**Difference between hbase and hive**

Hadoop is, essentially, HDFS (Hadoop Distributed File System) and MapReduce. HDFS is meant for storing massive amounts of data across a distributed system. Technically speaking, your question should be on the difference between HBase and HDFS.

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.

Another primary difference would be the way data is stored in the two. HBase stores data as key/value pairs as in a column database (something similar to Cassandra DB) while data, in HDFS is stored as flat files.

To put it simply, HBase is an extension for the Hadoop environment that allows you to quickly read/write data.

**Components of HBASE.**

HBase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

1. **HMaster**

HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

* Manages and Monitors the Hadoop Cluster
* Performs Administration (Interface for creating, updating and deleting tables.)
* Controlling the failover
* DDL operations are handled by the HMaster
* Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

1. **Region Server**

These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

* Block Cache – This is the read cache. Most frequently read data is stored in the read cache and whenever the block cache is full, recently used data is evicted.
* MemStore- This is the write cache and stores new data that is not yet written to the disk. Every column family in a region has a MemStore.
* Write Ahead Log (WAL) is a file that stores new data that is not persisted to permanent storage.
* HFile is the actual storage file that stores the rows as sorted key values on a disk.

1. **Zookeeper**

HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning. ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization. Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.

ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

* Establishing client communication with region servers.
* Tracking server failure and network partitions.
* Maintain Configuration Information
* Provides ephemeral nodes, which represent different region servers.

**When to use hbase:**

* The most important consideration when looking at HBase is that, while it is a great solution to many problems, it is not a silver bullet. HBase is not optimized for classic transactional applications or even relational analytics. It is also not a complete substitute for HDFS when doing large batch MapReduce. Take a look at some of the [use cases](http://blog.cloudera.com/blog/2011/04/hbase-dos-and-donts/#usecases) in this post to get a sense of which applications are a good fit for HBase and if you have questions, go ahead and post on the [lists](http://blog.cloudera.com/community/). Have I mentioned that the [community](http://blog.cloudera.com/community/) is fantastic?
* With that caveat out the way – why should you use HBase? If your application has a variable schema where each row is slightly different, then you should look at HBase. As an example, doing a modeling exercise using a standard relational schema; When you can’t add columns fast enough and most of them are NULL in each row, you should consider HBase. If you find that your data is stored in collections, for example some meta data, message data or binary data that is all keyed on the same value, then you should consider HBase. If you need key based access to data when storing or retrieving, then you should consider HBase.

**Different modes of hbase:**

### **Standalone HBase**

This is the default mode. In standalone mode, HBase does not use HDFS -- it uses the local filesystem instead -- and it runs all HBase daemons and a local ZooKeeper all up in the same JVM. Zookeeper binds to a well known port so clients may talk to HBase

**Distributed**

Distributed mode can be subdivided into distributed but all daemons run on a single node -- a.k.a pseudo-distributed-- and fully-distributed where the daemons are spread across all nodes in the cluster [[9](http://hbase.apache.org/0.94/book/standalone_dist.html" \l "ftn.d1984e717)].

Distributed modes require an instance of the Hadoop Distributed File System (HDFS). Before proceeding, ensure you have an appropriate, working HDFS.

**Needs of zookeeper in hbase:**

A distributed Apache HBase (TM) installation depends on a running ZooKeeper cluster. All participating nodes and clients need to be able to access the running ZooKeeper ensemble. Apache HBase by default manages a ZooKeeper "cluster" for you. It will start and stop the ZooKeeper ensemble as part of the HBase start/stop process. You can also manage the ZooKeeper ensemble independent of HBase and just point HBase at the cluster it should use. To toggle HBase management of ZooKeeper, use the HBASE\_MANAGES\_ZK variable in conf/hbase-env.sh. This variable, which defaults to true, tells HBase whether to start/stop the ZooKeeper ensemble servers as part of HBase start/stop.

When HBase manages the ZooKeeper ensemble, you can specify ZooKeeper configuration using its native zoo.cfg file, or, the easier option is to just specify ZooKeeper options directly in conf/hbase-site.xml. A ZooKeeper configuration option can be set as a property in the HBase hbase-site.xml XML configuration file by prefacing the ZooKeeper option name with hbase.zookeeper.property. For example, the clientPort setting in ZooKeeper can be changed by setting the hbase.zookeeper.property.clientPort property.

**Hbase is a schema less database, what does it mean?**

It really means is that the "schema" is stored with the record, not the table. In a RDBMS, the schema is defined and that table has the schema. In HBase (and other BigTable implementations) data is labeled with its types.

An analogy is CSV : RDBMS is to XML : HBase. In CSV, you have an assumption on what a column is. In XML, you specify exactly what it is.

benefit of using connection pool in Hbase?

For applications which require high-end multithreaded access (e.g., web-servers or application servers that may serve many application threads in a single JVM), one solution is [HTablePool](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTablePool.html). But as written currently, it is difficult to control client resource consumption when using HTablePool.

Another solution is to precreate an HConnection

**Difference between memstore and hfile in hbase**

Memstore: Memstore is an in-memory storage, hence the Memstore utilizes the in-memory storage of each data node to store the logs. Rows are written to theMemStore. The data in the MemStore is ordered.When certain thresholds are met, Memstore data gets flushed into HFile. Every time Memstore flush happens one HFile created for each ColumnFamily

HFile: HFiles are the actual storage files i.e. physical representation of data in HFile, specifically created to serve one purpose: store HBase’s data fast and efficiently. Clients do not read HFiles directly but go through region servers to get to the data.

**Compactions in hbase:**

A[pache HBase](http://hbase.apache.org) is a distributed data store based upon a log-structured merge tree, so optimal read performance would come from having only one file per store (Column Family). However, that ideal isn’t possible during periods of heavy incoming writes. Instead, HBase will try to combine HFiles to reduce the maximum number of disk seeks needed for a read. This process is called *compaction*.

Compactions choose some files from a single store in a region and combine them. This process involves reading KeyValues in the input files and writing out any KeyValues that are not deleted, are inside of the time to live (TTL), and don’t violate the number of versions. The newly created combined file then replaces the input files in the region.

Now, whenever a client asks for data, HBase knows the data from the input files are held in one contiguous file on disk — hence only one seek is needed, whereas previously one for each file could be required. But disk IO isn’t free, and without careful attention, rewriting data over and over can lead to some serious network and disk over-subscription. In other words, compaction is about trading some disk IO now for fewer seeks later.

**How can filters be applied in HBase and what are the benefits?**

It allows you to perform server-side filtering when accessing [**HBase**](http://www.hadooptpoint.com/category/nosql-databases/hbase/)over [**Thrift**](http://thrift-tutorial.readthedocs.org/en/latest/usage-example.html) or in the HBase shell. You can find out more about shell integration by using the scan help command in the shell.

**data model operations in hBase**

The four primary data model operations are Get, Put, Scan, and Delete. Operations are applied via [HTable](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html) instances.

### **Get**

[Get](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Get.html) returns attributes for a specified row. Gets are executed via [HTable.get](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html#get%28org.apache.hadoop.hbase.client.Get%29).

**Put**

[Put](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Put.html) either adds new rows to a table (if the key is new) or can update existing rows (if the key already exists). Puts are executed via [HTable.put](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html#put%28org.apache.hadoop.hbase.client.Put%29) (writeBuffer) or [HTable.batch](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html#batch%28java.util.List%29) (non-writeBuffer).

**Scans**

[Scan](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Scan.html) allow iteration over multiple rows for specified attributes.

### **Delete**

[Delete](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Delete.html) removes a row from a table. Deletes are executed via [HTable.delete](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html#delete%28org.apache.hadoop.hbase.client.Delete%29).

HBase does not modify data in place, and so deletes are handled by creating new markers called tombstones. These tombstones, along with the dead values, are cleaned up on major compactions.

**How MapReduce can be used with HBase?**

To run MapReduce jobs that use HBase, you need to add the HBase and Zookeeper JAR files to the Hadoop Java classpath. You can do this by adding the following statement to each job:

TableMapReduceUtil.addDependencyJars(job);

This distributes the JAR files to the cluster along with your job and adds them to the job's classpath, so that you do not need to edit the MapReduce configuration.

When getting an Configuration object for a HBase MapReduce job, instantiate it using the HBaseConfiguration.create() method.

**Regionserver:**

RegionServers are the software processes (often called daemons) you activate to store and retrieve data in HBase (Hadoop Database). In production environments, each RegionServer is deployed on its own dedicated compute node. When you start using HBase, you create a table and then begin storing and retrieving your data.

However, at some point — and perhaps quite quickly in big data use cases — the table grows beyond a configurable limit. At this point, the HBase system automatically splits the table and distributes the load to another RegionServer.

In this process, often referred to as *auto-sharding*, HBase automatically scales as you add data to the system — a huge benefit compared to most database management systems, which require manual intervention to scale the overall system beyond a single server. With HBase, as long as you have in the rack another spare server that’s configured, scaling is automatic!