockCache**: is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.
•**MemStore**: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
•**Hfiles** store the rows as sorted KeyValues on disk.

All the above are sub components of Region server which manages the data in the HBASE.



Task 2:

a. Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.

b. Add few records in the table and update some of them. Use IP Address as row-key. Scan the table to view if all the previous versions are getting displayed.

Ans:

Commands:

create 'clicks',{NAME=>'hits', VERSIONS=>5}

Explanation :

creates a table named '**clicks**' with column family '**hits**' which would keep five version for corresponding columns values of **hits** column family

OR

create 'clicks','hits'
alter 'clicks',{NAME=>'hits', VERSIONS=>5}

Explanation :

creates a table named '**clicks**' with column family '**hits**'
Alter the table/Modify the table attributes to retains five versions for column values of **hits** column family.

create 'clicks','hits'
alter 'clicks',{NAME=>'hits', VERSIONS=>5}

Explanation :

creates a table named '**clicks**' with column family '**hits**'

Alter the table/Modify the table attributes to retains five versions for column values of **hits** column family.

put 'clicks','IP','hits:hostname','gpu_vdi_01'
put 'clicks','IP','hits:location','India'
put 'clicks','IP','hits:browser','chrome'
Explanation :
Inserting values in columns(hostname,location,browser <in purple>) for column family hits with row-key "**IP**" of table '**clicks**'
scan 'clicks'
Explanation :
Reading values of table clicks. To check if values are inserted.
put 'clicks','IP','hits:browser','firefox'
put 'clicks','IP','hits:browser','safari'
put 'clicks','IP','hits:browser','IExplorer'
put 'clicks','IP','hits:browser','Opera'
put 'clicks','IP','hits:browser','Chromium'
put 'clicks','IP','hits:browser','Netscape'
Explanation :
Changing or **updating** values of **browser** column to check if table retains the last five versions of the updated values.
scan 'clicks'
Explanation:
Although table clicks retains the latest updated value for **browser i.e** 'Netscape'
scan 'clicks',{COLUMN=>'hits:browser',**VERSIONS**=>5}
Explanation:
Versions*=>*5 shows the last five values that were updated/changed for column **browser** in column family **'hits'**
ScreenShot:

```
hbase(main):004:0> create 'clicks', 'hits'
0 row(s) in 1.2740 seconds

=> Hbase::Table - clicks
hbase(main):005:0> alter 'clicks',{NAME=>'hits',VERSIONS=>5}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 1.9880 seconds

hbase(main):006:0> put 'clicks','IP','hits:hostname','gpu_vdi_01'
0 row(s) in 0.1510 seconds

hbase(main):007:0> put 'clicks','IP', 'hits:location','India'
0 row(s) in 0.0170 seconds

hbase(main):008:0> put 'clicks','IP', 'hits:browser','chrome'
0 row(s) in 0.0180 seconds

hbase(main):009:0> scan 'clicks'
ROW                                   COLUMN+CELL
 IP                                   column=hits:browser, timestamp=1534395368349, value=chrome
 IP                                   column=hits:hostname, timestamp=1534395294519, value=gpu_vdi_01
 IP                                   column=hits:location, timestamp=1534395332837, value=India
1 row(s) in 0.0680 seconds

hbase(main):010:0>
hbase(main):011:0* put 'clicks','IP', 'hits:browser','firefox'
0 row(s) in 0.0150 seconds

hbase(main):012:0> put 'clicks','IP', 'hits:browser','safari'
0 row(s) in 0.0040 seconds

hbase(main):013:0> put 'clicks','IP', 'hits:browser','Opera'
0 row(s) in 0.0110 seconds

hbase(main):014:0> put 'clicks','IP', 'hits:browser','Chromium'
0 row(s) in 0.0080 seconds

hbase(main):015:0> put 'clicks','IP', 'hits:browser','Netscape'
0 row(s) in 0.0070 seconds
```

```
hbase(main):008:0> put 'clicks','IP', 'hits:browser','chrome'
0 row(s) in 0.0180 seconds

hbase(main):009:0> scan 'clicks'
ROW                                   COLUMN+CELL
 IP                                   column=hits:browser, timestamp=1534395368349, value=chrome
 IP                                   column=hits:hostname, timestamp=1534395294519, value=gpu_vdi_01
 IP                                   column=hits:location, timestamp=1534395332837, value=India
1 row(s) in 0.0680 seconds

hbase(main):010:0>
hbase(main):011:0* put 'clicks','IP', 'hits:browser','firefox'
0 row(s) in 0.0150 seconds

hbase(main):012:0> put 'clicks','IP', 'hits:browser','safari'
0 row(s) in 0.0040 seconds

hbase(main):013:0> put 'clicks','IP', 'hits:browser','Opera'
0 row(s) in 0.0110 seconds

hbase(main):014:0> put 'clicks','IP', 'hits:browser','Chromium'
0 row(s) in 0.0080 seconds

hbase(main):015:0> put 'clicks','IP', 'hits:browser','Netscape'
0 row(s) in 0.0070 seconds

hbase(main):016:0> scan 'clicks'
ROW                                   COLUMN+CELL
 IP                                   column=hits:browser, timestamp=1534395471811, value=Netscape
 IP                                   column=hits:hostname, timestamp=1534395294519, value=gpu_vdi_01
 IP                                   column=hits:location, timestamp=1534395332837, value=India
1 row(s) in 0.0320 seconds

hbase(main):017:0> scan 'clicks',{COLUMN=>'hits:browser',VERSIONS=>5}
ROW                                   COLUMN+CELL
 IP                                   column=hits:browser, timestamp=1534395471811, value=Netscape
 IP                                   column=hits:browser, timestamp=1534395460073, value=Chromium
 IP                                   column=hits:browser, timestamp=1534395448082, value=Opera
 IP                                   column=hits:browser, timestamp=1534395438242, value=safari
 IP                                   column=hits:browser, timestamp=1534395426471, value=firefox
1 row(s) in 0.0310 seconds

hbase(main):018:0>
```