**HBase : Assignment 9.1**

**Q. What is No SQL data base?**

**Ans.** A NoSQL database environment is, simply put, a non-relational and largely distributed database system that enables rapid, ad-hoc organization and analysis of extremely high-volume, disparate data types. NoSQL databases are sometimes referred to as cloud databases, non-relational databases, Big Data databases and a myriad of other terms and were developed in response to the sheer volume of data being generated, stored and analyzed by modern users (user-generated data) and their applications (machine-generated data).

In general, NoSQL databases have become the first alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. They go well beyond the more widely understood legacy, relational databases (such as Oracle, SQL Server and DB2 databases) in satisfying the needs of today’s modern business applications. A very flexible and [schema-less data model](http://planetcassandra.org/blog/post/schema-vs-schema-less), horizontal scalability, distributed architectures, and the use of languages and interfaces that are “not only” SQL typically characterize this technology.

From a business standpoint, considering a NoSQL or ‘Big Data’ environment has been shown to provide a clear competitive advantage in numerous industries. In the ‘age of data’, this is compelling information as a great saying about the importance of data is summed up with the following “if your data isn’t growing then neither is your business”.

**Q. How does data get stored in No SQL data base?**

**Ans.** Data will store in following way in No SQL database :-

**In the in-memory databases** like Redis / CouchBase / Tarantool / Aerospike everything is stored in RAM in balanced trees like RB-Tree or in hash tables. All the writes are applied on both RAM and disk, but on disk it goes in an append-only way. A file append can be done as fast as 100Mbytes per second on a normal magnetic disk. If a record size is, say, 1K, then the data will be written at 100krps.  
  
**In the on-disk NoSQL databases and db-engines**like Cassandra / HBase / RocksDB / LevelDB / Sophia the main idea is that you have a snapshot file and a write ahead log (WAL) file. Snapshot contains already prepared data in a form of B-Tree with upper levels of that tree being permanently in RAM, that can be accesses for reading by doing only one disk seek. A WAL contains all the new changes on top of a current snapshot. A snapshot file is being totally rebuilt on a regular basis using current snapshot and a WAL. All the writes are done nearly as fast as with in-memory databases. "Nearly" because disk is partially busy by doing regular snapshot converting that was described earlier. Reads are significantly slower than that are in in-memory databases, because they take at least one disk seek, but good news is that they can be cached in optimized in-memory structures like RB-Trees/hash tables.

**Q. What is a column family in Hbase?**

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

**Q. How many maximum number of column can be added in Hbase table?**

**Ans.** HBase currently does not do well with anything above two or three column families so keep the number of column families in your schema low. 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 though the amount of data they carry is small. When many column families the flushing and compaction interaction can make for a bunch of needless i/o loading (To be addressed by changing flushing and compaction to work on a per column family basis).

**Q. Why column are not defined at the time of table creation in Hbase?**

**Ans.** 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. The colon character (:) delimits the column family from the. 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 an running.

**Q. How does data get managed in Hbase?**

**Ans.** NoSQL databases are designed for scalability where unstructured data is spread across multiple nodes. When data volumes increase you just need to add another node to accommodate the growth. The lack of structure in NoSQL databases relaxes stringent requirements of consistency enforced in relational databases to improve speed and agility. Hbase, MongoDB and Cassandra are the three major options that provide NoSQL capabilities. The options differ in the features they provide, so the decision on which to use is informed by the workload that will be handled. The main difference between Hbase and Cassandra databases is the consistency model they implement. Cassandra implements eventual consistency which guarantees writes are available. This provides excellent write scaling but suffers a penalty when reading because for consistency in reads you have to read from many copies of data. On the other hand HBase provides a strong consistency model that excels at scaling reads but does not scale on writes as well as Cassandra does.

Hbase is natively supported on Hadoop and it is the subject of this tutorial. The main characteristics that make Hbase an excellent data management platform are fault tolerance, speed and usability. Fault tolerance is provided by automatic fail-over, automatically sharded and load balanced tables, strong consistency in row level operations and replication. Speed is provided by almost real time lookups, in memory caching and server side processing. Usability is provided by a flexible data model that allows many uses, a simple Java API and ability to export metrics.

Hbase can run standalone on the local file system but this set up does not guarantee durability. Edits will be lost when daemons are not cleanly started and stopped. Such a set up is not suitable in a production environment but it provides a way of exploring how the database functions. Alternatively Hbase can be installed on a single or multi node cluster and use HDFS.

**Q. What happens internally when new data gets inserted into Hbase table?**

**Ans.** When we insert new 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, you would issue a Put with a different value for the data itself, but the same timestamp.

HBase's behavior regarding versions is highly configurable. The maximum number of versions defaults to 1 in CDH 5, and 3 in previous versions. You can change the default value for HBase by configuring hbase.column.max.version in hbase-site.xml, either using an advanced configuration snippet if you use Cloudera Manager, or by editing the file directly otherwise.

You can also configure the maximum and minimum number of versions to keep for a given column, or specify a default time-to-live (TTL), which is the number of seconds before a version is deleted. The following examples all use alter statements in HBase Shell to create new column families with the given characteristics, but you can use the same syntax when creating a new table or to alter an existing column family. This is only a fraction of the options you can specify for a given column family.

hbase> alter ‘t1′, NAME => ‘f1′, VERSIONS => 5

hbase> alter ‘t1′, NAME => ‘f1′, MIN\_VERSIONS => 2

hbase> alter ‘t1′, NAME => ‘f1′, TTL => 15

HBase sorts the versions of a cell from newest to oldest, by sorting the timestamps lexicographically. When a version needs to be deleted because a threshold has been reached, HBase always chooses the "oldest" version, even if it is in fact the most recent version to be inserted. Keep this in mind when designing your timestamps. Consider using the default generated timestamps and storing other version-specific data elsewhere in the row, such as in the row key.