HBase Conceptual Architecture

EECS 4314

Team Members:
Daniel McVicar - 213027479
Rafay Sheikh - 213033451
Daniel Fortunato - 216796443
Adham El Shafie - 212951018
Yahya Ismail - 213235403

Introduction

HBase is a non-relational, distributed database. As an open source project it also makes an ideal subject when studying the architecture of large scale software engineering programs. This report will investigate the general architecture style of Apache HBase, detail the subsystems used to realize that architectural design, and follow several use cases to examine how the system works in practice.

Background Information

HBase was modeled after a 2006 paper, "*Bigtable: A Distributed Storage System for Structured Data*", written by Google. The paper detailed a distributed approach to storing and retrieving large amounts of data, which was becoming increasingly important for digital services to be able to do. HBase excels at handling data that is likely to be written rarely but read often.

HBase is a nonrelational database. This means means that the data it stores has no fixed schema that it must conform to. This allows data to be split as required across multiple servers allowing for a distributed server network. HBase is written and interacted with using Java. It does not use SQL, unlike some other common database programs.

Architectural Style

HBase makes use of a layered master/slave architecture. Commands are passed down from high level modules to low level modules. Low level modules are responsible for executing commands and returning information to the module in control of them. High level modules are in turn responsible for maintaining the health of their assigned lower level modules and low level modules are responsible for the health of individual sections of data.

System Evolution

The distributed nature of HBase is a key feature that allows it to scale up to supporting millions or trillions of data entries. HBase is said to scale “horizontally” because it’s capacity and performance can be improved by adding more servers to the system, versus “vertically” scalable systems that improve performance by making a limited number of machines more powerful. Because individual machine performance is limited by physical constraints horizontal scaling is the only reasonable way to support large data sets.

Another important feature of HBase that allows it to scale and evolve as the size of the database increases is the server monitoring provided by the Zoomaster component. Zoomaster provides a way to keep track of the state of servers and identify servers that have failed. Failed servers can then be replaced and data migrated to other servers. The Zoomaster module will be discussed in detail in the subsystems section of the report.

Top-Level Components

The 3 top-level components of HBase master-slave architecture are: **HMaster, Zookeeper and RegionServers.** We will also look and talk about other systems and components as they are interconnected with these components.

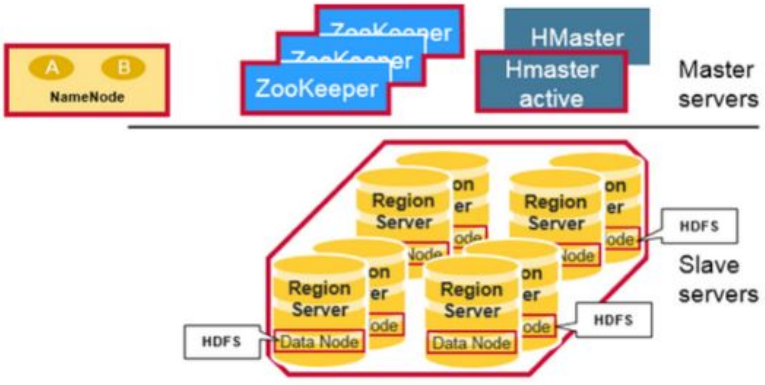

Figure 1: Layered (master/slave architecture overview)

Regions

In HBase data tables are divided horizontally by row key range into what we define as regions. A region includes all the data rows between the start key and end key of the range. HBase tables can be divided into regions in a way that all the columns of a column family are part of that region. One important property of these rows is that they are all sorted. Each region is assigned to a node in the cluster which is the Region Servers. We will see later on what region servers are, but they serve the purpose to allow read and write operation of the data. One region server can have up to 1000 regions assigned to it. You can see an illustration in the diagram below.

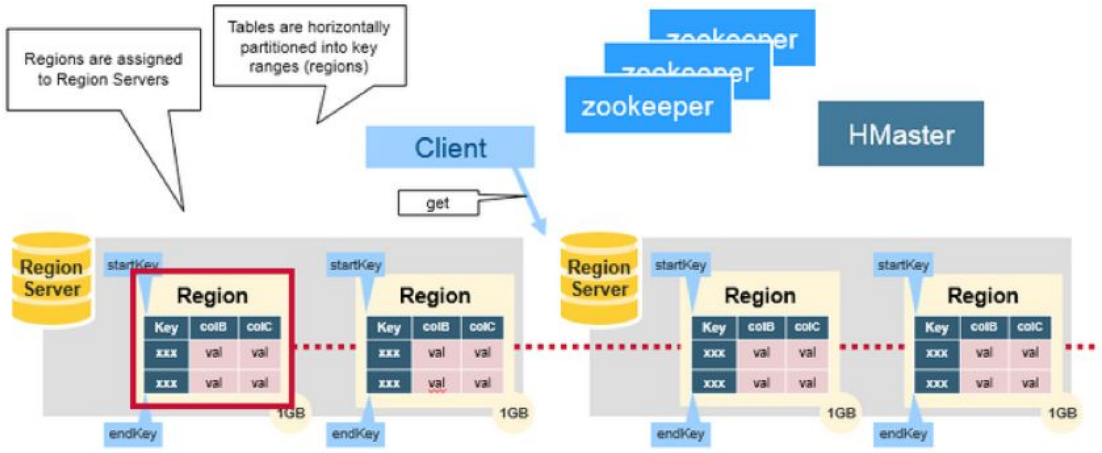

Figure 2: Regions and RegionServers

HBase Master

In the master-slave architecture the HBase Master is one of the (master) servers presents. It's similar to NameNode in HDFS. HBase master is a light weight process whose requirements and duties in the HBase environment can be summarized into two categories; 1. Administrative Operations and 2. Coordinating Region Servers. Administrative operations include DDL and having an interface to create, update and delete tables. The client also uses HMaster when it wants to make changes to the schema or metadata operations are needed to be performed.

The coordinating region servers means HMaster is responsible to managing failover in the Hadoop cluster which could be for load balancing or recovery procedures, it needs to manage and monitor the region servers as well as assign/reassign regions on start-up to a designated region server – HMaster does most of its monitoring of region servers by keeping in touch with and listening to the Zookeeper. We will talk about the second important and top-level components which is Zookeeper and we will see that its played an important part in handling the HBase environment as HMaster alone cannot manage everything.

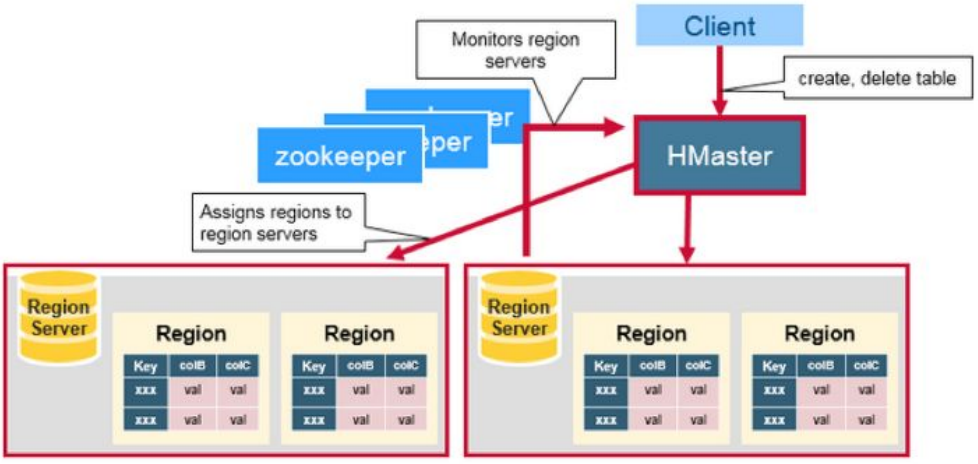

Figure 3: HMaster Component

Zookeeper

The Zookeeper is the next top-level component in the master/slave architecture. It also resides on the master side of the hierarchy similar to HBase master and NameNode. The main purpose of this component is to maintain a live cluster state. The Zookeeper is able to do this through many properties and system in place. Firstly, its designed to be a distributed coordination service which means that there is a mediator or a hub from communicating with the different nodes in the cluster. The Zookeeper is designed to be high availability and also provide a fault tolerant system to maintain the status of servers in the cluster. Since the Zookeeper is a distributed coordination service it requires the use of consensus to guarantee

common shared data. There need to be a 3-5 nodes in the server which approve the consensus which means values are consistent when they are stored and agreed upon by the multiple Zookeeper servers which may be active at that time. The Zookeeper is an important system because it's also the centralized monitoring server which means that its job is to maintain configuration information and provide distributed synchronization. HBase uses Zookeeper as a coordination centralized system as its responsibilities include: initiating communication with the region servers (client contact Zookeeper when they want to connect to a region server), and tracking network or server failure. Zookeeper works with the HMaster providing it with updates and notifications on any failures and region server updates. The HMaster and RegionServers send heartbeats (we will talk about it next) as a method of communication.

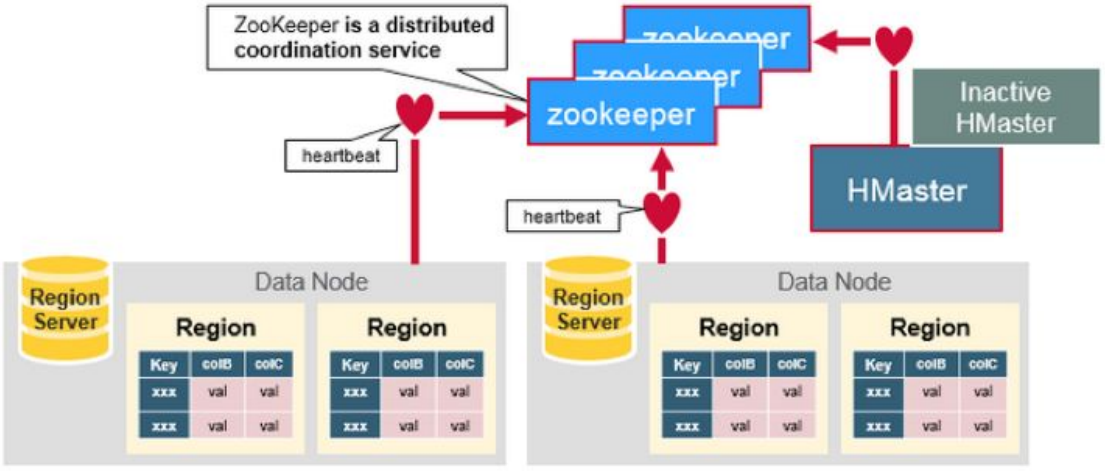

Figure 4: Zookeeper Component

Heartbeat: HMaster and Region servers are registered with Zookeeper service. You can see in the diagram above that heartbeats are sent on a timed interval from the region servers and the active Hamster to the zookeeper (signal between DataNode and NameNode) as a method of communication but moreover this is a confirmation if a node in a particular cluster is dead or alive. This is very important because if the zookeeper does not know of a failed node then it will continue to assign it tasks and/or send it data. A failed node in a cluster is one that has a power or network failure.

RegionServers

These are slave components of this master/slave architecture (as seen in the images and diagrams above). We already talked about regions in the beginning and that a region server is composed of many regions based on range keys. Region servers run on the HDFS Datanode (slave). The RegionServers system is responsible for: (read, write, update, delete) operations from the client. The RegionServer itself has 4 sub components which are: Memstore, Block cache, Hfile, and Write ahead log (WAL). You can see the diagram below which shows the client making a request - the Zookeeper is involved as its directly communicating with region servers and holds the metadata needed. These components will be discussed later on as they will help us better understand the read and write operations. One important thing that the region servers perform is load balancing and this done when an Hbase table in a region becomes too big after a client performs an insert command.

Control and Data Flow

HBase, as mentioned before, is based on the Hadoop Distributed File System (HDFS). As you know, HBase is capable to randomly access and update data stored in the HDFS, but the files is HDFS can only be appended and are immutable after they are created. To understand how HBase provides low-latency reads and writes we will first need to explain in further detail the components of a region server, and their purpose.

I. Region Server: Components

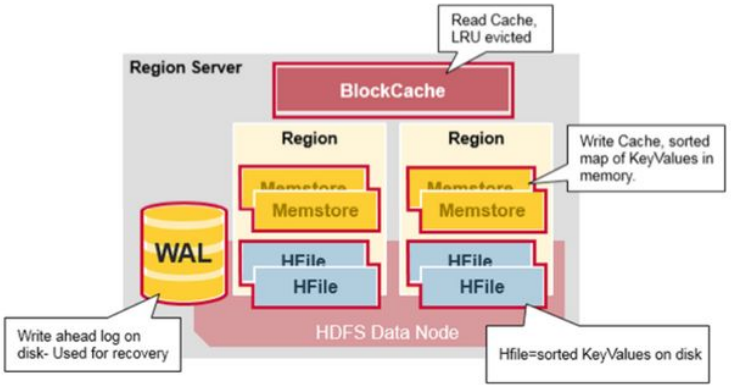

Figure 1: Region Server and its components

1. BlockCache

The BlockCache is the primary mechanism through which HBase can serve random reads with millisecond latency. When performing a scan, the BlockCache is the first thing that is checked. It stores recently used that in memory so that next time accessing data in the same block it can be served by the BlockCache. It helps in reducing disk I/O for retrieving data.

2. Write Ahead Log

The Write Ahead Log (WAL) records all changes to data in HBase. If a RegionServer crashes or becomes unavailable before the MemStore is flushed, the WAL ensures that the changes to the data can be replayed. They are written chronologically, new data is appended to the end of the WAL, so there is never a need to write to random place within the file. As the WAL gets full it is eventually closed and a new WAL will take its place. There is only one active WAL per RegionServer.

3. MemStore

The MemStore is an essential part of the region, there is one for each column family. It is a write buffer where HBase accumulates data in memory (volatile memory) before a permanent write. It acts as an in-memory cache which keeps recently added data. This is useful in numerous cases when last written data is accessed more frequently than older data.
The main reason for the existence of the MemStore is the need to store data ordered by key/Values. To solve this problem the MemStore stores updates in memory by column family sorted by key/values the same as it would be stored in an HFile.

4. HFile

An HFile is the storage format used by HBase it stores the rows as sorted key/values on disk. It's a very rigid type of storage, HFiles are only written in sequential writes, since they cannot alter information already persisted new data is always appended at the end of the file. Nevertheless, for reading, they contain a multi-layered index which allows HBase to seek to the data without having to read the whole file.

II. Data Access

A Client wants to write and read some data.

1. The META table

The META table is the place where the information about the locations of the regions in the cluster is held. It is the Zookeeper that store its location. That is where the Client will start to get its row location.
Obtaining the row location is a three-step operation. When accessing a row for the first time, the Client first must get the Region server that hosts the META table from the Zookeper, when he obtains it, he caches that information along with the META table location. Finally, he gets the row from the server.

This three-step operation is crucial, for from now on, to access that row, the client only needs to use the cache to retrieve the META. There is no need to communicate with the Zookeeper. Although, over time he may need to update its cache with new information.

2. Write Path

Writing something to HBase is not that simple. The one thing we absolutely need is data to persist, so the client will first write to the WAL to ensure that its data won't be lost in case of a server crash. Once it is safely stored in the WAL it is then written to the MemStore who stores it as key/value in memory. When enough data is accumulated everything is written to a new HFile in HDFS. There can be multiple HFiles per column family, as key/values sorted in the MemStore are flushed as files to disk. When flushing the last sequence number written is saved so the system knows what persisted so far, it is stored as meta field in each HFile that way, on region startup, the sequence number is read, and the writing can continue where it was left off.

3. Read Path

Reading data is much easier than to write it. We have seen that key/value cells corresponding to one particular row can be in three different places: recently read files in the BlockCache, recently updated cells in the MemStore and, of course, in the HFiles. As so, to read data we only need one of those three to get us the data we want so we do a read merge of the three until it finds the target row cells. We start by the easier and faster one, the BlockCache, next we scan in the MemStore, and if none of those proved to be worth the scan we load HFiles into memory.

Data Dictionary (Glossary)

NameNode: HDFS cluster has one node - a master server that manages the file system namespace
DataNode: One node per cluster that is responsible to manage storage connected to the nodes - slave servers in the cluster (more than one)
HDFS: Stands for - Hadoop Distributed File System. This is the primary storage that deploys a master/slave architecture for a distributed file system for fast data access that has high availability and scalability. HBase is on top HDFS.

Concurrency

As discussed earlier in the report, Apache HBase is primarily made for large datasets, this means that concurrency is an obvious boon to have in terms of performance. Despite this, HBase is not ACID compliant, but guarantees ACID compliance per row. This is mainly done

for performance reasons, as guaranteeing ACID compliance across all the data in the database would take a toll on HBase's overall performance.

Multiversion Concurrency Control

HBase achieves concurrency and ACID compliance across it's rows using Multiversion Concurrency Control. MCC focuses especially on preserving the C and I of ACID, that is, Consistency and Isolation. Consistency means that applying the same operation will always yield the same consistent result, regardless of circumstance. This aids in our understanding of what should currently be in the datastore, and is a requirement to have in a database that is meant to be accessed at any time. Isolation means that the operations that happen concurrently do not affect each other. These two terms are similar, and play hand-in-hand, but are not identical.

The way MCC achieves this is through a simple locking mechanism over rows currently being accessed. Imagine a row in a dataset somewhere in an HBase installation. We will use this to simulate our concurrency use case.

Write Process

1. A write operation is requested on this row, this means that the row is locked and cannot be accessed by other write operations
    a. A unique write number (ascending) is assigned to the write operation
2. The Write-Ahead Log (WAL) is updated. This is done in case recovery of the data is needed in the case of server shutdown or failure.
3. The MemStore is updated
    a. The write number is released and deemed finished
4. The row lock is released

Read Process

Now in this intricate process, if a read operation is sent out, the following process occurs:

1. The read operation is assigned a read timestamp, known as the read point
2. The read point is assigned to be the highest number such that all writes with write numbers less than or equal to the read number have been completed
3. The read operation returns the data that is requested that correlates with the last completed write operation, based on last completed write number.

Competition

Cassandra

Similar

- Both incorporate NoSQL which allows for more information to be stored as there is no restriction on there being a specific attribute for every piece of entered information.
- They both use variations of a wide-column mechanic in storing information
- Both descendants from Google's BigTable
- They have very similar security checks which also come from being children of Google's BigTable

Different

- ➢ Different in that Cassandra does not incorporate the master-slave relationship that HBase uses between the HMaster and the Region Servers
- ➢ Cassandra has consistent data replication in case of emergencies, HBase has it optional
- ➢ Because of the above, if any component dies, the system can still manage with the backups and so, Cassandra has little-to-no downtime. Unlike HBase where if the HMaster goes down, then there is a sort of a bottleneck in the system while the HMaster is connected
- ➢ Cassandra has its own Query Language, CQL, meanwhile HBase needs external applications to aid the query searches

Apache Kudu

Similar

- Kudu uses HBase/HDFS update/insert/write/scan speeds
- Also incorporates the wide-column storage mechanic
- Since Kudu uses HBase's Update/scan speeds, then their IO and CPU process allocation systems are similar to provide the required speed

Different

- ➢ Kudu, on top of the adopted speeds mentioned before, decided to incorporate feasible analytic speeds into their database as well meanwhile HBase didn't have that focus
- ➢ HBase uses column families while Kudu doesn't
- ➢ Kudu has a lower max size per row for the amount of data withheld in them to allow for feasible analytic speeds at the cost of max amount of data per row

Amazon Redshift

Similar

- Both use the ‘sharding’ partitioning technique where data is split up into many tables that each deal with certain categories of attributes. This allows access to different sorts of information from a variety of possible pathways.
- They both support concurrency with regards to the information being accessed by multiple users
- Both immediately consistent with an all-or-nothing mechanic to ensure stability in data. Immediately consistent means that the information is placed immediately and feedback is given immediately. All-or-nothing means that, either all the information is placed in the database when it gets processed or, if an error occurs, then all that information being processed is not placed in the database. This ensures that no corrupt or incomplete data is stored in the database.

Different

- ➢ Redshift is more for BI (Business Intelligence) tools as a result of their focus on analytics.
- ➢ HBase is a non-relational database but redshift is a relational one. This aids in the analytics aspect as data is reached much easier this way.
- ➢ Data replication is always a given with Amazon’s Redshift. HBase provides the option for data replication but it’s not always used.

Redis

Similar

- Similar to redshift as well, Redis uses the sharding partitioning mechanism.
- Redis also adopts the master-slave architecture as well as the data replication technique used in unison with this architecture
- They both also allow concurrent access to information from clients

Different

- ➢ Redis guarantees *strong* eventual consistency meanwhile HBase conforms to immediate consistency. What eventual consistency entails is that with concurrent access to information, then eventual consistency will return the required information but at the risk that the information might not be completely up-to-date as the replicated data has not been updated. *Strong* eventual consistency means that data will not be returned until the data has been updated.
- ➢ Redis is a relational DBMS unlike HBase’s non-relational DBMS.
- ➢ Redis does not use the MapReduce functionality like HBase.

References

https://www.janbasktraining.com/blog/hbase-architecture-main-server-components/

https://mapr.com/blog/in-depth-look-hbase-architecture/

https://blogs.apache.org/hbase/entry/apache_hbase_internals_locking_and

https://hbase.apache.org/acid-semantics.html