1: When RegionServer (RS) receives write request, it directs the request to specific Region. Each Region stores set of rows. Rows data can be separated in multiple column families (CFs). Data of particular CF is stored in HStore which consists of Memstore and a set of HFiles. Memstore is kept in RS main memory, while HFiles are written to HDFS. When write request is processed, data is first written into the Memstore. Then, when certain thresholds are met (obviously, main memory is well-limited) Memstore data gets flushed into HFile.

The main reason for using Memstore is the need to store data on HDFS ordered by row key.

Explanation: After the request is received by the right region server, the change cannot be written to a HFile immediately because the data in a HFile must be sorted by the row key. This allows searching for random rows efficiently when reading the data. Data cannot be randomly inserted into the HFile in HDFS. Instead, the change must be written to a new file. If each update were written to a file, many small files would be created. Such a solution would not be scalable nor efficient to merge or read at a later time. Therefore, changes are not immediately written to a new HFile. Instead, each change is stored in a place in memory called the memstore, which cheaply and efficiently supports random writes. Data in the memstore is sorted in the same manner as data in a HFile. When the memstore accumulates enough data, the entire sorted set is written to a new HFile in HDFS.

2: Every time Memstore flush happens one HFile created for each Column Family. Frequent flushes from memstore may create tons of HFiles. Since during reading HBase will have to look at many HFiles, the read speed can suffer.

To prevent opening too many HFiles and avoid read performance deterioration there’s HFiles compaction process. HBase will periodically (when certain configurable thresholds are met) compact multiple smaller HFiles into a big one. Obviously, the more files created by Memstore flushes, the more work (extra load) for the system.

When something is written to HBase, it is first written to an in-memory store (memstore), once this memstore reaches a certain size, it is flushed to disk into a store file(HFile). The store files created on disk are immutable. Sometimes the store files are merged together, this is done by a process called compaction.

There are two kinds of compactions:

a: the minor compactions: these are triggered each time a memstore is flushed, and will merge some of the store files, determined by an algorithm described below.

b: the major compactions: these run about every 24 hours (after the currently oldest store file was written), and merge together all store files into one. The 24 hours is adjusted with a random margin of up to 20% to avoid many major compactions happening at the same time. Major compactions can also be triggered manually, via the API or the shell.

3: Logical entities of HBase:

a. Table: HBase organizes data into tables. Table names are Strings and composed of characters that are safe for use in a file system path.

b. Row: Within a table, data is stored according to its row. Rows are identified uniquely by their row key. Row keys do not have a data type and are always treated as a byte[ ] (byte array).

c. Column Family: Data within a row is grouped by column family. Column families also impact the physical arrangement of data stored in HBase. For this reason, they must be defined up front and are not easily modified. Every row in a table has the same column families, although a row need not store data in all its families. Column families are Strings and composed of characters that are safe for use in a file system path.

d. Column Qualifier: Data within a column family is addressed via its column qualifier, or simply, column. Column qualifiers need not be specified in advance.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[ ].

e. Cell: A combination of row key, column family, and column qualifier uniquely identifies a cell. The data stored in a cell is referred to as that cell’s value. Values also do not have a data type and are always treated as a byte[ ].

f. Timestamp: Values within a cell are versioned. Versions are identified by their version number, which by default is the timestamp of when the cell was written.If a timestamp is not specified during a write, the current timestamp is used. If the timestamp is not specified for a read, the latest one is returned. The number of cell value versions retained by HBase is configured for each column family. The default number of cell versions is three.

5: When reading data from HBase using Get or Scan operations, you can use custom filters to return a subset of results to the client. While this does not reduce server-side IO, it does reduce network bandwidth and reduces the amount of data the client needs to process. Filters are generally used using the Java API, but can be used from HBase Shell for testing and debugging purposes.

6: Mainly four data model operations are performed in HBase. They are get, put, scan and delete.

7: Suppose HBase as a data source and sink for Mapreduce jobs, then these steps would be performed for the execution of job.

a. HBase provides a TableInputFormat, to which a table scan is provided, that splits the rows resulting from the table scan into the regions in which those rows reside.

b. The map process is passed an ImmutableBytesWritable that contains the row key for a row and a Result that contains the columns for that row.

c. The map process outputs its key/value pair based on its business logic in whatever form makes sense to application.

d. The reduce process builds its results but emits the row key as an ImmutableBytesWritable and a Put command to store the results back to HBase.

e. Finally, the results are stored in HBase by the HBase MapReduce infrastructure. (You do not need to execute the Put commands.

8. 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.

The region servers have regions that -

a. Communicate with the client and handle data-related operations.

b. Handle read and write requests for all the regions under it.

c. Decide the size of the region by following the region size thresholds.

When we take a deeper look into the region server, it contain regions and stores. The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.