Session 10 - HBASE BASICS

Assignment 1

Task 1:

1. What is NoSQL database?

NoSQL databases are nonrelational databases that are optimized for scalable performance and schemaless data models.

Some features of NoSQL databases are

- their ease of development
- low latency
- resilience.

They use a variety of data models, including columnar, document, graph, and in-memory key-value stores.

2. How does data get stored in NoSQl database?

There are 4 different types of NoSQL databases based on how the data is stored, accessed, and structured, and they are optimized for different use cases and applications.

- **Columnar databases** are optimized for reading and writing columns of data as opposed to rows of data. Column-oriented storage for database tables is an important factor in query performance because it drastically reduces the overall disk I/O requirements and reduces the amount of data you need to load from disk.

  Examples : HBase, Cassandra

- **Document databases** are designed to store semi-structured data as documents, typically in JSON or XML format. Unlike traditional relational databases, the schema for each NoSQL document can vary, giving you more flexibility in organizing and storing application data and reducing storage required for optional values.

Examples : CouchDB, MongoDB, DynamoDB

- **Graph databases** store vertices and directed links called edges. Graph databases can be built on both SQL and NoSQL databases. Vertices and edges can each have properties associated with them.

  Examples : Apache Giraph,MarkLogic, Virtuoso

- **In-memory key-value** stores are NoSQL databases optimized for read-heavy application workloads (such as social networking, gaming, media sharing, and Q&A portals) or compute-intensive workloads (such as a recommendation engine). In-memory caching improves application performance by storing critical pieces of data in memory for low-latency access.

  Examples : DynamoDB, Apache Ignite

3. What is a column family in HBase?

HBase tables are organized by column, rather than by row. The columns are organized in groups called column families. These column families are the logical and physical grouping of columns. When creating a HBase table, we must define the column families before inserting any data.

Column families are the base storage mechanism in HBase. A HBase table is comprised of one or more column families, each of which is stored in a separate set of regionfiles sharing a common key. To express it in terms of an RDBMS, a column family is roughly analogous to a RDBMS table with the rowkey as a clustered primary key index.

"CustomerName": {"FirstName": "John",
"LastName": "Smith",
"MiddleName": "Timothy",
"MiddleNameInitial": "T"},
"ContactInfo": {"EmailAddress": "John.Smith@xyz.com",
"ShippingAddress": "1 Hadoop Lane, NY11111"}

The table shows two column families: CustomerName and ContactInfo.

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?

HBase is generally used in the case of wide tables I.e., the tables with large number or columns. So, defining such a huge number of column names at the table creation is not feasible. Therefore, the column families are included in the table creation syntax and not the individual column names.

In the case where there are a limited number of columns, a hive table can be used to query, we need not go to Hbase.

6. How does data get managed in HBase?

HBase is a columnar database, so all data is stored into tables with rows and columns similar to relational database management systems (RDBMSs). The intersection of a row and a column is called a cell. One important difference between HBase tables and RDBMS tables is **versioning**.

Each cell value includes a “version” attribute, which is nothing more than a timestamp uniquely identifying the cell. Versioning tracks changes in the cell and makes it possible to retrieve any version of the contents should it become necessary. HBase stores the data in cells in decreasing order (using the timestamp), so a read will always find the most recent values first.

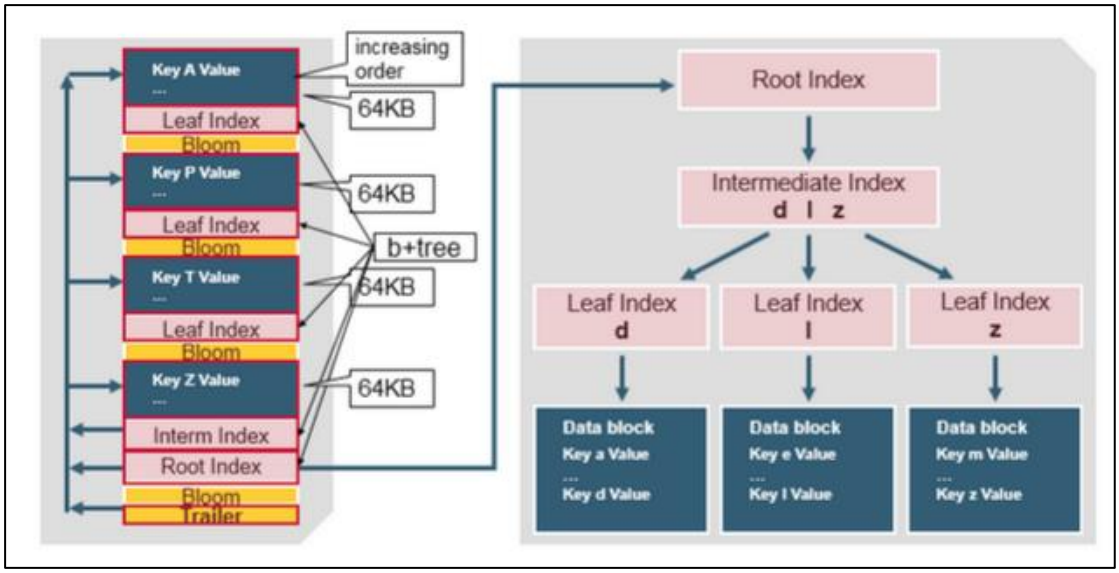
Columns in HBase belong to a column family. The column family name is used as a prefix to identify members of its family. For example, fruits:apple and fruits:banana are members of the fruits column family. HBase implementations are tuned at the column family level, so it is important to be mindful of how you are going to access the data and how big you expect the columns to be.

The rows in HBase tables also have a key associated with them. The structure of the key is very flexible. It can be a computed value, a string, or even another data structure. The key is used to control access to the cells in the row, and they are stored in order from low value to high value.

All of these features together make up the schema. The schema is defined and created before any data can be stored. Even so, tables can be altered and new column families can be added after the database is up and running. This extensibility is extremely useful when dealing with big data because you don’t always know about the variety of your data streams.

Storage Structure in HBase (HFile) :

Data is stored in an HFile which contains sorted key/values. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.



An HFile contains a multi-layered index which allows HBase to seek to the data without having to read the whole file. The multi-level index is like a b+tree:

- Key value pairs are stored in increasing order
- Indexes point by row key to the key value data in 64KB “blocks”
- Each block has its own leaf-index
- The last key of each block is put in the intermediate index
- The root index points to the intermediate index

The trailer points to the meta blocks, and is written at the end of persisting the data to the file. The trailer also has information like bloom filters and time range info. Bloom filters help to skip files that do not contain a certain row key. The time range info is useful for skipping the file if it is not in the time range the read is looking for.

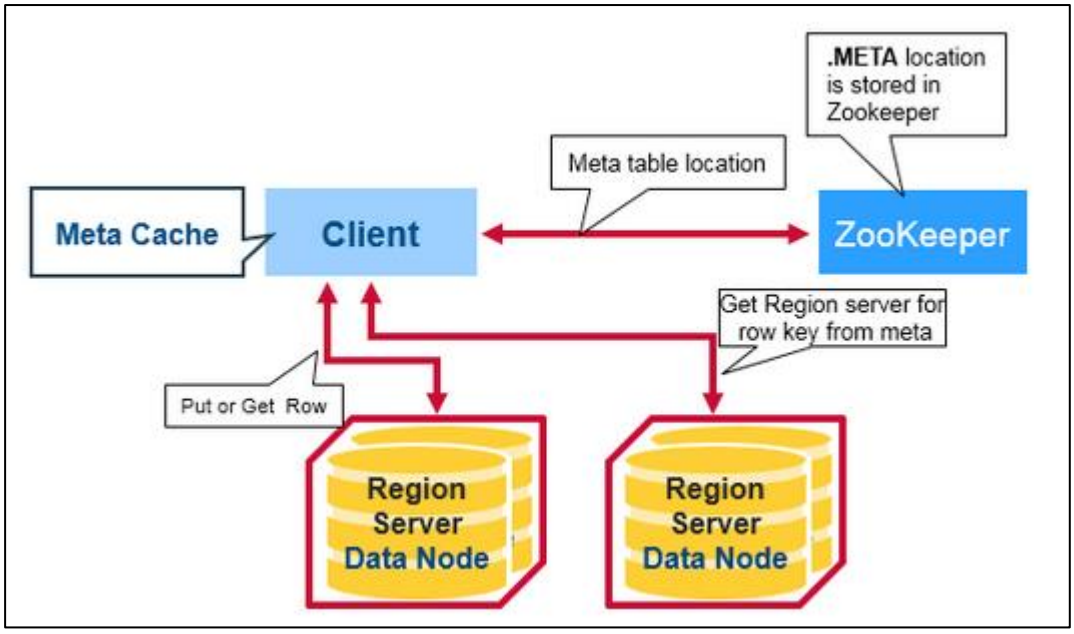
7. What happens internally when new data gets inserted into HBase table?

There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster. ZooKeeper stores the location of the META 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. 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.

For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

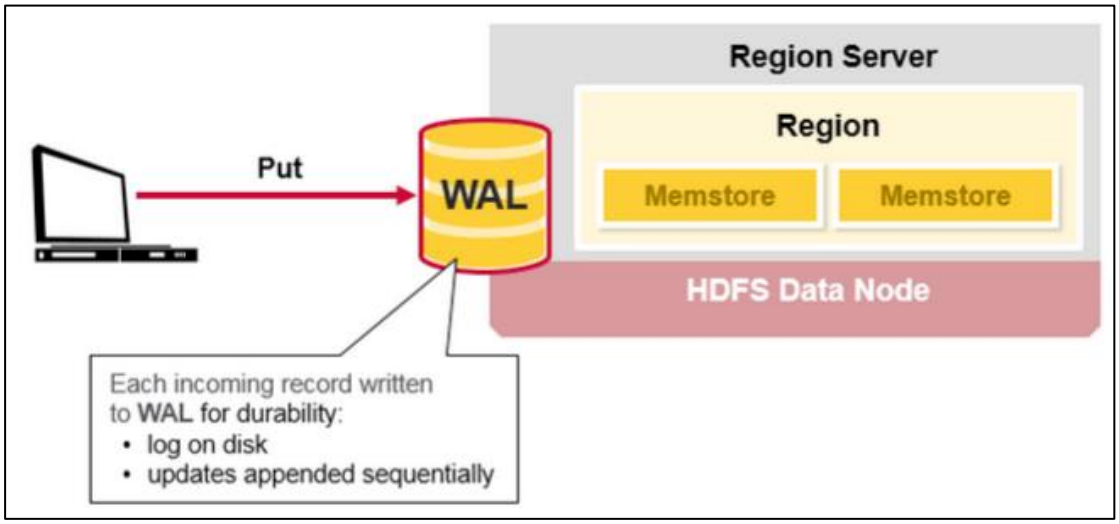


Below are the steps regarding what exactly happens in the Region Server when a put request is issued.

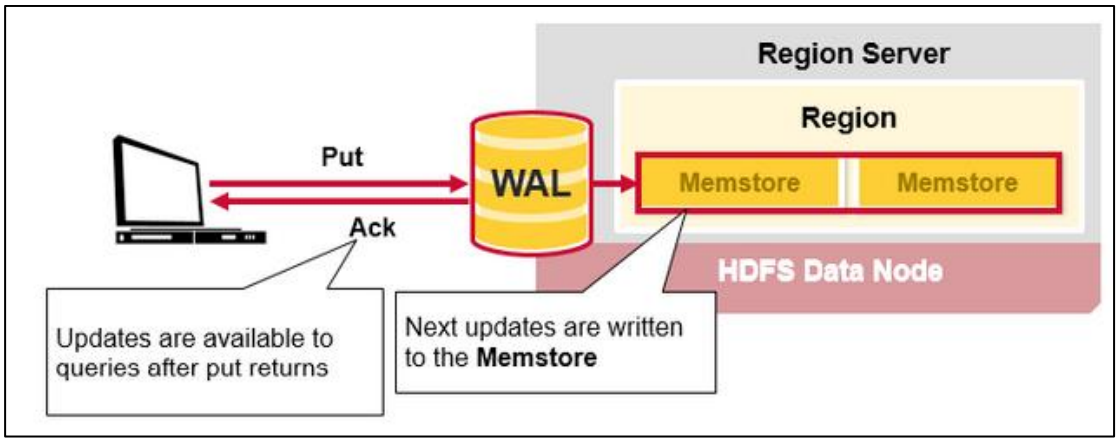
Step 1 : When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL in the Region Server:

- Edits are appended to the end of the WAL file that is stored on disk.

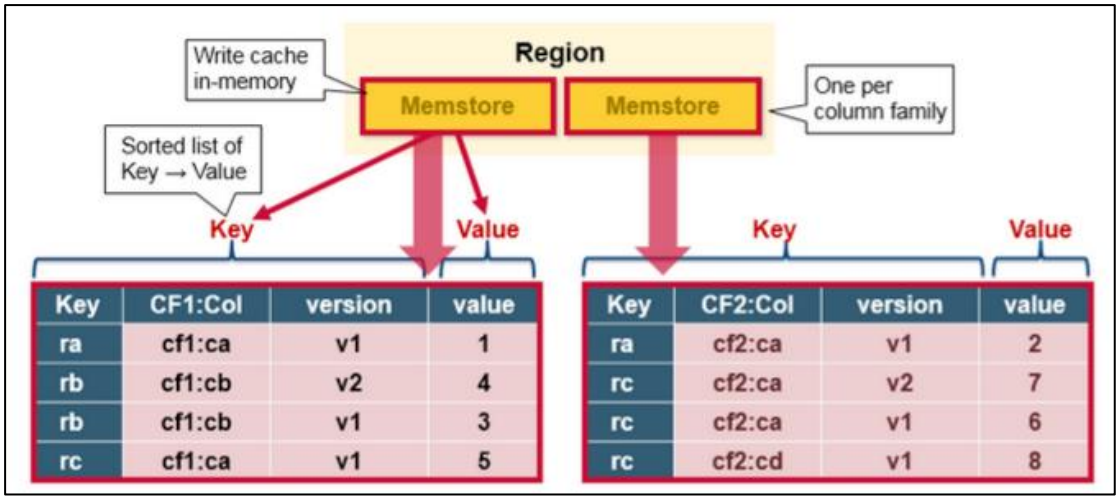
- The WAL is used to recover not-yet-persisted data in case a server crashes.



Step 2 : Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.



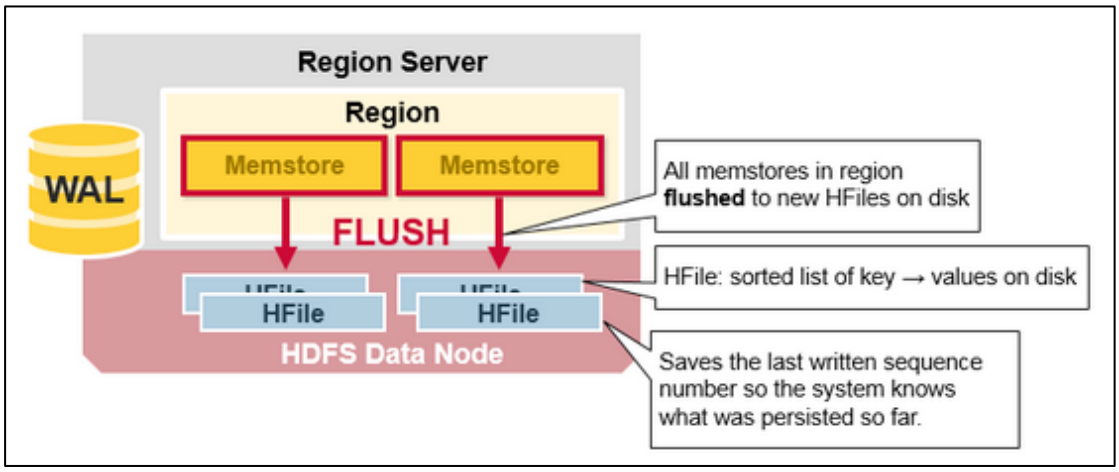
Step 3 : The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.



Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

Step 4 : When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS. HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances. These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.

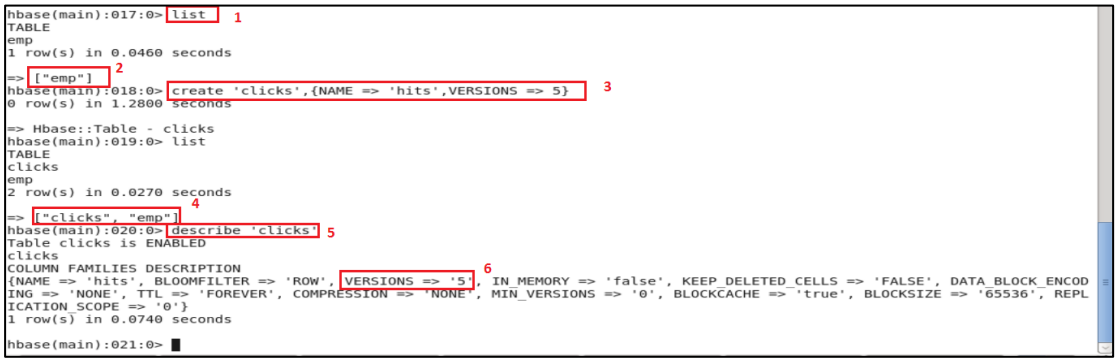


Task 2 :

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.

Command to create table :

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



With reference to the screenshot above,

1 : list command gives the list of existing tables in HBase.

2 : When list is issued, we cannot see the ‘clicks’ table in the list

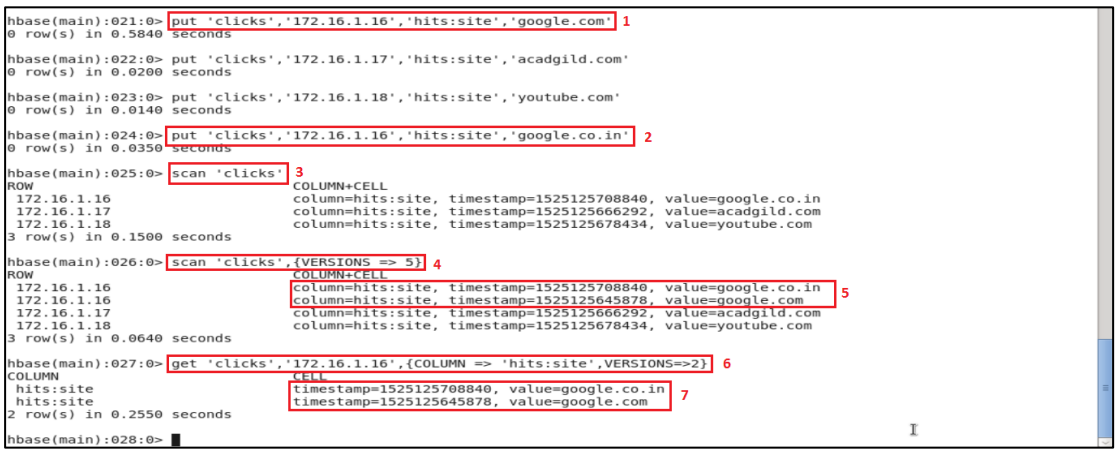
3 : create table command is issued.

4 : When the list command is issued now, ‘clicks’ table name is present in the result.

5 : describe command is used to give out the details of the table.

6 : In the describe section of ‘clicks’ table, we can see that the versions for the ‘hits’ column family is 5, meaning that the columns of this family can store 5 versions of values.

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.



With reference to the screenshot above,

1 : to insert data into Hbase table, following command is issued.

put 'clicks','172.16.1.16','hits:site','google.com'

clicks - table name

172.16.1.16 - row_key in the format of IP address.

hits:site -hits is the column family and site is the column name in the hits column family.

google.com - value assigned to the column.

2 : when a put command is issued with the same row-key and the same column name, the existing value is updated and the old value is stored as a version, based on the version settings for that column family.

3 : scan command is used to see the content values in the table.

4 : scan command along with the versions parameter shows upto those many versions of each record, if they exist.

5 : The 2 values of the record with row-key 172.16.1.16 are displayed.

6 : get command can also be used for seeing the contents of a particular row-key.

get 'clicks','172.16.1.16',{COLUMNS => 'hits:site', VERSIONS => 2}

Clicks - table name

172.16.1.16 - row-key of the record whose values are to be displayed.

7 : the versions of values for that particular rowkey are displayed.