BIG DATA HADOOP & SPARK TRAINING

-by Rashmi Krishna

Assignment on Hbase Basics

Task 1

Answer in your own words with example.

1.What is NoSQL data base?

A NoSQL provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called ***Not only SQL*** to emphasize that they may support SQL-like query languages.

Motivations for this approach include:

- simplicity of design
- simpler horizontal scaling to clusters of machines
- finer control over availability.

The data structures used by NoSQL databases e.g. key-value, wide column, graph, or document are different from those used by default in relational databases, make some operations faster in NoSQL. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables. Many NoSQL stores compromise consistency in favor of availability, partition tolerance, and speed.

Barriers to the greater adoption of NoSQL stores include:

1. the use of low-level query languages
2. lack of standardized interfaces,
3. huge previous investments in existing relational databases

Most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes, so queries for data might not return updated data immediately or might result in reading data that is not accurate, this problem is known as ***stale reads.***

Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss. Some NoSQL systems provide concepts such as write-ahead logging to avoid data loss.

2.How does data get stored in NoSQl database?

There are various NoSQL Databases. Each one uses a different method to store data. Some might use column store, Key-Value store, document store, graph, etc., Each database has its own unique characteristics. Which are as follows:

Key-Value Store:

- Key-value (KV) stores use the associative array i.e. as a map or dictionary as their fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.

- The key-value model can be extended to a discretely ordered model that maintains keys in lexicographic order. This extension is computationally powerful, in that it can efficiently retrieve selective key ranges

Document Store:

- The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings.

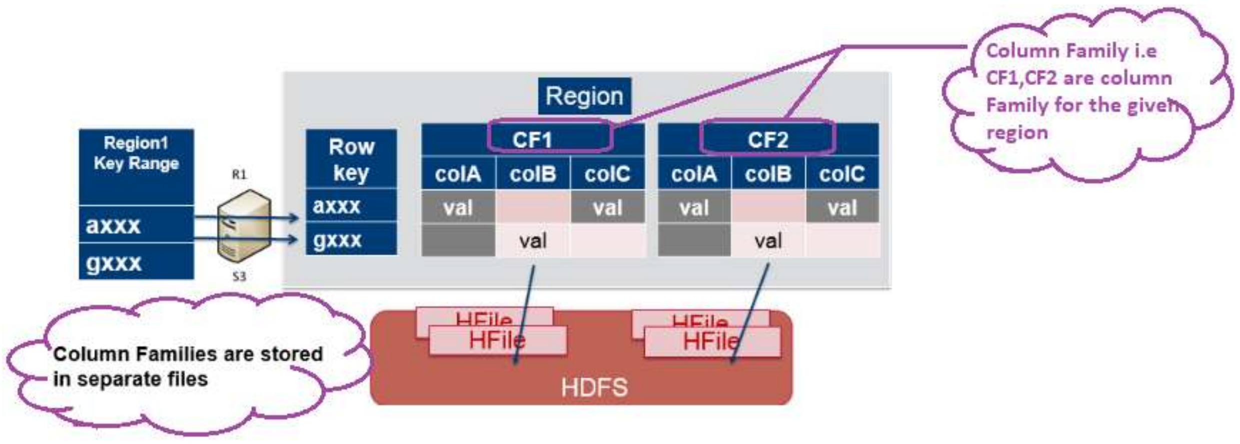
Graph:

This kind of database is designed for data whose relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them. The type of data could be social relations, public transport links, road maps, network topologies, etc

3.What is a column family in HBase?

In the HBase data model columns are grouped into column families, which must be defined up front during table creation. Column families are stored together on disk, which is why HBase is referred to as a column-oriented data store.

All column members of a column family have the same prefix. For example, the columns CF1: colA, colB, colC and CF2: colA, colB, colC . The colon character (:) delimits the column family. 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 Family i.e CF1,CF2 are column Family for the given region
Region
Region1 Key Range
R1
S3
axxx
gxxx
Row key
axxx
gxxx
CF1
CF2
colA
colB
colC
val
HFile
HDFS
Column Families are stored in separate files

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

There is no hard limit to number of columns in HBase , we can have more than 1 million columns but usually three column families are recommended (not more than three).

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

Column qualifiers need not be specified in advance, because Column qualifiers need not be consistent between rows. Like row keys, column qualifiers do not have a data type and are always treated as a byte[ ].

Column qualifiers are dynamic and can be defined at write time. They are stored as byte[ ] so you can even put data in them.

6.How does data get managed in HBase?

Hbase is the open source implementation of Google's Big Table database, which is where Google stores data for, for example, Google Earth and web index data.

HBase is a structured noSQL database that rides atop Hadoop. You can store Hbase data in the HDFS . And you can also store HBase data in Amazon S3, which has an entirely different architecture. HBase is structured because it has the row-and-column structure of an RDBMS, like Oracle. But it a column-oriented database and not a row-oriented one.

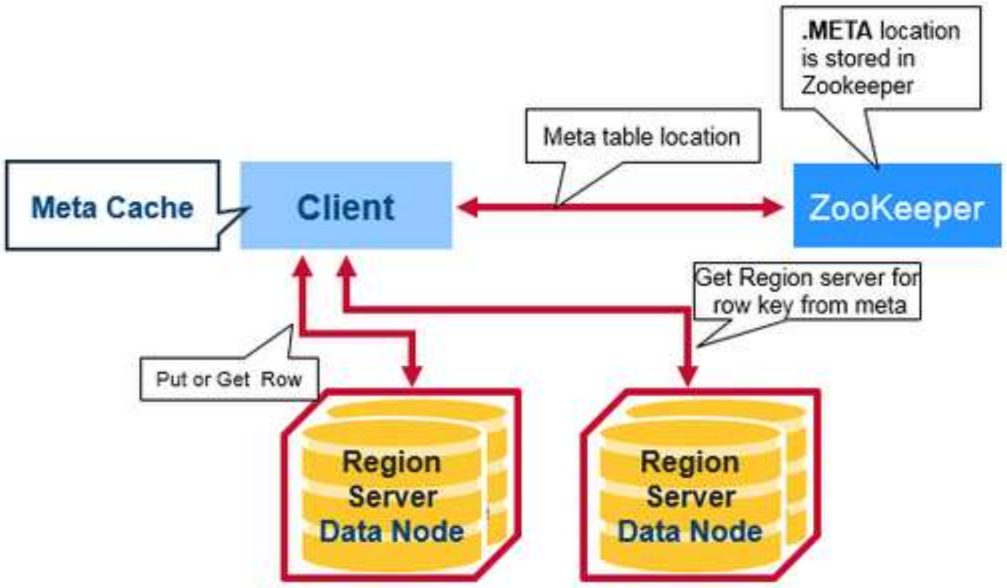
What HBase does is provide random access to big data. Hadoop does not: it is a batch system that only writes files but does not update them.

HBase stores data in a memory table and then flushes it to storage in massive writes to disk. So that gives it the high throughput needed for big data applications.

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

This is what happens the first time a client reads or writes to HBase:

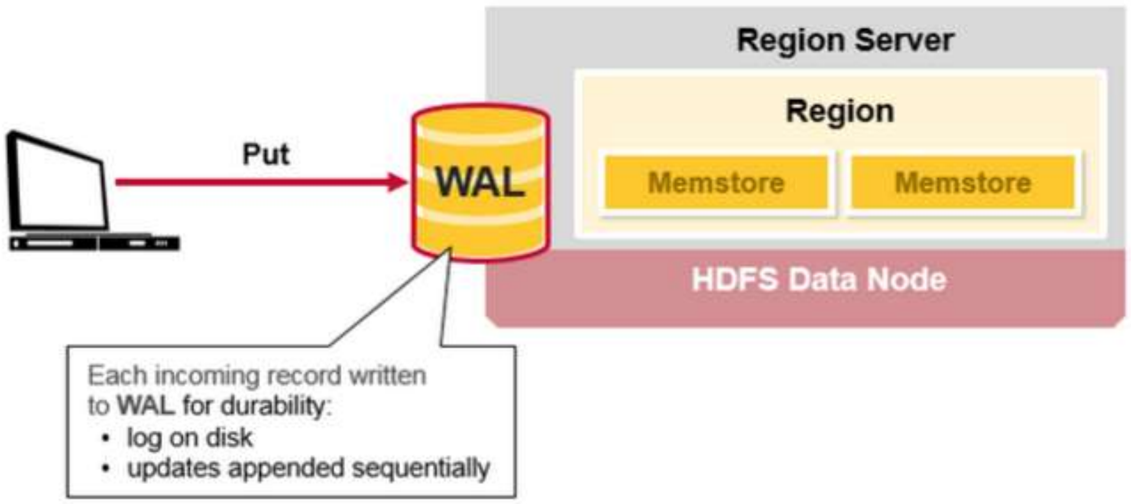
1. The client gets the Region server that hosts the META table from ZooKeeper.
2. The client will query the .META table. Server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
3. It will get the Row from the corresponding Region Server.



HBase Write Steps (1)

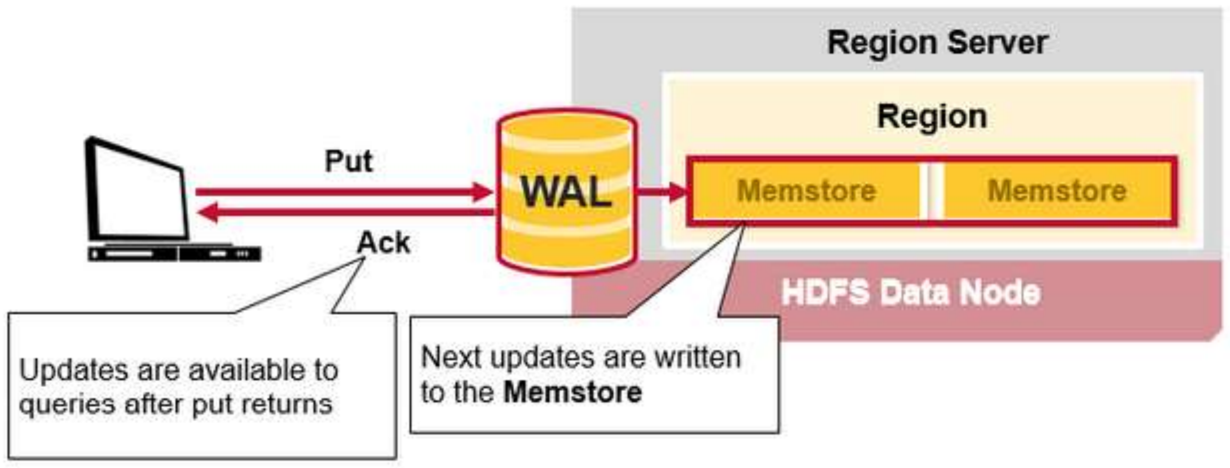
When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

- 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.


Region Server
Region
Put
WAL
Memstore
Memstore
HDFS Data Node
Each incoming record written to WAL for durability:
• log on disk
• updates appended sequentially

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.


Region Server
Region
Put
WAL
Memstore
Memstore
Ack
HDFS Data Node
Updates are available to queries after put returns
Next updates are written to the Memstore

Task 2

1. 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.

This can be performed using the command: this is creating a table called “clicks” and column family “hits”.

create 'clicks','hits'

2. 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.

The above task can be performed using the following command:

“put” command inserts the data, as shown below:-

```
put 'clicks', '172.16.254.1 ', 'hits:c1', '1'

put 'clicks', '172.18.234.1 ', 'hits:c1', '10'
```

put 'clicks', '172.14.259.1 ', 'hits:c2', '33'

put 'clicks', '172.12.284.1 ', 'hits:c2', '27'

put 'clicks', '172.18.224.1 ', 'hits:c2', '19'

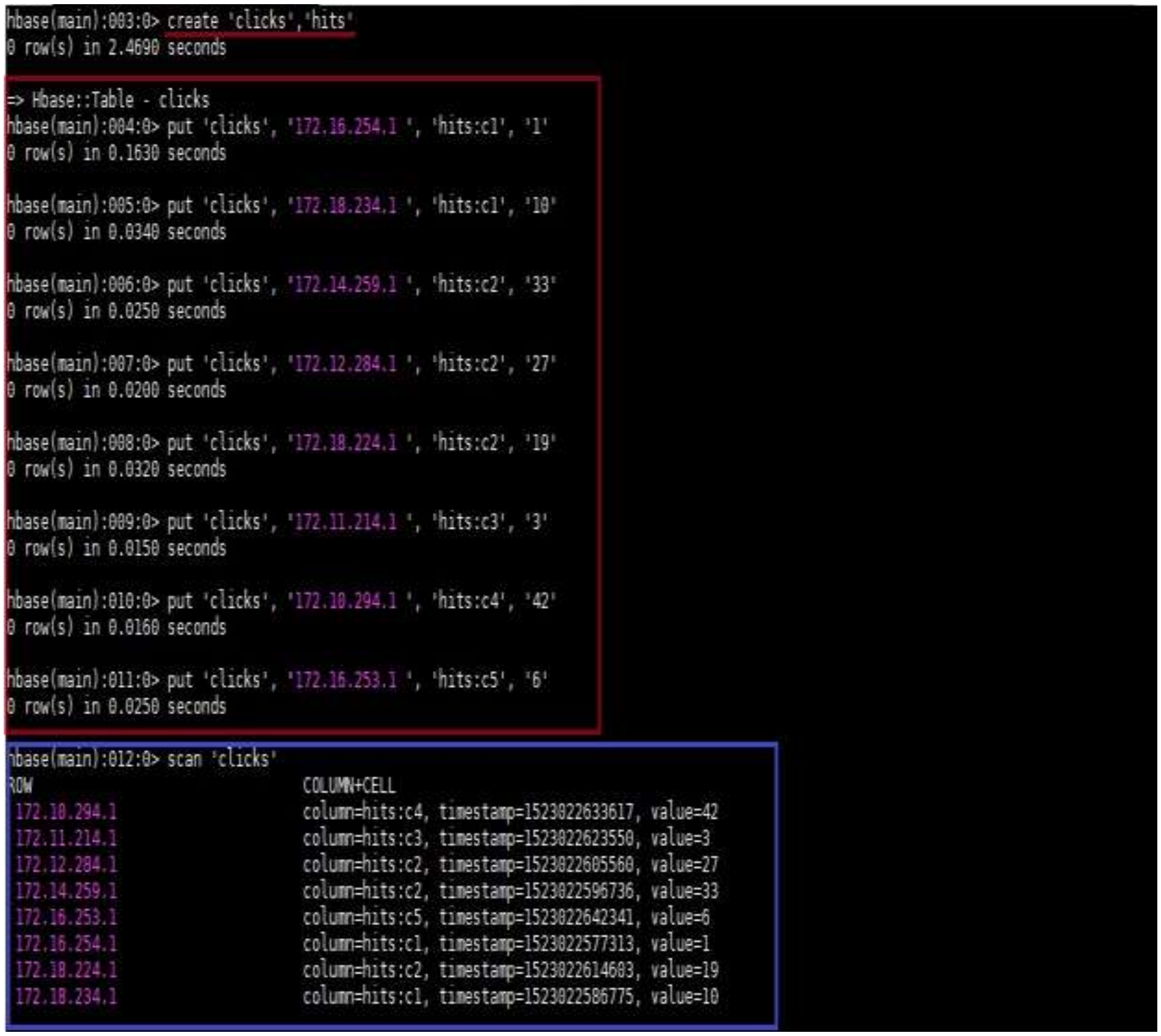
put 'clicks', '172.11.214.1 ', 'hits:c3', '3'

put 'clicks', '172.10.294.1 ', 'hits:c4', '42'

put 'clicks', '172.16.253.1 ', 'hits:c5', '6'

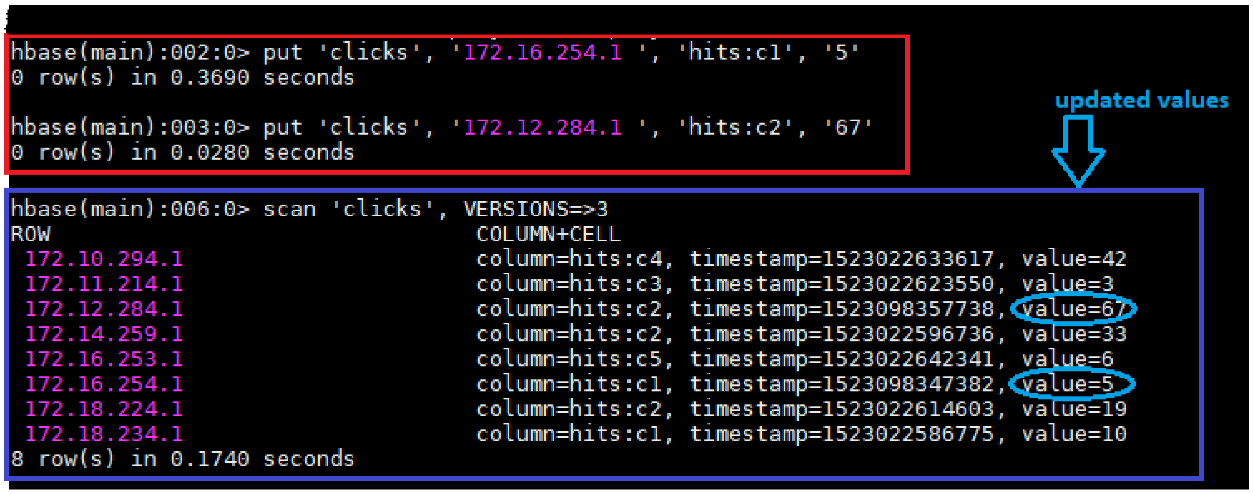
To read the data from the table, use the scan command, as shown below: -

scan 'clicks'



```
hbase(main):003:0> create 'clicks','hits'
0 row(s) in 2.4690 seconds

=> Hbase::Table - clicks
hbase(main):004:0> put 'clicks', '172.16.254.1 ', 'hits:c1', '1'
0 row(s) in 0.1630 seconds

hbase(main):005:0> put 'clicks', '172.18.234.1 ', 'hits:c1', '10'
0 row(s) in 0.0340 seconds

hbase(main):006:0> put 'clicks', '172.14.259.1 ', 'hits:c2', '33'
0 row(s) in 0.0250 seconds

hbase(main):007:0> put 'clicks', '172.12.284.1 ', 'hits:c2', '27'
0 row(s) in 0.0200 seconds

hbase(main):008:0> put 'clicks', '172.18.224.1 ', 'hits:c2', '19'
0 row(s) in 0.0320 seconds

hbase(main):009:0> put 'clicks', '172.11.214.1 ', 'hits:c3', '3'
0 row(s) in 0.0150 seconds

hbase(main):010:0> put 'clicks', '172.10.294.1 ', 'hits:c4', '42'
0 row(s) in 0.0160 seconds

hbase(main):011:0> put 'clicks', '172.16.253.1 ', 'hits:c5', '6'
0 row(s) in 0.0250 seconds

hbase(main):012:0> scan 'clicks'
ROW                            COLUMN+CELL
 172.10.294.1                  column=hits:c4, timestamp=1523022633617, value=42
 172.11.214.1                  column=hits:c3, timestamp=1523022623550, value=3
 172.12.284.1                  column=hits:c2, timestamp=1523022605560, value=27
 172.14.259.1                  column=hits:c2, timestamp=1523022596736, value=33
 172.16.253.1                  column=hits:c5, timestamp=1523022642341, value=6
 172.16.254.1                  column=hits:c1, timestamp=1523022577313, value=1
 172.18.224.1                  column=hits:c2, timestamp=1523022614603, value=19
 172.18.234.1                  column=hits:c1, timestamp=1523022586775, value=10
```

Updating the data and fetching data based on versions:



```
hbase(main):002:0> put 'clicks', '172.16.254.1 ', 'hits:c1', '5'
0 row(s) in 0.3690 seconds

hbase(main):003:0> put 'clicks', '172.12.284.1 ', 'hits:c2', '67'
0 row(s) in 0.0280 seconds

hbase(main):006:0> scan 'clicks', VERSIONS=>3
ROW                                COLUMN+CELL
 172.10.294.1                      column=hits:c4, timestamp=1523022633617, value=42
 172.11.214.1                      column=hits:c3, timestamp=1523022623550, value=3
 172.12.284.1                      column=hits:c2, timestamp=1523098357738, value=67
 172.14.259.1                      column=hits:c2, timestamp=1523022596736, value=33
 172.16.253.1                      column=hits:c5, timestamp=1523022642341, value=6
 172.16.254.1                      column=hits:c1, timestamp=1523098347382, value=5
 172.18.224.1                      column=hits:c2, timestamp=1523022614603, value=19
 172.18.234.1                      column=hits:c1, timestamp=1523022586775, value=10
8 row(s) in 0.1740 seconds
```