

**Q.1. Differences between HBASE and HDFS.**

**HDFS is a file system and HBASE is a datastore.**

HDFS is meant for storing massive amounts of data across a distributed system and has the following properties:

1. It is optimized for streaming access of large files. You would typically store files that are in the 100s of MB upwards on HDFS and access them through MapReduce to process them in batch mode.

2. HDFS files are write once files. HDFS files are write-once and read-many files. There is no concept of random writes.

3. HDFS doesn't do random reads very well.

4.Data in HDFS is stored as flat files.  
  
**HBase** on the other hand is a database that stores it's data in a distributed filesystem. HBase is a non-relational database that can run on top of Hadoop and provides you random data access/querying capabilities. HDFS, by itself has no support for reads/writes at random location. HBase is an extension for the Hadoop environment that allows you to quickly read/write data.

HBase provides you with the following:

1. Low latency access to small amounts of data from within a large data set. You can access single rows quickly from a billion row table.

2. Flexible data model to work with and data is indexed by the row key.

3.HBase stores data as key/value pairs as in a column database (something similar to Cassandra DB).

4. Fast scans across tables.

5. Scale in terms of writes as well as total volume of data.

|  |  |
| --- | --- |
| **HBASE** | **HDFS** |
| HBase is a database built on top of the HDFS. | HDFS is a distributed file system suitable for storing large files. |
| HBase provides fast lookups for larger tables | HDFS does not support fast individual record lookups. |
| It provides low latency access to single rows from billions of records (Random access). | It provides high latency batch processing; no concept of batch processing. |
| HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups. | It provides only sequential access of data. |
| Hbase can be updated | HDFS files are write once files |

**Q.2. List and explain the main components of HBASE.**

**Components of HBase:**

The three major components of HBase, which takes part in an operation are as follows:

**1. Hmaster**

**2. Zookeeper**

**3. RegionServer**

These three components work together to make HBase a fully functional and efficient database.

**HMaster:**

The journey of an operation starts with the Client sending a request to the HBase. The following are the steps in the order of its execution.

1. Hmaster gives instruction to Hregion.

2. Hmaster takes information from Zookeeper if any services fail to respond.

3. Hmaster is the one, which responds and takes the request in.

4. Hmaster on startup coordinates &amp; monitors Region Server also assign Region

5. Hmaster creates Region and let know the Hregion Server to store data to the following region.

6. Hmaster is responsible for creating a table.

7. Hmaster is responsible for load balancing.

8. HMaster monitors nodes to discover all available region servers, and also monitors these nodes for server failures.

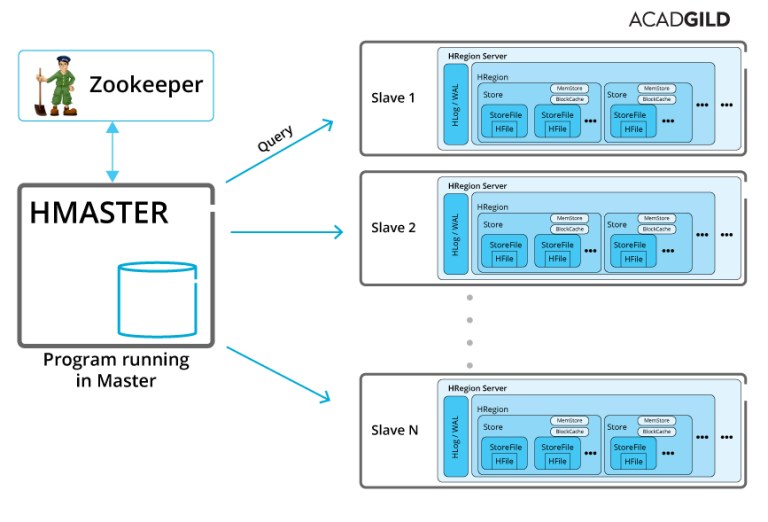
9. HMaster reassigns the regions from the crashed server to active Region servers.

10. There can be multiple Hmaster, just in case of backup as an inactive state.

11. The active HMaster sends heartbeats to Zookeeper, and the inactive HMaster listens for notifications of the active HMaster failure.

12. HMaster splits the WAL belonging to the crashed region server into separate files and stores these file in the new region servers’ data nodes.

Similar to this, several requests might be coming to Hmaster, making it too busy to perform all these work by itself.



**Region Server :**

● Region Server actually stores data.

● Region server does both the work of reading and writing data into the table.

● Write Ahead Log (WAL) gives fault tolerant feature, which is also known as Hlog.

● A region of a table is served to the client by a Region Server.

● A region server can serve about 1,000 regions (which may belong to the same table or different tables).

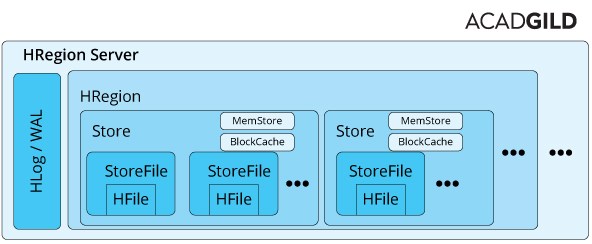
● When accessing data, the clients communicate with HBase Region Servers directly.

● Region Server has BlockCach, which is a read cache that frequently stores the read data in memory. Least Recently Used data is removed when the cache is full.

● Region Server have MemStore, which is the write cache. It keeps the new data which has not yet been written to disk. There is one MemStore available per column family per region.

● Hfiles store the rows as sorted KeyValues on disk.

● Multiple Hfile make 1 region.



The servers lay in different nodes in distributed system and are slave nodes. The master node is known as Hmaster.

**Zookeeper**

● **Manage configuration across nodes** – If you have dozens or hundreds of nodes, it becomes hard to keep configuration in sync across nodes and quickly make changes. ZooKeeper helps you to quickly push the configuration changes.

● **Implement reliable messaging** – With ZooKeeper, you can easily implement a producer/consumer queue that guarantees delivery, even if some consumers or even one of the ZooKeeper servers fails.

**● Implement redundant services** – With ZooKeeper, a group of identical nodes (e.g. database servers) can elect a leader/master and let ZooKeeper refer all clients to that master server. If the master fails, ZooKeeper will assign a new leader and notify all clients.

● **Synchronize process execution** – With ZooKeeper, multiple nodes can coordinate the start and end of a process or calculation. This ensures that anyfollow-up processing is done only after all nodes have finished their calculations.

**Q.3. Does Hbase support sql?**

HBase does not support a structured query language like SQL. Infact, HBase is not a relational data store at all. HBase applications are written in Java much like a typical [MapReduce](http://www.ibm.com/software/data/infosphere/hadoop/mapreduce/) application. HBase does support writing applications in Avro, REST, and Thrift.  It is well suited for sparse data sets, which are common in many big data use cases.