Big Data And Hadoop

Session 17 – Assignment 1

Problem statement:

Answer the following questions:

1. **What is NoSQL data base?**

Answer: a. A NoSQL (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

b. NoSQL databases are increasingly used in big data and real-time web applications.

c. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages.

d. The data model of these databases supports heterogeneous containers, including sets, maps, and arrays.

e. NoSQL analytics systems support runtime type identification and conversion so that custom business logic can be used to dictate analytic treatment of variation.

f. Data is stored in single tables as compared to joining multiple tables.

g. They are highly distributable and hence, can store and process a set of information on more than one devices.

h. Different types of NoSQL databases include document based databases (Eg. MongoDB), graph based (Eg. Neo4j), etc.

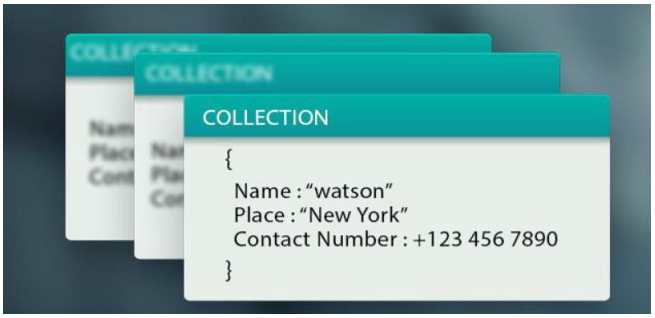
i. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

2. **How does data get stored in NoSQL data base?**

Answer: There are various NoSQL Databases. Each one uses a different method to store data. Some might use column store, some document, some graph, etc., Each database has its own unique characteristics.

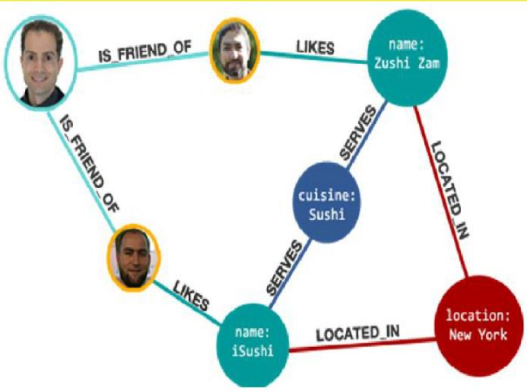
a. Document databases:
   In document-based databases, each key is paired with a complex data structure called document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents. They support embedded documents as well. These databases

consume more space as compared to other databases. Each document may contain diverse and heterogeneous fields.



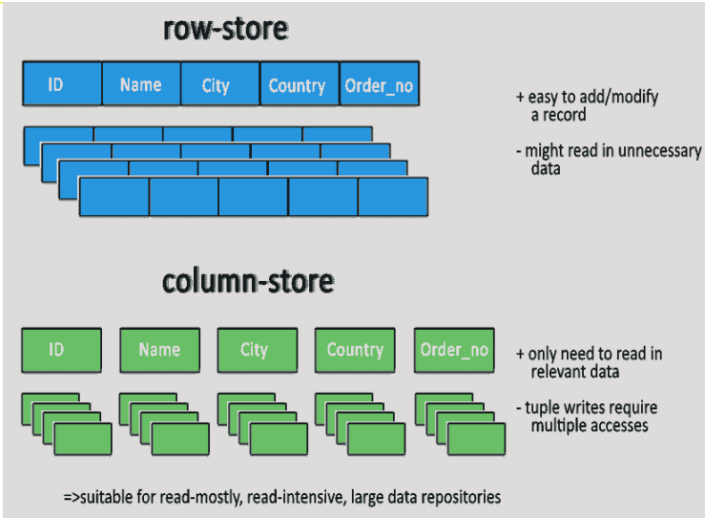
An example of document based database is Mongo DB.

b. Graph based database:
These are used to store information about networks of data, such as social connections. These are best suited for connected data. Not very well suited for all sets of problems.



Examples of these databases include Neo4j, InfoGrid, etc.

c. Wide column stores:
These store columns of data together, instead of rows. These work very well for lookups on a single field. However, lookups on other fields is not supported.



Examples of these databases are Casanndra and HBase.

3. What is column family in HBase?

Answer:

a. Columns in Apache HBase are grouped into *column families*. All column members of a column family have the same prefix. For example, the columns *courses:history* and *courses:math* are both members of the *courses* column family.
b. The colon character (:) delimits the column family from the column family qualifier. The column family prefix must be composed of *printable* characters.
c. The qualifying tail, the column family *qualifier*, can be made of any arbitrary bytes.
d. 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.
e. Physically, all column family members are stored together on the file system.
f. Consider the following example:

Logical View of Customer Contact Information in HBase

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.

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

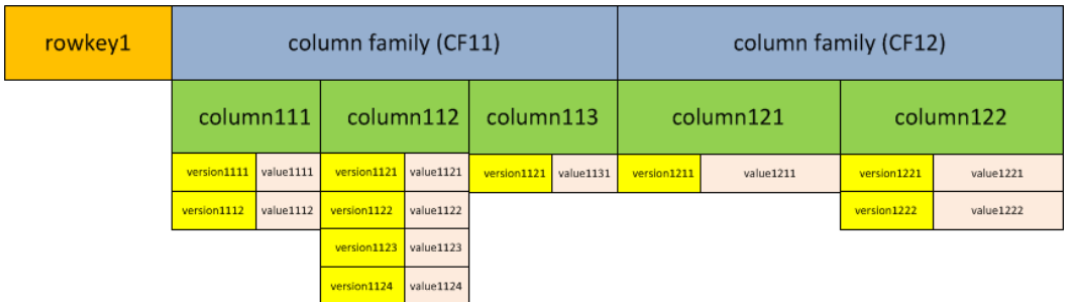
Answer: There is not really a limit on the number of columns that can be added to HBase table. However, there are a few things we should consider.

a. As columns are grouped together logically into column families, HBase currently does not do well with anything above two or three column families. So we should try to keep the number of column families in your schema low. However, a column family can have arbitrary number of columns in it.
b. Currently, flushing and compactions are done on a per Region basis so if one column family is carrying the bulk of the data bringing on flushes, the adjacent families will also be flushed even though the amount of data they carry is small. When many column families exist the flushing and compaction interaction can make for a bunch of needless i/o.
c. When you do an operation within a row, the RegionServer code briefly holds a lock on that row while applying the mutation.
   On the plus side, this means that you can act atomically on several columns - concurrent readers will either see the entire update or won't see the update at all. They shouldn't (barring one or two bugs we're still stomping on) see a partial update.

On the minus side, this means that the throughput of write operations within a single row is limited (probably a few hundred per second).

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

Answer: In HBase, the data is stored in the format that contains a row key and a number of column families that contain multiple columns in them.



The column families need to be defined beforehand, before they are used. However, we need not define the columns during the creation of the table. This provides a flexibility to the user as Hbase is used for big data and most of the times, this data is semi-structured or unstructured. Hence, we may not have the complete view of the data that can arrive in future. The columns can be added to the schema on the fly as and when needed.

Also, different rows can have different column families and hence different columns. So the schema is not the uniform for all the rows.

6. How does data get managed in HBase?

Answer:

a. HBase stores data in a form of a distributed sorted multidimensional persistence maps called Tables. HBase is designed to manage tables with billions of rows and millions of columns.
b. HBase data model consists of tables containing rows. Data is organized into column families grouping columns in each row.
c. The HBase table contains the following:
    - row key maps to a list of column families
    - column family maps to a list of column qualifiers (columns)
    - column qualifier maps to a list of timestamps (versions)
    - timestamp maps to a value (the cell itself)
d. If you are retrieving data that a row key maps to, you'd get data from all column families related to the row that the row key identifies
e. If you are retrieving data which a particular column family maps to, you'd get all column qualifiers and associated data (maps with timestamps as keys and corresponding values)
f. If you are retrieving data that a particular column qualifier maps to, you'd get all timestamps (versions) for that column qualifier and all associated values.
g. Tables are declared up front at schema definition time. Row keys are arrays of bytes and they are lexicographically sorted with the lowest order appearing first.
h. HBASE returns the latest version of data by default but you can ask for multiple versions in your query. HBase returns data sorted first by the row key values, then by column family,

column qualifier and finally by the timestamp value, with the most recent data returned first.

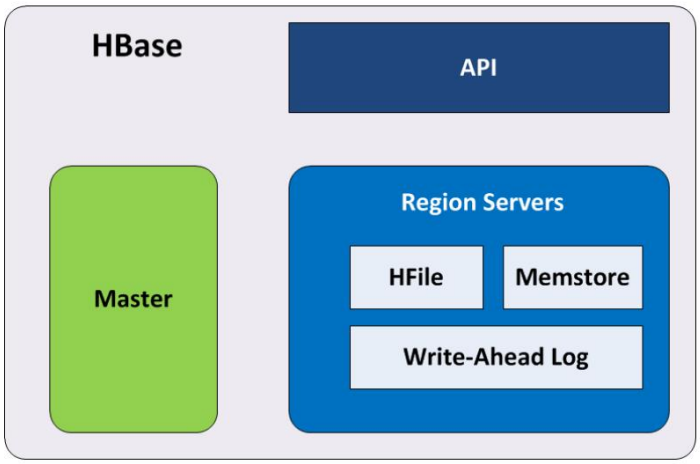
Example of the data is :

```
"rowkey1": {"cf11": {"column111": {"version1111": value1111,
                                   "version1112": value1112},
                     "column112": {"version1121": value1121,
                                   "version1122": value1122,
                                   "version1123": value1123,
                                   "version1124": value1124},
                     "column113": {"version1131": value1131}
                    },
           "cf12": {"column121": {"version1211": value1211},
                     "column122": {"version1221": value1221},
                                   "version1222": value1222}
                    }
          },
"rowkey2": {"cf11": {"column111": {"version2111": value2111,
                                   "version2112": value2112},
                     "column112": {"version2121": value2121,
                                   "version2122": value2122,
                                   "version2123": value 2123,
                                   "version2124": value2124}
                    },
            "cf12": {"column121": {"version2211": value2211},
                     "column122": {"version2221": value2221}
                    }
          }
```

i. HBase Tables are divided horizontally by row key range into “Regions.” A region contains all rows in the table between the region’s start key and end key. Regions are assigned to the nodes in the cluster, called “Region Servers,” and these serve data for reads and writes. A region server can serve about 1,000 regions.

j. 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.

k. HBase is built upon distributed filesystems with file storage distributed across commodity machines.



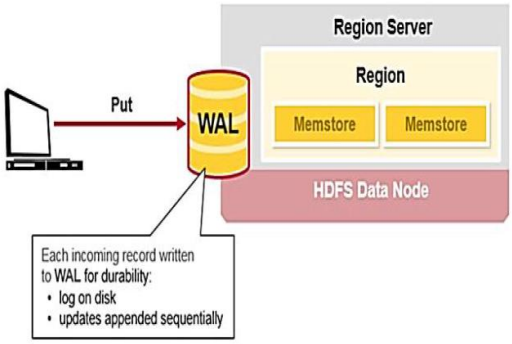
l. The data is stored in HFiles, which are ordered immutable key/value maps. Internally, the HFiles are sequences of blocks with a block index stored at the end. The block index is loaded when the HFile is opened and kept in memory. The default block size is 64 KB but it can be changed since it is configurable. HBase API can be used to access specific values and also scan ranges of values given a start and end key.

m. Since every HFile has a block index, lookups can be performed with a single disk seek. First, HBase does a binary search in the in-memory block index to find a block containing the given key and then the block is read from disk.

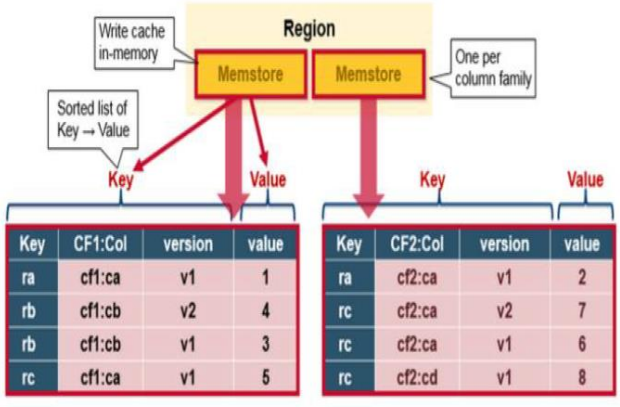
n. When data is updated it is first written to a commit log, called a write-ahead log (WAL) and then it is stored in the in-memory memstore.
o. When the data in memory exceeds a given maximum value, it is flushed as an HFile to disk and after that the commit logs are discarded up to the last unflushed modification. The system can continue to serve readers and writers without blocking them while it is flushing the memstore to disk. This is done by rolling the memstore in memory where the new empty one is taking the updates and the old full one is transferred into an HFile. At the same time, no sorting or other special processing has to be performed since the data in the memstores is already sorted by keys matching what HFiles represent on disk.
p. The write-ahead log (WAL) is used for recovery purposes only. Since flushing memstores to disk causes creation of HFiles, HBase has a housekeeping job that merges the HFiles into larger ones using compaction. Various compaction algorithms are supported.

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

a. 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.



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

c. When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS.

d. HBase uses multiple HFiles per column family, which contains the actual cells, or KeyValue instances. These files that are created over time as KeyValue edits, sorted in the MemStores are flushed as files to disk. This is one reason why there is a limit to the number of column families in HBase. There is one MemStore per CF; when one is full, they all flush.