HBase Architecture

In this video, we shall talk about the architecture of HBase. In HBase, the table data is stored in Regions and when the table becomes too big, the table is partitioned into multiple Regions with each Regions storing a range of the tables rows. The data is partitioned based on the RowKeys into different Regions. Here, the basic unit of horizontal scalability is the Region. So essentially Regions are subset of table's data and are nothing but contiguous sorted range of rows that are stored together.

So you start with one Region and when the table grows, it gets split into multiple Regions and these Regions are represented in HBase as HRegionServers. Now, there should be someone to manage these Regions and that is handled in HBase by the RegionServer which is represented as a service called HRegionServer.

Now, HRegionServer performs the following work. It is responsible for hosting and managing Regions. It is also responsible for splitting the Regions automatically when the tables grow. It's also responsible for handling the read/write request and also to communicate with the clients directly. So within the Hadoop cluster, typically, the RegionServer runs on nodes on which HDFS

data node process is running. That can and will be multiple RegionServers running in the cluster, each with multiple Regions or it is not necessary that there are as many HBase RegionServers as there are HDFS nodes.

In order to monitor all the RegionServers, we have a master server. The master server is the interface for all META data changes. It is also responsible for performing administration, managing and monitoring the cluster, also responsible for assigning Regions to different RegionServers and it also controls the load balancing and failover activities. The master typically runs on the same machine on which the HDFS name node services are running.

To quickly summarise, we have a sign HBase master that monitors the slave RegionServers which are in turn responsible for serving and managing table Regions and Regions are nothing but HBase table data which is partitioned based on the RowKeys.

HBase uses log structured merge tree or the LSM data structure to process the data writing operation.

The LSM tree works in the following way. All puts and inserts are first appended to a Write Ahead Log. The Write Ahead Log file is common to the all the Regions in a

RegionServer and it stores a data in HDFS and it can be used to restore the database in case if anything goes wrong. So we will get the feel of why Write Ahead Logs are used in a couple of minutes.

The inserts are then written into a in- memory structure called as the memory store, which basically is used to store the most recent inserts. It stores in fast and an ordered way and when the Memstore accumulates enough data, the entire sorted data is then written on to a new HFile. These HFiles are in HBase specific file format and they constitute the actual data of HBase table and that is partitioned in that Region. There can be multiple HFiles for a table in a single Region.

Note that HFile is used for HBase as a format to store tables in HDFS. HFiles stores the keys in lexicographic order using RowKeys. In case if they are going to be numeric RowKeys, they are going to be ordered in numeric order. It is basically a block indexed file format for storing key value pairs and block indexed here basically means that data is stored in sequence of blocks and a separate index is maintained at the end of the file to locate these blocks. So you have a sequence of blocks that represent the data and then the separate index at the end of the file which hold the Metadata for these blocks.

So when the read request comes in, the index is what is searched for first and then it goes to the block location and then the data is read from that particular block.

We know that the RegionServers contribute to heading the HBase tables and each RegionServer handles one or more Regions or partitions of the table data. So there needs to be a placeholder to capture which Region is being served by which RegionServer and etc.

So, for that purpose, HBase has two special catalogue tables. First we have the META Table which basically holds the location of the Regions of all the tables.

Remember table data is basically partition into multiple Regions. So the META holds the information as to which Region of a particular table is stored in which RegionServer. So it basically keeps track of the current location for each Region. For each table along with some information about the Region like its name, Region info, etc.

A row in the META Table corresponds to a single Region. So one row represents a reason, its corresponding RegionServer and all the necessary information.

Then we have the ROOT Table which basically holds the location of the META Table and because the RegionServer holding the ROOT can crash, we want to always know where the ROOT lives.

So we store the location of the ROOT in a ZooKeeper node. ZooKeeper is basically a distributed coordination service for distributed applications and HBase uses it for storing the META data related to the ROOT file. And HBase client updates a table by invoking, Put or Delete commands.

When a Put or a Delete command is invoked by a client, HBase has determine which Region it is supposed to write and which RegionServer manages that Region. Since the row key is sorted, it is easy to determine which RegionServer managers which key. So based on the commands, RowKey and HBase client, they can altogether locate a proper RegionServer.

So, the client first asks the ZooKeeper where to find the ROOT Table and then the client asks the ROOT Table for the correct META Table. Finally, it asks the META Table for the correct RegionServer. The clients store the information about ROOT and META Tables to speed up future lookups. So after the request is received by the write RegionServer, the

data cannot to written into a HFile because the data in HFile must be sorted by RowKey.

Remember, HFile is a lowest level of granularity of the data that is going to be stored in HBase. This file will have the partition data that is stored in Regions and each Region can have multiple HFiles and these HFiles must be sorted by RowKey that is where HBases performance will come into picture because this allows searching for random rows efficiently when reading the data. Data cannot be inserted randomly into a HFile because HDFS does not allow you to do that. HFile gets stored in HDFS. So I cannot directly or randomly insert record in that HFile.

Instead of writing into existing HFile directly, the change which is either an Insert or Delete will return to a new file but if each update, Insert or Delete where to be written to a new file, there would be many small files that could be created. Such a solution will not be scalable or efficient to merge or read at a later time.

Therefore, the changes are not immediately written to a new file. Instead each change is store in an in-memory store called as Memstore. Remember, in the architecture diagram, we showed you that there is a

Memstore for each Region and the Memstore, we can write data randomly and also read the data in the Memstore. And this is going to be sorted in the same manner as that is going to be in the case of HFile. So basically it's this in-memory Memstore, which is going to be used for sorting the data based on RowKeys. The sorting of data in a HFile is similar to the sorting of data in Memstore and in the Memstore accumulates enough data, the entire sorted data is written to a new file. As you can see here, the Memstore writes the data into the HFile and the Right Ahead Log that you can see in the architecture here is basically because the Memstore data is stored in memory.

What happens if the server crashes? The insert or update will be lost. So instead of that the data is first written into a Write Ahead Log file which is stored on HDFS. In case of scenarios where the servers goes down and the data in the memory is lost, I can always look after the Right Ahead Log and replay the transactions.

Now that the data is there in the Memstore and once it has sufficient amount of data, it will then be written into a new HFile. This is the architecture of HBase. It basically is a set of servers.

The Region servers which is managed by the master, which is the HMaster and within the Region servers, there will be one or more Regions for a particular table within the Region. There will be one or more HFiles which present the data. In order to enter data into HBase table, these are available options. I can use Apache Flume and gather streaming data and write directly into a HBase sink. I can use Apache PIG to load certain data into HBase.

Now imagine I have a PIG script. I have performed some transformation. That transformed data, I can then directly write into a HBase table. I can also do the same thing using Hive as well.

MapReduce

With MapReduce in HBase APIs, I can directly write or read data into a HBase table. For that matter, I can use Apache's Sqoop as well to write data directly into a HBase table. And in order to return retrieve data from HBase table, I can use PIG, I can use Hive and MapReduce. There are also HBase KPIs. I can also use Impala. Once data is stored in HBase, I can always extract them and still perform analysis on top of that. We can look at a couple of examples of loading bulk data into HBase in the next set of videos.