edureka.co

HBase Architecture | HBase Data Model | HBase Read/Write | Edureka

Shubham Sinha

18-23 minutes

HBase Architecture

In my previous blog on *HBase Tutorial*, I explained what is HBase and its features. I also mentioned Facebook messenger's case study to help you to connect better. Now further moving ahead in our *Hadoop Tutorial Series*, I will explain you the data model of HBase and HBase Architecture. Before you move on, you should also know that HBase is an important concept that makes up an integral portion of the *course curriculum* for Big Data Hadoop Certification.

The important topics that I will be taking you through in this HBase architecture blog are:

Let us first understand the data model of HBase. It helps HBase in faster read/write and searches.

HBase Architecture: HBase Data Model

As we know, HBase is a column-oriented NoSQL database. Although it looks similar to a relational database which contains rows and columns, but it is not a relational database. Relational databases are row oriented while HBase is column-oriented. So, let us first understand the difference between Column-oriented and Row-oriented databases:

Row-oriented vs column-oriented Databases:

- Row-oriented databases store table records in a sequence of rows. Whereas column-oriented databases store table records in a sequence of columns, i.e. the entries in a column are stored in contiguous locations on disks.

To better understand it, let us take an example and consider the table below.

Customer ID	Name	Address	Product ID	Product Name
1	Paul Walker	US	231	Gallardo
2	Vin Diesel	Brazil	520	Mustang

If this table is stored in a row-oriented database. It will store the records as shown below:

1, Paul Walker, US, 231, Gallardo,

2, Vin Diesel, Brazil, 520, Mustang

In row-oriented databases data is stored on the basis of rows or tuples as you can see above.

While the column-oriented databases store this data as:

1,2, Paul Walker, Vin Diesel, US, Brazil, 231, 520, Gallardo, Mustang

In a column-oriented databases, all the column values are stored together like first column values will be stored together, then the second column values will be stored together and data in other columns are stored in a similar manner.

- When the amount of data is very huge, like in terms of petabytes or exabytes, we use column-oriented approach, because the data of a single column is stored together and can be accessed faster.
- While row-oriented approach comparatively handles less number of rows and columns efficiently, as row-oriented database stores data is a structured format.
- When we need to process and analyze a large set of semi-structured or unstructured data, we use column oriented approach. Such as applications dealing with **Online Analytical Processing** like data mining, data warehousing, applications including analytics, etc.
- Whereas, **Online Transactional Processing** such as banking and finance domains which handle structured data and require transactional properties (ACID properties) use row-oriented approach.

HBase tables has following components, shown in the image below:

Figure: HBase Table

- **Tables**: Data is stored in a table format in HBase. But here tables are in column-oriented format.
- **Row Key**: Row keys are used to search records which make searches fast. You would be curious to know how? I will explain it in the architecture part moving ahead in this blog.
- **Column Families**: Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.
- **Column Qualifiers**: Each column's name is known as its column qualifier.
- **Cell**: Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
- **Timestamp**: Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

  In a more simple and understanding way, we can say HBase consists of:

- Set of tables
- Each table with column families and rows
- Row key acts as a Primary key in HBase.
- Any access to HBase tables uses this Primary Key
- Each column qualifier present in HBase denotes attribute corresponding to the object which resides in the cell.

Now that you know about HBase Data Model, let us see how this data model falls in line with HBase Architecture and makes it suitable for large storage and faster processing.

HBase Architecture: Components of HBase Architecture

HBase has three major components i.e., **HMaster Server**, **HBase Region Server, Regions** and **Zookeeper**.

The below figure explains the hierarchy of the HBase Architecture. We will talk about each one of them individually.

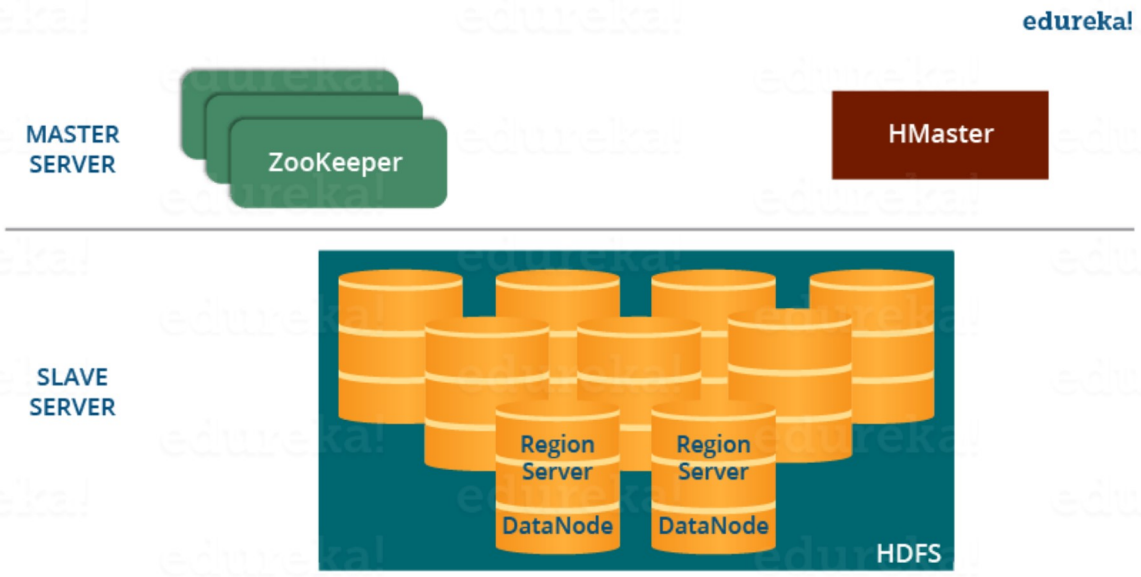

Figure: HMaster & Region Servers

Now before going to the HMaster, we will understand Regions as all these Servers (HMaster, Region Server, Zookeeper) are placed to coordinate and manage Regions and perform various operations inside the Regions. So you would be curious to know what are regions and why are they so important?

HBase Architecture: Region

A region contains all the rows between the start key and the end key assigned to that region. HBase tables can be divided into a number of regions in such a way that all the columns of a column family is stored in one region. Each region contains the rows in a sorted order.

Many regions are assigned to a **Region Server**, which is responsible for handling, managing, executing reads and writes operations on that set of regions.

So, concluding in a simpler way:

- A table can be divided into a number of regions. A Region is a sorted range of rows storing data between a start key and an end key.
- A Region has a default size of 256MB which can be configured according to the need.
- A Group of regions is served to the clients by a Region Server.
- A Region Server can serve approximately 1000 regions to the client.

Now starting from the top of the hierarchy, I would first like to explain you about HMaster Server which acts similarly as a

NameNode in *HDFS*. Then, moving down in the hierarchy, I will take you through ZooKeeper and Region Server.

HBase Architecture: HMaster

As in the below image, you can see the HMaster handles a collection of Region Server which resides on DataNode. Let us understand how HMaster does that.

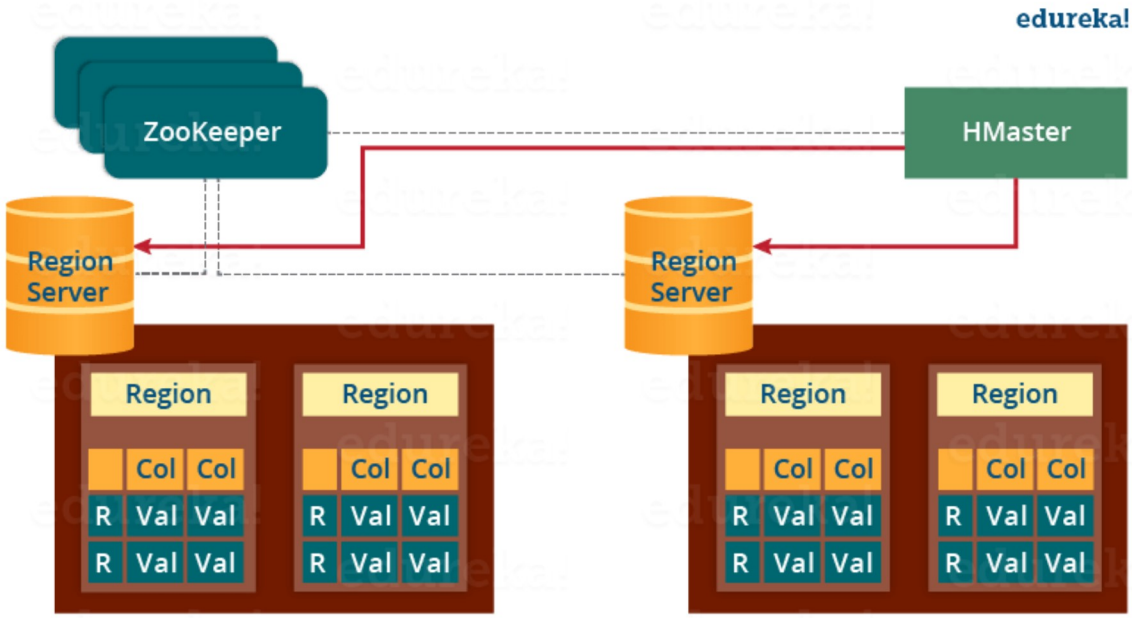

Figure: Components of HBase

- HBase HMaster performs DDL operations (create and delete tables) and assigns regions to the Region servers as you can see in the above image.
- It coordinates and manages the Region Server (similar as NameNode manages DataNode in HDFS).
- It assigns regions to the Region Servers on startup and re-assigns regions to Region Servers during recovery and load balancing.
- It monitors all the Region Server's instances in the cluster (with the help of Zookeeper) and performs recovery activities whenever any Region Server is down.

- It provides an interface for creating, deleting and updating tables.

  HBase has a distributed and huge environment where HMaster alone is not sufficient to manage everything. So, you would be wondering what helps HMaster to manage this huge environment? That's where ZooKeeper comes into the picture. After we understood how HMaster manages HBase environment, we will understand how Zookeeper helps HMaster in managing the environment.

HBase Architecture: ZooKeeper – The Coordinator

This below image explains the ZooKeeper's coordination mechanism.

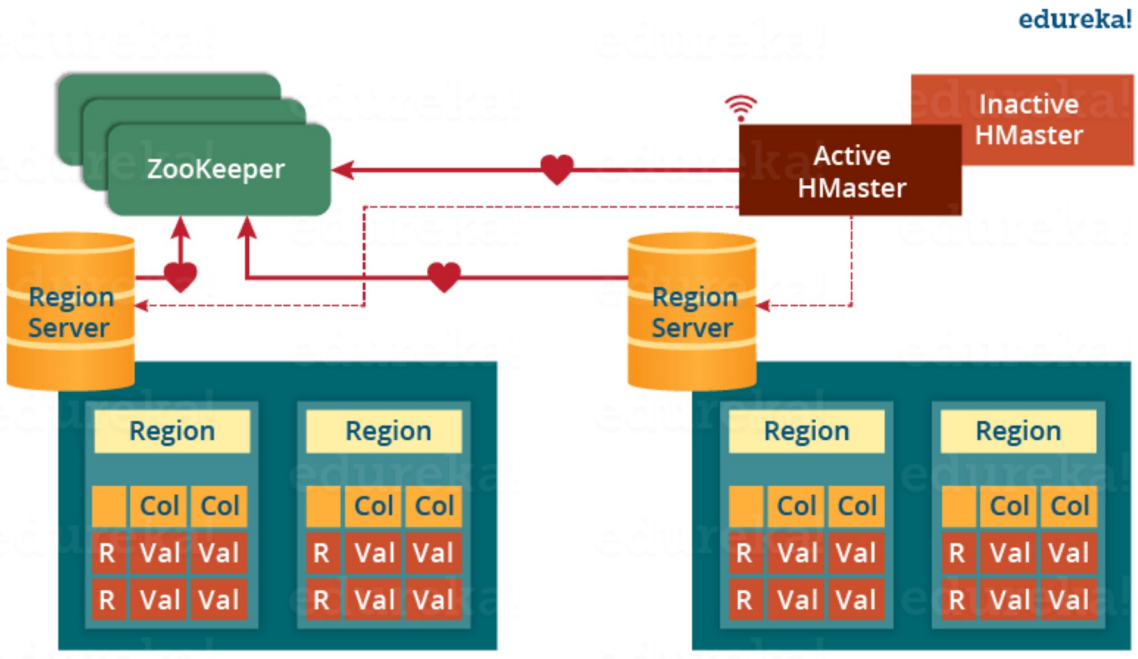

Figure: ZooKeeper as Coordination Service

- Zookeeper acts like a coordinator inside HBase distributed environment. It helps in maintaining server state inside the cluster by communicating through sessions.

- Every Region Server along with HMaster Server sends continuous heartbeat at regular interval to Zookeeper and it checks which server is alive and available as mentioned in above image. It also provides server failure notifications so that, recovery measures can be executed.
- Referring from the above image you can see, there is an inactive server, which acts as a backup for active server. If the active server fails, it comes for the rescue.
- The active HMaster sends heartbeats to the Zookeeper while the inactive HMaster listens for the notification send by active HMaster. If the active HMaster fails to send a heartbeat the session is deleted and the inactive HMaster becomes active.
- While if a Region Server fails to send a heartbeat, the session is expired and all listeners are notified about it. Then HMaster performs suitable recovery actions which we will discuss later in this blog.
- Zookeeper also maintains the .META Server's path, which helps any client in searching for any region. The Client first has to check with .META Server in which Region Server a region belongs, and it gets the path of that Region Server.

  As I talked about .META Server, let me first explain to you what is .META server? So, you can easily relate the work of ZooKeeper and .META Server together. Later, when I will explain you the HBase search mechanism in this blog, I will explain how these two work in collaboration.

HBase Architecture: Meta Table

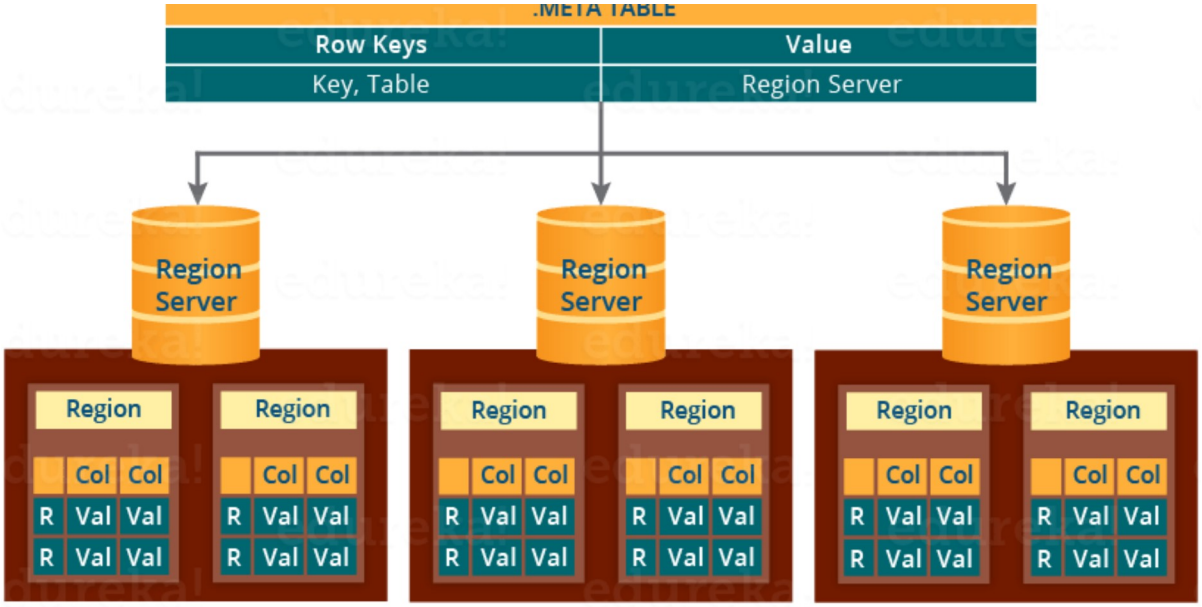

Figure: META Table

- The META table is a special HBase catalog table. It maintains a list of all the Regions Servers in the HBase storage system, as you can see in the above image.
- Looking at the figure you can see, **.META** file maintains the table in form of keys and values. Key represents the start key of the region and its id whereas the value contains the path of the Region Server.

  As I already discussed, Region Server and its functions while I was explaining you Regions hence, now we are moving down the hierarchy and I will focus on the Region Server's component and their functions. Later I will discuss the mechanism of searching, reading, writing and understand how all these components work together.

HBase Architecture: Components of Region Server

This below image shows the components of a Region Server. Now, I will discuss them separately.

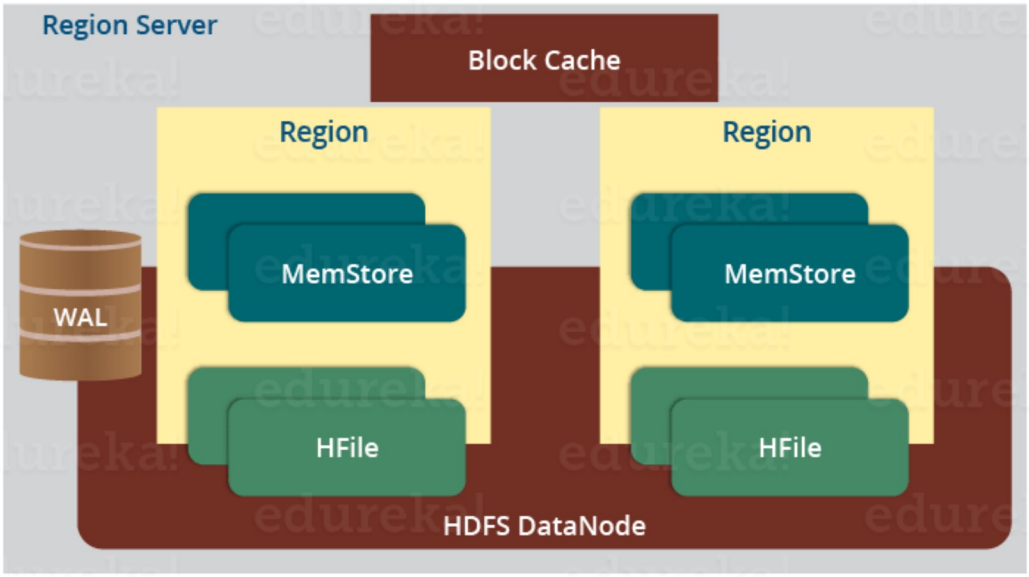

Figure: Region Server Components

A Region Server maintains various regions running on the top of ***HDFS***. Components of a Region Server are:

- **WAL:** As you can conclude from the above image, Write Ahead Log (WAL) is a file attached to every Region Server inside the distributed environment. The WAL stores the new data that hasn't been persisted or committed to the permanent storage. It is used in case of failure to recover the data sets.
- **Block Cache:** From the above image, it is clearly visible that Block Cache resides in the top of Region Server. It stores the frequently read data in the memory. If the data in BlockCache is least recently used, then that data is removed from BlockCache.
- **MemStore:** It is the write cache. It stores all the incoming data before committing it to the disk or permanent memory. There is one MemStore for each column family in a region. As you can see in the image, there are multiple MemStores for a region because each region contains multiple column families. The data is sorted in lexicographical order before committing it to the disk.
- **HFile:** From the above figure you can see HFile is stored on HDFS.

Thus it stores the actual cells on the disk. MemStore commits the data to HFile when the size of MemStore exceeds.

Now that we know major and minor components of HBase Architecture, I will explain the mechanism and their collaborative effort in this. Whether it's reading or writing, first we need to search from where to read or where to write a file. So, let's understand this search process, as this is one of the mechanisms which makes HBase very popular.

HBase Architecture: How Search Initializes in HBase?

As you know, Zookeeper stores the META table location. Whenever a client approaches with a read or writes requests to HBase following operation occurs:

1. The client retrieves the location of the META table from the ZooKeeper.
2. The client then requests for the location of the Region Server of corresponding row key from the META table to access it. The client caches this information with the location of the META Table.
3. Then it will get the row location by requesting from the corresponding Region Server.

For future references, the client uses its cache to retrieve the location of META table and previously read row key's Region Server. Then the client will not refer to the META table, until and unless there is a miss because the region is shifted or moved. Then it will again request to the META server and update the cache.

As every time, clients does not waste time in retrieving the location

of Region Server from META Server, thus, this saves time and makes the search process faster. Now, let me tell you how writing takes place in HBase. What are the components involved in it and how are they involved?

HBase Architecture: HBase Write Mechanism

This below image explains the write mechanism in HBase.

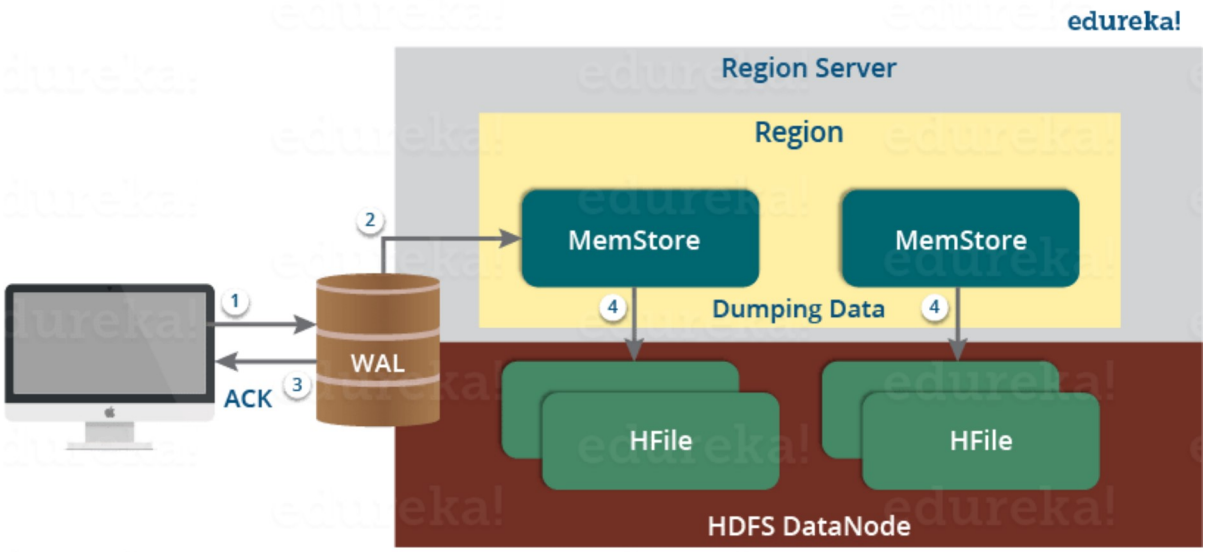

Figure: Write Mechanism in HBase

The write mechanism goes through the following process sequentially (refer to the above image):

Step 1: Whenever the client has a write request, the client writes the data to the WAL (Write Ahead Log).

- The edits are then appended at the end of the WAL file.
- This WAL file is maintained in every Region Server and Region Server uses it to recover data which is not committed to the disk.

Step 2: Once data is written to the WAL, then it is copied to the MemStore.

Step 3: Once the data is placed in MemStore, then the client

receives the acknowledgment.

Step 4: When the MemStore reaches the threshold, it dumps or commits the data into a HFile.

Now let us take a deep dive and understand how MemStore contributes in the writing process and what are its functions?

HBase Write Mechanism- MemStore

- The MemStore always updates the data stored in it, in a lexicographical order (sequentially in a dictionary manner) as sorted KeyValues. There is one MemStore for each column family, and thus the updates are stored in a sorted manner for each column family.
- When the MemStore reaches the threshold, it dumps all the data into a new HFile in a sorted manner. This HFile is stored in HDFS. HBase contains multiple HFiles for each Column Family.
- Over time, the number of HFile grows as MemStore dumps the data.
- MemStore also saves the last written sequence number, so Master Server and MemStore both knows, that what is committed so far and where to start from. When region starts up, the last sequence number is read, and from that number, new edits start.

  As I discussed several times, that HFile is the main persistent storage in an HBase architecture. At last, all the data is committed to HFile which is the permanent storage of HBase. Hence, let us look at the properties of HFile which makes it faster for search while reading and writing.

HBase Architecture: HBase Write Mechanism- HFile

- The writes are placed sequentially on the disk. Therefore, the movement of the disk's read-write head is very less. This makes write and search mechanism very fast.
- The HFile indexes are loaded in memory whenever an HFile is opened. This helps in finding a record in a single seek.
- The trailer is a pointer which points to the HFile's meta block . It is written at the end of the committed file. It contains information about timestamp and bloom filters.
- Bloom Filter helps in searching key value pairs, it skips the file which does not contain the required rowkey. Timestamp also helps in searching a version of the file, it helps in skipping the data.

  After knowing the write mechanism and the role of various components in making write and search faster. I will be explaining to you how the reading mechanism works inside an HBase architecture? Then we will move to the mechanisms which increases HBase performance like compaction, region split and recovery.

HBase Architecture: Read Mechanism

As discussed in our search mechanism, first the client retrieves the location of the Region Server from .META Server if the client does not have it in its cache memory. Then it goes through the sequential steps as follows:

- For reading the data, the scanner first looks for the Row cell in Block cache. Here all the recently read key value pairs are stored.

- If Scanner fails to find the required result, it moves to the MemStore, as we know this is the write cache memory. There, it searches for the most recently written files, which has not been dumped yet in HFile.
- At last, it will use bloom filters and block cache to load the data from HFile.

  So far, I have discussed search, read and write mechanism of HBase. Now we will look at the HBase mechanism which makes search, read and write quick in HBase. First, we will understand *Compaction*, which is one of those mechanisms.

HBase Architecture: Compaction

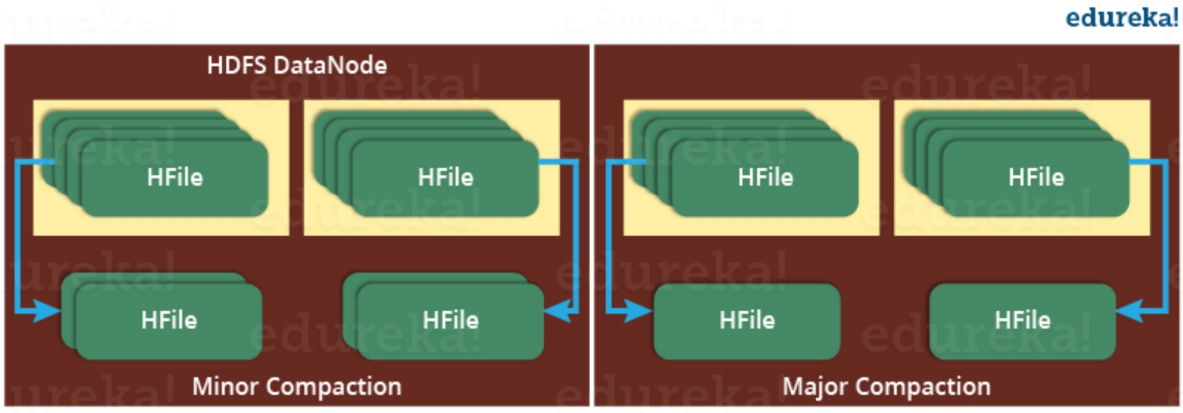

Figure: Compaction in HBase

HBase combines HFiles to reduce the storage and reduce the number of disk seeks needed for a read. This process is called **compaction**. Compaction chooses some HFiles from a region and combines them. There are two types of compaction as you can see in the above image.

1. **Minor Compaction**: HBase automatically picks smaller HFiles and recommits them to bigger HFiles as shown in the above image. This is called Minor Compaction. It performs merge sort for

committing smaller HFiles to bigger HFiles. This helps in storage space optimization.

2. **Major Compaction:** As illustrated in the above image, in Major compaction, HBase merges and recommits the smaller HFiles of a region to a new HFile. In this process, the same column families are placed together in the new HFile. It drops deleted and expired cell in this process. It increases read performance.

   But during this process, input-output disks and network traffic might get congested. This is known as **write amplification**. So, it is generally scheduled during low peak load timings.

   Now another performance optimization process which I will discuss is *Region Split*. This is very important for load balancing.

HBase Architecture: Region Split

The below figure illustrates the Region Split mechanism.

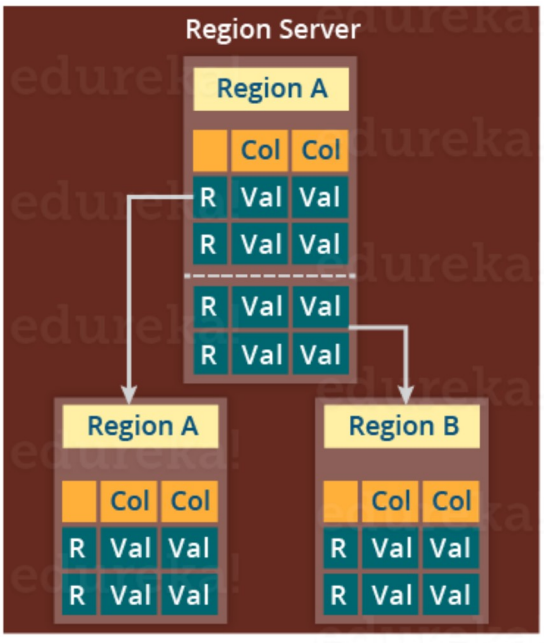

Figure: Region Split in HBase

Whenever a region becomes large, it is divided into two child regions, as shown in the above figure. Each region represents

exactly a half of the parent region. Then this split is reported to the HMaster. This is handled by the same Region Server until the HMaster allocates them to a new Region Server for load balancing.

Moving down the line, last but the not least, I will explain you how does HBase recover data after a failure. As we know that **Failure Recovery** is a very important feature of HBase, thus let us know how HBase recovers data after a failure.

HBase Architecture: HBase Crash and Data Recovery

- Whenever a Region Server fails, ZooKeeper notifies to the HMaster about the failure.
- Then HMaster distributes and allocates the regions of crashed Region Server to many active Region Servers. To recover the data of the MemStore of the failed Region Server, the HMaster distributes the WAL to all the Region Servers.
- Each Region Server re-executes the WAL to build the MemStore for that failed region's column family.
- The data is written in chronological order (in a timely order) in WAL. Therefore, Re-executing that WAL means making all the change that were made and stored in the MemStore file.
- So, after all the Region Servers executes the WAL, the MemStore data for all column family is recovered.

I hope this blog would have helped you in understating the HBase Data Model & HBase Architecture. Hope you enjoyed it. Now you can relate to the features of HBase (which I explained in my previous ***HBase Tutorial*** blog) with HBase Architecture and

understand how it works internally. Now that you know the theoretical part of HBase, you should move to the practical part. Keeping this in mind, our next blog of *Hadoop Tutorial Series* will be explaining a sample *HBase POC*.

Now that you have understood the HBase Architecture, check out the Hadoop training by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. The Edureka Big Data Hadoop Certification Training course helps learners become expert in HDFS, Yarn, MapReduce, Pig, Hive, HBase, Oozie, Flume and Sqoop using real-time use cases on Retail, Social Media, Aviation, Tourism, Finance domain.

Got a question for us? Please mention it in the comments section and we will get back to you.