A case study on Apache HBase

Abstract

Introduction:

Apache HBase is an open-source, non-relational and a distributed data base system built on top of HDFS (Hadoop Distributed File system). HBase was designed post Google‟s Big table and it is written in Java. It was developed as a part of Apache‟s Hadoop Project. It provides a kind of fault – tolerant mechanism to store minor amounts of non-zero items caught within large amounts of empty items. HBase is used when we require real-time read/write access to huge data bases. HBase project was started by the end of 2006 by Chad Walters and Jim Kellerman at Powerset.[2] The main purpose of HBase is to process large amounts of data. Mike Cafarella worked on code of the working system initially and later Jim Kellerman carried it to the next [2] stage. HBase was first released as a part of Hadoop 0.15.0 in October 2007 . The project goal was holding of very large tables like billions of rows X millions of columns. In May 2010, HBase advanced to a major project and it became an Apache Top Level Project. Several applications like Adobe, Twitter, Yahoo, Trend Micro etc. use this data base. Social networking sites like Facebook have implemented its messenger application using HBase. This document helps us to understand how HBase works and how is it different from other data bases. This document highlights about the current challenges in data security and a couple of models have been proposed towards the security and levels of data access to overcome the challenges. This document also discusses the workload challenges and techniques to overcome. Also an overview has been given on how HBase has been implemented in real time application Facebook messenger app.

Acknowledg ment

I, Rohit Reddy Nalla want to express my gratitude to Dr. Sam Sengupta, the advisor of this project has been a constant inspiration and has offered guidance in every possible way one could imagine. I would also like to thank my family and friends for their encouragement and support.

Table of Contents

Chapter 1

Introduction to Database systems

1.1 Introduction

A Data base is a place where data is stored in an organized way. The stored data is used across various aspects of an application which helps in processing the requests efficiently. In the recent times, usage of internet has brought some revolutionary changes across the world and many online websites and applications have been developed to give users more benefit. While developing an application (or website) maintaining a data base has become relatively important. Maintaining the customer‟s data and catalogue has become a challenging task since then. That is when Database management systems (DBMS) came into picture.

Database management systems (DBMS) are kind of software applications which allow implementing CURD operations i.e. create, update, read and delete the data in tables. It interacts with end users, third party applications, and internal applications within the system and with the database itself to capture and process the data efficiently. MySQL, PostgreSQL, NoSQL, Oracle etc. are some of the famous data bases. Generally depending on the database models they support DBMSs are classified. Most of the well-known data bases have supported relational model which has been represented by SQL since 1980‟s. Initially in 1960‟s some General-purpose DBMSs have typically served the needs of many applications and they were cost effective too. However sometimes it has introduced some unnecessary overhead and later there was a realization that it was not a favorable one to use. Then special-purpose DBMSs came into picture which served the purposes. To make use of email system in the most effective way, special-purpose DBMS were used to handle email systems. As days passed on there were lots of advancements in terms of technology perspective. In the areas of computer storage, networks and memory the performance expectations of the DBMS have been started growing. In mid 1960"s "Data Base Task Group" has come up with an navigational approach (CODASYL approach) where records in the data base can be found by navigating relationships from one set of records to another set of records and they were searched in a sequential order. However in further validations, it was proved to be a difficult procedure and it requires rigorous workout in order to serve applications.

There was a growing concern on handling large data banks. An efficient service should be given to the users in such a way that their activities should not be affected though there is any change in the internal or external representations of their data. In this context, in 1970"s Edgar Codd wrote numerous papers that outlined a new approach to the database construction that found a break through by *A Relational Model of Data for Large Shared Data Bank*[5]. He described a new approach for maintaining large data bases and for storing data in it. Unlike navigational approach where records are stored in linked lists, he came up with storing records in tables of fixed length, where each table is considered as one unit. For example, to create a table with user account information such as user ID, password, contact no., date of birth etc., user ID should be unique so that all the other fields are stored by putting the user ID as reference key or primary key. Whenever if we want to collect this information, we can search for it with the help of a primary key so that information stored in the optional tables can be found with the help of this key. With the help of Tuple Calculus (branch of mathematics) he has developed a system which can maintain the operations of a regular database like CURD operations and also it can sort the datasets in one single operation. Based on the Codd"s theory IBM had started working on a modal named System R. Their idea was to develop multi-table systems, in which all the data which is stored for record could be split instead of storing the whole data at one large place.

They implemented Codd"s concepts successfully and they have created a product of System R known as SQL or DS. Later it became Database 2(DB2). In later 1970"s based on IBM"s research Larry Ellison also came up with Oracle database. These Relational Database systems (RDBMS) have become popular and were used for a small scale to medium scale applications. SQL language is used for writing queries in order to implement the basic CURD operations. It has powerful features and query languages like MySQL and PostgreSQL. Secondary indexes can be easily created and it performs several inner and outer joins, count, sorting across the rows and columns in data base tables. However if we need to measure or scale up with large amounts of data and random read/write access, then we"ll find difficulty in performing these actions with RDBMS. It makes distribution of data complex with ACID properties and Codd"s 12 rules.

To overcome these drawbacks then came the next generation databases known as NoSQL databases. These databases are kind of document-oriented databases depend upon data attributes in the XML document while querying. XML"s are primarily used to for interacting with multiple systems. NoSQL databases don"t have any fixed table schemas and they scale horizontally. HBase is one of the NoSQL database which is an open source database with a distributed column-oriented built on top of HDFS. It is mainly used to for large data sets to give read/write access in real time. Though it has not been a direct alternative for traditional databases it performance has been deliberately improved in the recent times and it has been playing a key role in number of data-driven websites and web applications like face book messenger etc. In the coming section I have discussed about HBase and how it works.

Chapter 2

Introduction to HBase

2.1 Background

HBase was developed as a part of Apache‟s Hadoop project. It was designed post Google‟s Big Table and it was written in Java. HBase project was started by the end of 2006 by [2] Chad Walters and Jim Kellerman at Powerset. . The main purpose of this project is to process large amounts of data and provide a kind of fault-tolerant mechanism to store minor amounts of non-zero items caught within large amounts of empty items. HBase is primarily used when real-time read/write access is required for huge data bases. Mike Cafarella worked on code of the working system initially and later Jim Kellerman carried it to the next stage. HBase was first released as a part of Hadoop 0.15.0 in October 2007. [2] The project goal was holding of very large tables like billions of rows X millions of columns. In May 2010, HBase advanced to a major project and it became an Apache Top Level Project. Several applications like Adobe, Twitter, Yahoo, Trend Micro etc., use this database. Social networking sites like Facebook have implemented its messenger application using HBase.

2.2 Data Model

In this section a brief overview has been given on HBase data model for a better understanding. Generally an application (or a web application) stores all its content or data in the form of tables in data base. Here in HBase the table cells i.e. the intersection of rows and column coordinates are timestamp versioned which are assigned automatically by HBase at the time of inserting the cell into the table. Thus each cell has a time stamp version and the content of the cell is an array of bytes. Each row in the table has row key and the rows keys are represented by byte arrays. Thus anything from string to serialized data structures can represent the row keys and the sorting is done based upon these row keys. In the same way each table has a primary key which is used to access all the information in that table.

In HBase Rows and columns are congregated into column families. Columns belonging to the same family will have a common prefix so that they can easily be recognized and sorted. For suppose, assume there are two columns with names sport: cricket and sport: soccer. These two columns comes under the sport column family since both of the columns belongs to common group sport. In the similar way if we have another column like product: dell then this will come under the product column family. These column family names can be made of any random bytes. These columns families are designed as a part of table schema in the beginning itself. Thus if any client wants to add another column sport: baseball then can be added to column family sport, provided column family sport is already persisted in the table. All these column family members are stored on the file system. As we discussed earlier that HBase is a column-oriented store, we can also call it has a column-family-oriented store since all the storage specification and modifications are done at the column family level and it‟s better to have the general characteristics and accesses for all the column families.

Tables are auto-separated by HBase into regions. Each region has a subgroup of table rows. Thus a region is denoted by the table name and its first row and last row. Initially a table consists of single region but when the size of the region grows and after a specific increase of size then it will be split into two new regions of equal size. Until the split whole load of the region will be handled by only one server hosting the original region. Regions are the units that get circulated over the HBase cluster of servers. Thus whenever a table size increases then the region size will be increased and that results in distribution of the data load across the cluster of servers with each node carrying the part of the total tables region. In this way the excessive load of the table gets distributed.

The below 4dimensional data model will give an understanding of the above concepts.

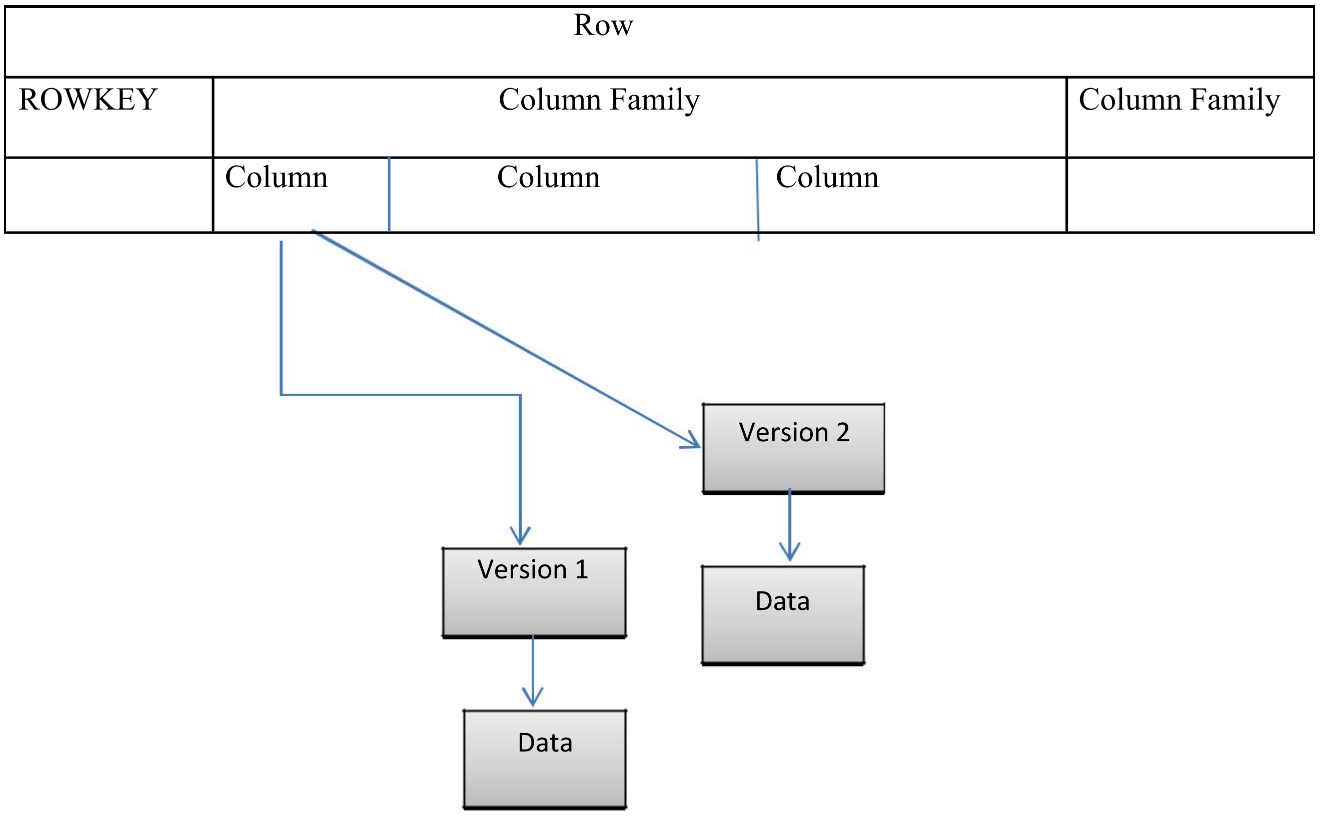


Figure 1 – Four Dimensional Data Model

As we discussed earlier in the above figure we can see each row is represented with a unique row key and it is treated as a byte array. The data in the row is structured into a column family and each row has the same set of column families. The column families have to be defined upfront while designing the schema so that later we can update an extra field of the same column family easily. Column families describe the actual columns and they are also called as column qualifiers. Each column has versions so that one can access data based upon the specific version for a column qualifier. Thus in order to access a data we need to give its row key, column family, column qualifier and version.

2.3 HBase Implementation and Operation

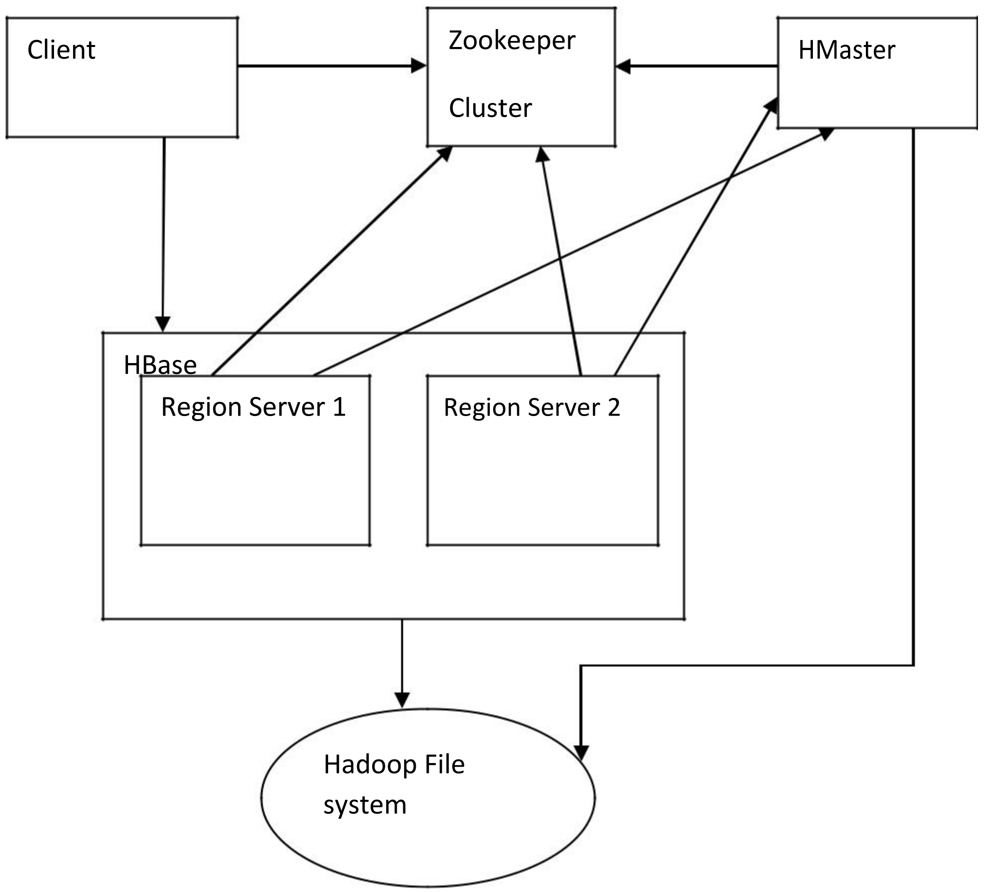


Figure 2 – HBase Model [2]

HBase is fabricated on clients, slaves and a master node (known as HBase master node) which coordinates with the region server slaves. The architecture of HBase is similar to HDFS and Map Reduce. The HBase master node usually assigns regions to the region servers which are the registered ones and are also responsible to recover the failed region servers due to load issues. The region servers carry the regions from one to many depending upon the data load and they are also taking care of the read/write requests from the client. Whenever there is an excess load on a region then it will be split into new two region and the region servers informs the master node about this split and the new daughter regions so that the assignment of the replacement daughter regions would be easy.

HBase depends upon Zookeeper which is an open-source server and acts as centralized service for maintaining configured information, group synchronization and provides group services. Zookeeper deal with an instance of it and it acts as an authority on the whole cluster state. HBase deals with the fundamentals such as hosting the location of the root catalogue table and address of the current cluster master. Usually assignment of regions are taken care by the participating servers and at some instances it is done via zookeeper when these servers got failed in the middle way. The advantage with the Zookeeper is it has the ability to pick up (or recover) the assignment transaction from the state where it has been left by the crashed server. At least if you want to bootstrap a client side connection to the HBase cluster, then the location of the Zookeeper band should be provided to the client so that the client can navigate through the zookeeper hierarchy to get to know about the servers and others attributes. In HBase the region server slave nodes are listed or programmed in HBase configuration/regionservers. SSH based remote commands are used to run the start and stop scripts. Cluster based specific configuration are listed in the HBase configuration/ hbase-site files with .xml or .sh extension. Usually HBase preserves data via hadoop file system API. Though there are multiple file systems to preserve data it goes with the HDFS. However sometimes it writes to the local file system too by default. With the initial installation of HBase we can go with the local file system but later on it‟s better to use the HDFS cluster.

HBase usually has two catalog tables internally to uphold the current data, location and state of all the regions present in that particular cluster. These tables are called as –ROOT and .META. [2] The ROOT table holds all the information of the table regions of the .META and META table holds all the list of user-space regions. Data entry into the tables is classified or keyed by the region name which is the table name the regions belongs to, start row of the regions, timestamp (time of creation), and the cryptographic hash function MD5 of all the former values. If there are any changes in the regions like the disabling or enabling, splitting, redeploying the region due to the server crash all these changes will be updated in the catalog tables so that they can maintain the current state of all the regions in a cluster.

The process is that clients who are hitting the HBase for the first time connect to the Zookeeper cluster initially to get the location of the catalog table –ROOT and then will obtain the location of the .META region from the root table. Then the client searches for the location of the hosting user-space region which is in the scope of .META region. In this way the client who visits for the first time interacts directly with the regional server. Instead of following all these repeated process for the new clients, caching the entire learnt pass through –ROOT and .META locations and the start stop rows of the user-space regions will be good enough so that they can easily get to know the hosting regional servers. In this way they can avoid the process of going to the .META table every time to get the information. Thus they can easily know it from the cache and this process can be continued until there is an error. When there is an error or a region has been moved then the client consults the .META to learn about the new location and if that is also been changed then it consults the –ROOT to figure out the location and this all information will be cached so that it will be helpful for the other upcoming clients.

The writings (also known as logs) which we get at the region server are primarily attached to the commit log which in turn is added to an in-memory known as memstore. If the memstore is full then its content is pushed into the file system. This commit log will be hosted on the HDFS so that it will be available at the time when the region server crashes too. When the master node is unable to reach out the region server as it got crashed, then it just divides the commit logs of the dead server by its regions. While reassigning, the regions which were on the dead server range will pick these commit logs and runs them to get back to their original state which was like before the failure. This helps them to work more efficiently in instances like server failure.

2.4 Running HBase:

2.4.1 Understanding of basic queries for table creation and updates:

In this section we are going to discuss briefly about the running HBase and creating tables and updating them.

HBase is an open source which can be downloaded from the Apache website which has multiple mirror sites to download [6]. Usually the HBase team recommends all the uses/developers to install the HBase on Linux or Unix platforms. If it has to be run on the windows platform then they recommend to install Cygwin. It provides the integration of the windows based applications, tools with the Unix based applications and tools. It provides a Unix like environment and a command line interface for windows.

We can interact with HBase by launching the HBase shell by typing „./hbase shell" in the directory. The output more or less looks in the similar way as shown below

```
HBase Shell; enter 'help<RETURN>' for list of supported commands.

Type "exit<RETURN>" to leave the HBase Shell

Version 0.98.5-hadoop2, rUnknown, Sun Tues Apr 4 23:58:06 PDT 2015

hbase(main):001:0>
```

Now let us create a table called "Songs List" in HBase with a column family "name".

```
hbase(main):002:0> create 'SongsList', 'Songdetails'

0 row(s) in 4.2150 seconds

=> Hbase::Table - SongsList
```

The above statement has created a table "SongsList" and as I said earlier we need to create a column-family for each table and we have named it as "Songdetails" for the above table.

If we want to look at the table info we can use "list" to fetch the details of the table as shown below

```
hbase(main):002:0> list
TABLE
SongsList
1 row(s) in 0.0350 seconds
=> ["SongsList"]
```

If we want to know more information about the tables like the versions, size, block cache, list of column families etc., we can use "describe" command.

To add some data into the table we use “put” command in HBase as shown below

```
hbase(main):004:0> put  'SongsList',  'rowkey1',  'Songdetails:name',
'Itsyourlife'

0 row(s) in 0.0850 seconds
```

Here we have added data into a new row with rowkey as “rowkey1”. Here the “Songdetails” is the column family in which we add a column “name” with value “Itsyourlife”.

As discussed earlier, we can pull out the entire row details using the “get” command and rowkey attribute “rowkey1” as shown beloww

```
hbase(main):005:0> get 'SongsList', 'rowkey1'
COLUMN                        CELL
Songdetails:name              timestamp=1410374788088, value=Itsyourlife
1 row(s) in 0.0250 seconds
```

We can see that under column we have the columnfamily “Songdetails” with “name” as column and its value as “Itsyourlife” with timestamp when the data was entered.

We use “Scan” command in order to fetch all the rows in the table. For that first let us add one more row to the table “Songdetails” as shown below:

```
hbase(main):006:0> put 'SongsList', 'rowkey2', ' Songdetails:name',
'We will rock'

0 row(s) in 0.0050 seconds
```

Since we have two rows now we can use “scan” command in order to retrieve all the rows from the table

```
hbase(main):007:0> scan ' SongsList'

ROW      COLUMN+CELL

rowkey1 column=Songdetails:name, timestamp=1410374788088, value=Itsyourlife
rowkey2 column=Songdetails:name, timestamp=1410374823590, value=We will rock

2 row(s) in 0.0350 seconds
```

We can observe that using "scan" we got all the rows present in the table. But if we want to retrieve a particular row from the table then we need to add a condition such as start row and end row in order to retrieve details of certain rows. To observe the difference let us add another row key which has a rowkey and it starts with letter „t" as shown below

```
hbase(main):012:0> put 'SongsList', 'trowkey3', 'Songdetails:name', 
'never together'
```

Now we will add a condition like we want all the rows with a rowkey greater than r and less than„t" as shown below

```
hbase(main):014:0> scan 'SongsList' , { STARTROW => 'r', ENDROW => 't' }
ROW     COLUMN+CELL
rowkey1    column=    Songdetails:name,    timestamp=1410374788088,    value=
Itsyourlife rowkey2 column= Songdetails:name, timestamp=1410374823590,value=
We will rock 2 row(s) in 0.0080 seconds
```

By running the above command it returned all the rows which started with r but not the row which has started with t. The comparison is based upon the complete row key and since the rowkey1and rowkey2 are greater than r they have displayed these 2 rows. In the statement we have given the end row as „t". Thus it excludes all the rows which starts with row key „t". Thus if we want all the rows then we can exclude the end statement and it displays all the rows which are greater than „t".

In this way we can create tables with column families and update the data. The important point to be noted is that we need to create column families before itself in order to add the columns to the families later. If not the table has to be offline in order to add the column families first which shows severe impact on the work load in real time environment.

2.4.2 Connecting HBase with Java

In this section we are going to discuss about the creation of tables and data manipulation using Java. Before that I just want to discuss some of the predefined Java classes provided by HBase which are used while connecting with HBase and for data manipulation.

In Java to manage HBase i.e. to create, list and drop the tables we use HBaseAdmin class. This class is a pre-defined class provided by the HBase. The syntax is given below:

```
public class HBaseAdmin
extends object implements Abortable, AutoCloseable , Closeable[4]
```

HBaseAdmin implements these interfaces in order to perform the operations like create, enable, disable, list and drop the tables. Abortable is used to terminate the server or client whereas the Closable is used to close the data source. When we implement the closable interface, close() method is invoked which closes the stream of the data and releases the resources which are associated with the object.

In order to update, add and delete data HTable class is used and it is provided by HBase. The syntax is given below:

```
public class HTable extends Object
implements HTableInterface [4]
```

It implements HTable interface which has methods like put, delete in order to add or delete methods.

In order to access a table in HBase we can do it by creating an instance of the HTable as shown below:

```
Configuration config = HBaseConfiguration.create();
HTableInterface songsListTable = new HTable(conf, “SongsList”); [4]
```

HBaseConfiguration has a method create() which creates a configuration with the HBase resources with the help of a config object. If we want to change the connection parameters in order to run the HBase on a remote machine then set() method can be used. This config object and table name are passed to the constructor by instantiating the HTable object in order to access the table. In order to perform all these operations in java using the pre-defined classes provided by HBase we need to download and add the hbase-0.94.8.jar to the class path. Also we need to add some of the jars related to Zookeeper and Hadoop to the class path. Now let us write a java program for creating a table:

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.HColumnDescriptor;

import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class HbaseConnection
{
 public static void main(String[] args) throws IOException
 {
  HBaseConfiguration hbc = new HBaseConfiguration(new Configuration());

  HTableDescriptor htd = new HTableDescriptor("Songslist");

  htd.addFamily( new HColumnDescriptor("Id"));

  htd.addFamily( new
HColumnDescriptor("songName")); //adding colums

  HBaseAdmin hba = new HBaseAdmin( hbc ); //connecting

  hba.createTable( htd ); //done with table creation

 }
}
```
[6]

When we run the above program "Songslist" table has been created by invoking the constructor of HTableDescriptor class. We have two added two columns i.e. "Id" and "songName". Here ID acts as a row key.

Now let us add some data to the “SongsList” table using the put constructor and display the columns using the scan as shown below:

```
import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.client.Get;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.ResultScanner;

import org.apache.hadoop.hbase.client.Scan;

import org.apache.hadoop.hbase.util.Bytes;

public class HTableExample

{

 public static void main(String[] args) throws

 IOException {

  /* When we invoke the create method in HBaseConfiguration, it reads
the resources which are present in the hbase-site.xml and in hbase-
default.xml, and we need to make sure that these xmls are present in
the class path.*/

org.apache.hadoop.conf.Configuration              config              =
HBaseConfiguration.create();

/*This instantiates an HTable object that connects you to the table*/

HTable table = new HTable(config, "Songslist");

/* We are calling the put constructor in order to add a row by
instantiating the put object. The constructor converts the string into
byte array. */

Put p = new Put(Bytes.toBytes("row1"));

/* Now in order to set the data in the row we need to specify the
column family, qualifier and value by calling the add method.

Note: In order to add the columns, the related column families should
be present already in the table. */
p.add(Bytes.toBytes("Id"),Bytes.toBytes("col1"),Bytes.toBytes("1"));

p.add(Bytes.toBytes("songName"),Bytes.toBytes("col2"),Bytes.toBytes("I
tsyourlife"));

table.put(p);

// To retrieve data

Get g = new Get(Bytes.toBytes("row1"));

Result r = table.get(g);

byte [] value1 =
r.getValue(Bytes.toBytes("Id"),Bytes.toBytes("col1"));

byte [] value1 =
r.getValue(Bytes.toBytes("songName"),Bytes.toBytes("col2"));

String Str1 = Bytes.toString(value);

String Str2 = Bytes.toString(value1);

System.out.println("Id: "+ valueStr+"Name: "+valueStr1);

Scan s = new Scan();

s.addColumn(Bytes.toBytes("Id"), Bytes.toBytes("col1"));

s.addColumn(Bytes.toBytes("songName"), Bytes.toBytes("col2"));

ResultScanner scanner = table.getScanner(s);

try

  {

   for (Result rr = scanner.next(); rr != null; rr = scanner.next())

   {

    System.out.println("row exists" + rr);

   }

  } finally

  {

   // Closing the scannner

   scanner.close();

  }

 }
```

```
}
```
[6]

After running this program in eclipse IDE as java application, we can see the output by typing the scan command

```
hbase(main):007:0> scan 'Songslist'

ROW  COLUMN+CELL

row1 column=Id:col1, timestamp=1378897861521, value=1

row1 column=songName:col2, timestamp=1378897861521, value=
Itsyourlife row(s) in 0.0875 seconds
```

In this way we can connect to HBase with Java and its better to write such programs in maven to manage the dependencies.

Chapter 3

Current Data Vulnerability and Workload challenges

3.1 Current issues in Data Security

As per the features of HBase, it has gained lot of interest from data management solution companies with new perspectives, requirements and challenges. The default HBase Configuration allows everyone to read and write data to all the tables in the database which is not a good aspect for the companies who are implementing HBase. It has been a major concern for various users like the health care providers who don"t want to share their patient"s information with everyone. In the same way there are many government organizations who want to give access to sensitive data only to a certain section of employees within their organization. To overcome this, firewalls can be setup by the admin in order to decide which machine has to be given access to communicate with HBase. But once these machines got access to communicate it can write and read data from all the tables. These machines are not restricted to use a section of tables. Thus in this section we will discuss some of the security features which HBase has provided for authentication and access to a certain data only and also a brief over view has been given on some of the research work done in companies like Intel on these data security issues.

3.2 Security model based on Kerberos Authentication and Access Control Lists (ACLs)

3.2.1 Introduction:

In the initial HBase releases it had the ability to protect the data against the unauthorized users, hackers and sniffers. However it didn"t have the ability to stop the authorized users from accessing the data from all the tables in the system who can accidentally or intentionally delete or modify all the data. HBase configuration allows only authenticated users to communicate with HBase systems. This authorization is achieved with the Simple Authentication and Security layer (SASL) which takes care of authentication and message encryption on the basis of per connection and is based upon the Kerberos protocol. We will discuss about Kerberos in the coming section. This authorization is based upon the Remote Procedure call protocol (RPC). Once the user authentication is done then the next target is to allow the user to access only certain sets of data tables. For this purpose HBase has brought in Access Control List (ACL) which describes set of rules for user to authorize a particular set of data. In ACL each entry describes the user and the data to which he has the access. Thus whenever a user requests for an access it uses the authentication mechanism which is based on Kerberos protocol to identify the user identity and then it checks in ACL for its appropriate entry in order to give the access to the data. It defines the authorization rules with a table or a column family or a qualifier. Users can have this ACL from HBase 0.92 jar API.

3.2.2 Kerberos protocol

Kerberos is a protocol which authenticates the identity of user and server across a non-secure network. This is a kind of client-server model. It uses cryptography keys like AES for a client to prove its identity to server and the same way for a server to prove its identity to client across a non-secure network. The whole process happens in the form of tickets and once their identities have been proved the communication between them is also encrypted in order to safeguard their data and business. Let us see how a client proves its identity and get an access to a service in below steps:

- Initially the client sends an authentication request to Kerberos Authentication center and proves its identity by receiving a Ticket Granting Ticket (TGT).
- Now in order to communicate with the server the clients sends a service request to Kerberos Authentication center and if the service request is valid then it sends the response via ticket and a session key.
- Then the client with the help of this service ticket contacts server for service.

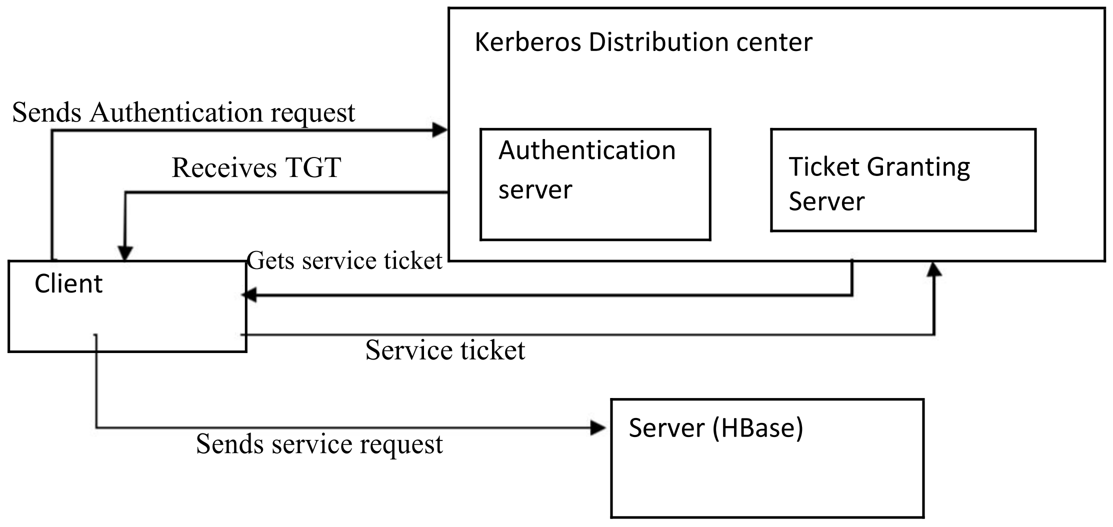


Figure 3 – Kerberos model [8]

Moreover for HBase in order to communicate with HDFS and Zookeeper, a service session has to be created which has to be more secure. The files which are written by HBase are stored in HDFS. HDFS provides access based upon the users and permissions. Thus the files which are written by HBase will have user as “hbase”. However the access control is based upon the user name and thus every user who wants to access the system for service has to get the security privileges as the user “hbase”. Thus HDFS will create some authentication steps which give assurance that the user “hbase” is a trusted one. In the same way Zookeeper also has an Access Control List (ACL) which gives access to users based upon the user name.

Since the users have successfully authenticated via Kerberos protocol, now the users should be restricted to use only a certain sets of data for which he has access but not for all the tables in the system. For that HBase came up with an Authorization mechanism where it has enabled the Access Controller coprocessor (ACC) by adding it to the HBase configuration files for the coprocessor classes of master and the region server. Coprocessor runs random code and executes it before and after the operations like put, deletes and get. Thus using this random code ACC stops all the operations (like put, get and delete) and it checks whether the user has access or rights to this data and then it allows executing the operations. If the user does not have the rights then it won‟t allow performing all these operations. In this way it restricts the user to certain sets of data based upon the rights and access he has. The below flow diagram explains the process.

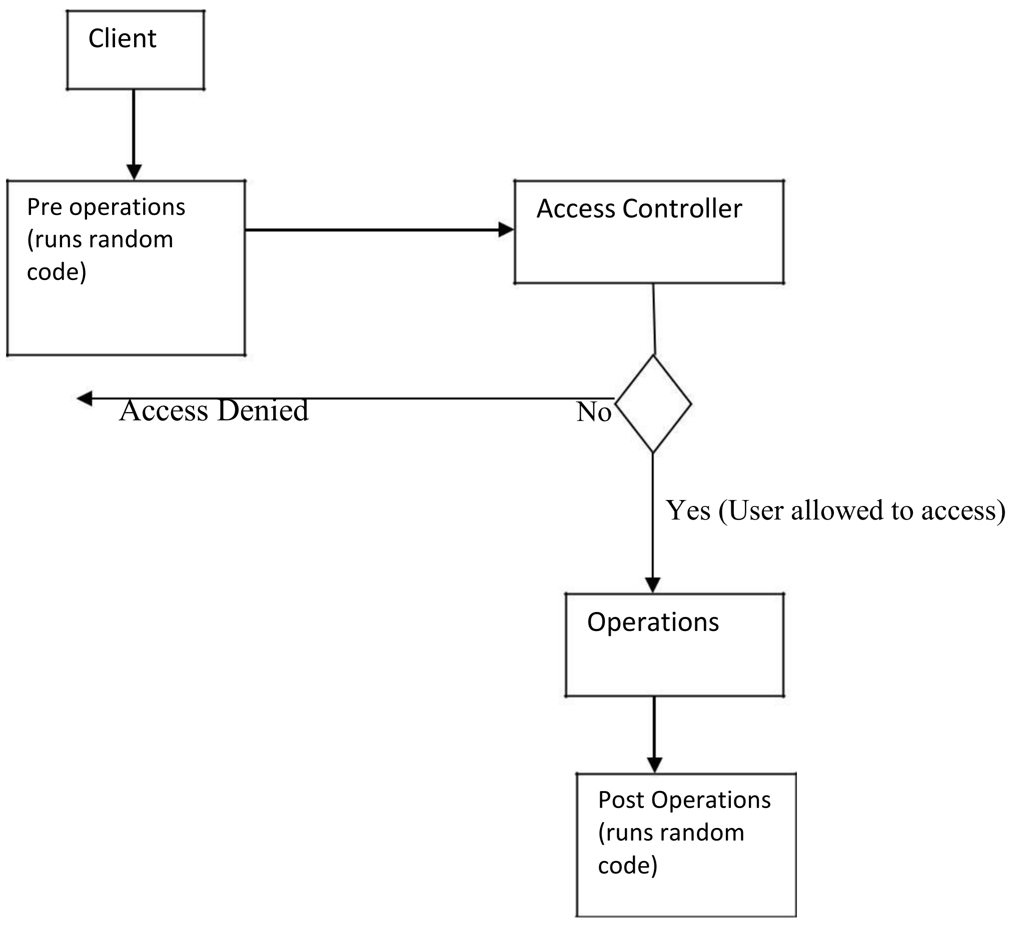

Client
Pre operations (runs random code)
Access Controller
Access Denied
No
Yes (User allowed to access)
Operations
Post Operations (runs random code)

Figure 4 – Flow diagram

The admin can manage the user rights by using some commands which are present in HBase shell:

```
Grant [table] [family] [qualifier] [4]

Revoke [table] [family] [qualifier] [4]
```

For instance admin can restrict user to a certain sets of data in a table named "Songslist" using a grant command as shown below:

```
grant [User-X] [R] [Table-Songslist] [Family-Artist]
```

The above grant command tells that the admin has given rights to user X to read table "Songslist" and also the column family "Artist".

By then this was still a basic security feature proposed by HBase and they were trying to contribute more to this model. This was tracked under ticket **HBASE-6096** in JIRA [5].

Though the primary goal of giving the access to a user for only certain sets of data was achieved with this security model, there were some concerns on the levels of access. With this model, a user can be granted access to a table or a particular column family. As discussed earlier, all the data which is written into HBase systems are stored into cells. If a user has been granted access to a column family then he can access all the cells in the column family. However this model does not support the feature of giving access to a particular cell in the column family. There were some other researches also going in Yahoo and in 2009 they have proposed a Hadoop security model which supports the user authentication but it didn"t meet the cell security requirement. Then at Intel, developers were inspired by the HBase security model based on Kerberos and they started working on a model which offers cell level access and security. In the next section we are going to discuss about this model.

3.3 HBase cell level security model

3.3.1 Cell Tags:

All values which are written to HBase are stored in cells. These values are stored in the form of key/value pairs. Additionally a tag has been introduced which is associated with the cell. These tags can store the metadata for a cell. In order to persist this information a new HFile version 3 has been created.

3.3.2 Cell ACL’s:

A new security feature can be added on top of cell tags which are the cell ACL‟s and Visibility expressions. Previously we have used Kerberos protocol and Coprocessor mechanism to give access to the user in order to access a table or a column family level. In this security model, access will be given to the user only to a particular cell in which he wants to read or write the data. He won‟t get access to all the other cells in the column family. It is Backwards compatible with existing installs and code and that is an additional feature. It uses the existing operations like put in order to add the cell ACL to the tag. It uses existing facilities i.e. for suppose if we use the ACL tag feature and hit the HBase previous version server it won‟t crash. It just ignores it. Cell ACLs are scoped to the same point in time as the cell itself which means Cell ACL just becomes another piece of the cell and it allows simple and straight forward evolution of security policy over time without any expensive updates. And if we want to revoke access to a user at a specific point of time a new ACL is created and stored back exactly at the coordinates in HBase and it will override the value which is present over there.

3.3.3 Visibility labels (cell labels):

Initially the HBase team at Intel has started adding tags to the cell which can store the meta data for a cell. The detail implementation and support of tags can be seen in HBASE-8496 and HBASE-7897. Then they have made enhancements in such a way that the Access Control List (ACL) is applied on the cell level so that if a user has met the ACL entries then access should be granted to that user. Then the visibility labels are added to the tags which provide security on the cell level.

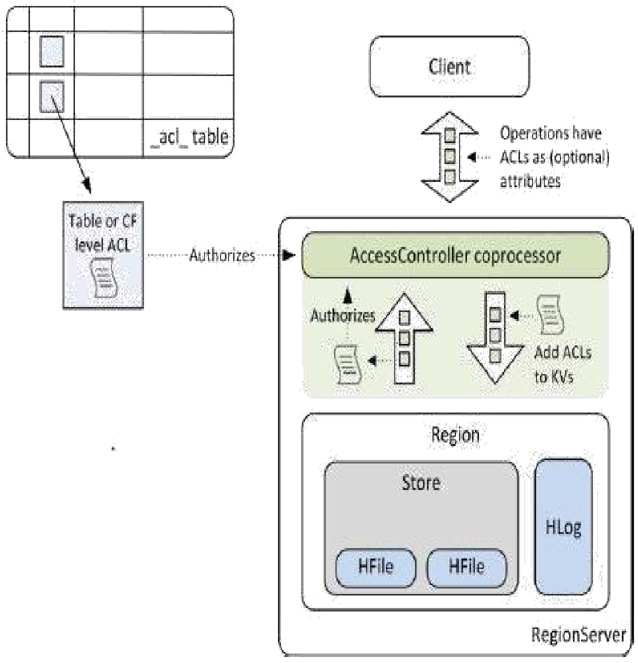


Figure 5 [12]

When we store a cell we can create these visibility expressions which are Boolean and store it along with the cell. These visibility labels are added in the form of label expressions tags and it has a logical expression at the end like &,| and !. In order to create these visibility labels or expressions they have designed a new Visibility Controller coprocessor which verifies the user authorization based upon the meta data label stored in the cell and the label associated with the user. The labels which are associated with the user are handled by some new shell commands `getauths` and `setauths`[5]. The visibility expressions can be added to the cells by using the below command

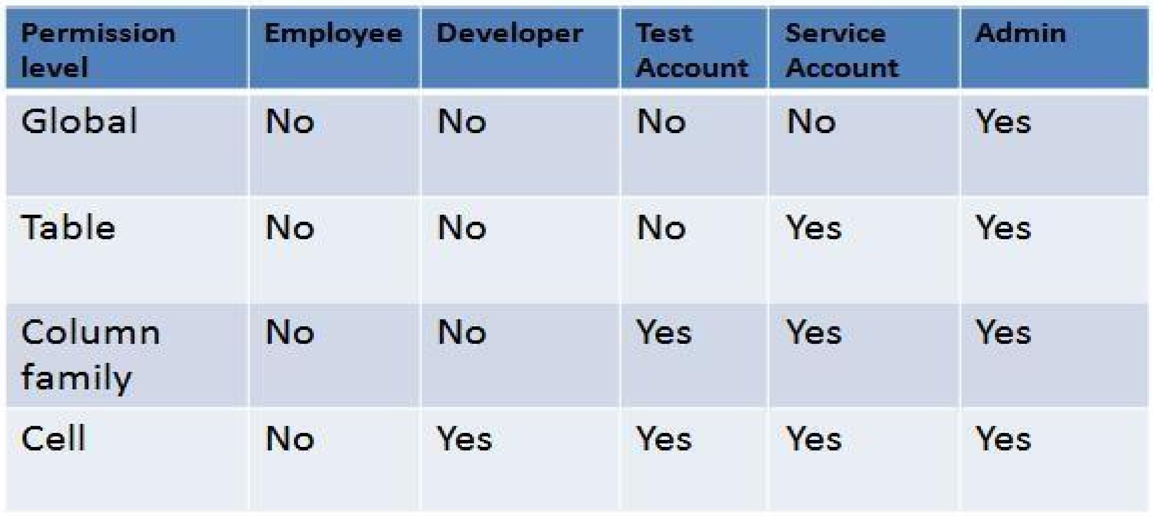
```
Mutation#setCellVisibility(new CellVisibility(String labelExpession));[5]
```

This is done by using an HBase API and it takes care that the older servers does not take this metadata for cells by throwing an error.

As discussed earlier the visibility labels are added with logical expressions and for instance let‟s take a label set which has stored with the cell as SECRET & !PUBLIC. Now any user who has a label which is associated with secret and not public will have access to see that particular cell. Thus any user who has only secret label also will not have any access to the cell since there is nothing mentioned like whether he is private or public. In this way by adding the visibility expressions access can be granted to user on the cell level. They also have a thought of adding some extra plug-ins while building or making extra changes to the label. Also some inputs regarding the encryption have been given towards the server side in order to avoid the leakage of data if there is any. This cell based security model is in experimental phase in HBase and in a year or so it could be implemented on a full fledge.

For instance take an example where we need to define labels corresponding to the roles as per the company security policy which is shown below:

Column Family: X



Permission level	Employee	Developer	Test Account	Service Account	Admin
Global	No	No	No	No	Yes
Table	No	No	No	Yes	Yes
Column family	No	No	Yes	Yes	Yes
Cell	No	Yes	Yes	Yes	Yes

Table 1

In the above table we have defined the labels in such a way that an employee doesn‟t have access to any of the data base tables. Whereas the developer has been given access only to cell level in order to make any changes in that particular cell. Similarly the test and service accounts have been given access to different levels. Only Admin has been gives access to all the levels of the data base to make changes.

Now we can assign one or more roles (or labels) to each user by associating their principle with a label set as shown below:

```
hbase> set_auths 'service', [ 'service' ]

hbase> set_auths 'testuser', [ 'test' ]

hbase> set_auths 'manager', [ 'admin' ]

hbase> set_auths 'dev', [ 'developer' ]

hbase>set_auths‟emp‟,[„employee‟]

hbase> set_auths 'qa', [ 'test', 'developer' ]
```

In the above statements we can observe that a manager should have access to all the levels in the data base and thus he has been associated with the label „admin‟. Similarly QA has been associated with developer and test role in order to access all the column families and all the cells associated with it.

Now based upon the labels defined in the table above we can apply appropriate visibility expressions to cells as shown below:

```
hbase> set_visibility 'user', 'admin|service', \ { COLUMNS => „X' }
```

The above statement says that in user table access has been given to the column family X and its cells to the users who are associated with the roles admin or service. Since we have applied the labels to the user, now if an employee wants to access the column family X in the user table he will not be able to do because the visibility expressions has been set to admin and service roles but not for employee in the user table. Thus, in this way the users can be restricted and can be given access to only a particular cell or a column family based upon his role.

3.3.4 HFile version 3:

In order to reflect these latest changes they have created a new HFile version 3 which stores all the newly written files in it. They have created this new version by making changes to the HBase config xml files. The HFile version 3 has all the new features like the visibility expressions and storing and retrieving of those expressions. It also as the server side encryption features too. Since all the existing files were of version 2, these new features will be present only to the newly written files as they are of version 3. They have planned to push the version 3 file for HBase 0.98 release.

3.4 Running multiple workloads on a single HBase cluster

HBase gives equal priority to all the requests, workloads and this works well with the single workloads on a cluster. However due to this huge scale of data i.e. in petabytes sometimes multiple requests or workloads have to be run on a single cluster. These workloads are planned based upon their access patterns and resources they need. However if there are multiple workloads or requests on a single cluster then this can cause a serious inconsistency. Thus to overcome this multi-tenancy issue HBase team and researchers outside have started looking into some approaches.

3.4.1 Throttle

In order to achieve the consistency in multi-tenant environment we can use insist on some limits and instructions to users so that all the workloads won‟t bother the system at a time. This technique is called Throttling. In general, Throttle is a kind of device (valve) used to adjust the fuel flow and power. Similarly, here we impose some manual rules for specific user and workloads to wait till certain requests i.e. till X requests/sec in order to move the other workloads which needs to address first. For example consider the below statement

Throttle User X to Y req/min

The above statement imposes limit on the requests of User X for a certain time i.e. till Y requests/min. Once this time limit has been met then the requests of User X are allowed to interact with HBase in the backend. The throttle can be changed at the run time too.

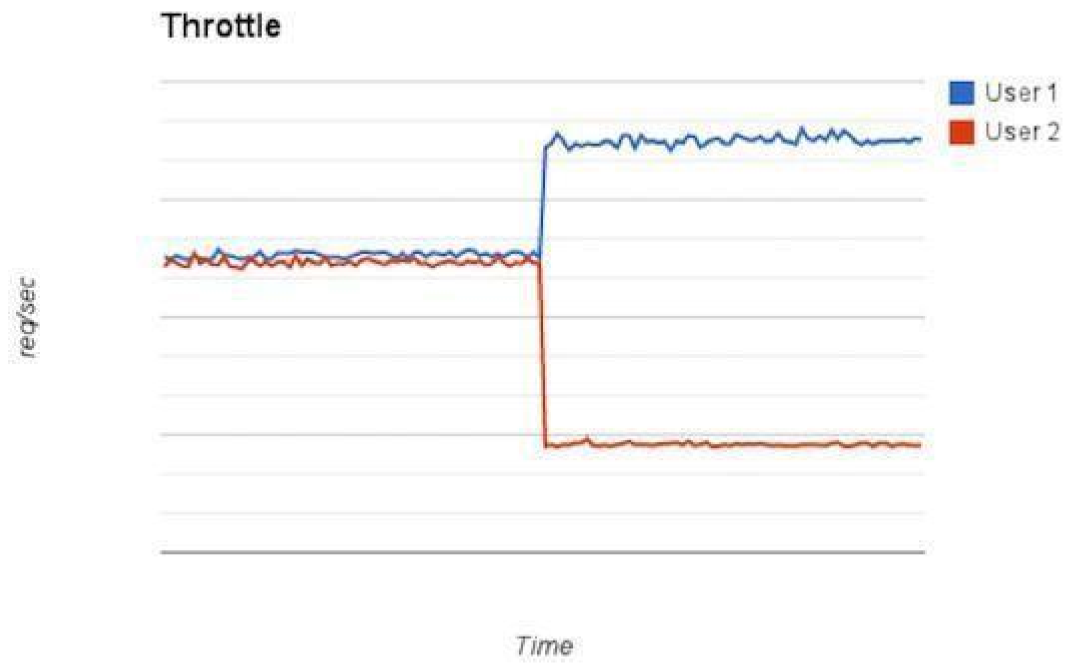


Figure 6 - Throttle results [12]

From the above graph we can observe that user 1 and user 2 have started simultaneously without any priority. Later on once the admin has decided that user 1 should get more preference then he has throttled back the User 2 for „n" req/sec and thus it came down compared to User 1. These throttles can be written in hbase shell and the throttling can be dropped by using "NONE" command in shell. We can also exempt one of the workloads from throttle by using setting the "GLOBAL_BYPASS" to true. Thus we can exempt some of the workloads in production in order to reduce the impact of Throttle.

3.4.2 Decreasing the priority of long running requests

At present there is no technique to guess the estimated time each long request can take. So we can"t separate the long requests from the short requests based on the time estimations. De-prioritizing affects only the scans in the table. Thus if there are any simultaneous events we can delay the scans up to some time by setting up the hbase.ipc.server.queue.max.call.delay property in HBase.

Another approach to de-prioritize the long requests is by using the handlers and queues. For each queue a hander is dedicated to handle the requests. Thus all the scan requests which are long enough can be separated into one single queue and one single handler is dedicated for that. Where as in for all the other short scan requests can be pushed into different queues which are available to accommodate.

Apart from the above techniques there are some other approaches which are still in development phase. One approach is prioritizing the requests as per the user interest. Previously in Throttle technique we used to prioritize the all the requests irrespective of the table and user. But here a user can have his own scheduling policy where he can decide his own priorities for each table and the table can decide the priorities of each workload requests. Another approach is if the admin has an idea about the workload in each table, and then he can distribute the workloads into different machines. Thus based upon the priority the workloads on those machines can be executed. However these techniques are still in early stages of development. For now Throttle technique is widely recommended and it has been included in the latest release of HBase.

3.5 Data Backup and Disaster Recovery

As many data-driven companies started implementing HBase, Data backup and also restoring the huge data (as big as petabytes) after a disaster has become a major concern. There are some mechanisms which can assist with data backup and restoring the same when there is any disaster. Moreover these backup mechanisms should be suitable for the business requirements. An over view has been given on some of these mechanisms.

3.5.1 Snapshots

All the data which is written into HBase is stored into HFiles and log files in HDFS and also into the memory of the region servers. Within HBase we have a feature named snapshot which captures the region which has all the data stored in HFiles. There is no performance issue with this process since it takes place only in seconds. It captures the data by creating an unix kind of link to Hfiles stored in HDFS. The captured data is stored in the form of meta data in Memstore and that block is stored in HFiles itself so that the system can roll back to the snapshot easily within seconds.

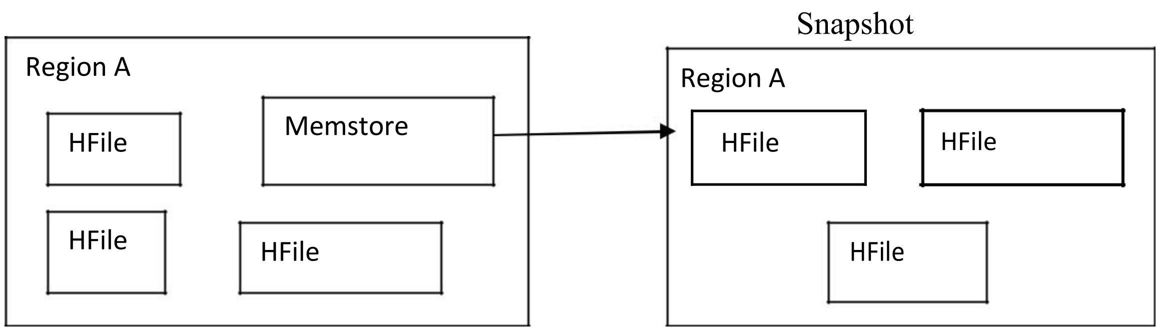


Figure 7 – Snapshot of Region A

In the above figure we can see that the data which is stored in HFiles in Region A is captured and catalogued in the form of meta data in Memstore. This can be used at any time to restore the data from the snap shot provided the system should be offline while restoring.

3.5.2 HBase Replication

HBase is a process where the data is captured all the rows from one data center and it is replicated across all the other data centers. Thus due to any disaster if there is any data center failure then all the users are redirected to the other data center which has all its data. This replication has a minor draw back where sometimes the current edits to a table may not be available across all the other data centers at that point of time. However it is available at a later stage.

3.5.3 HBase Export

Export tool has been provided by HBase to export the data from the tables into the directories in HDFS. To export data it implements map reduce jobs and makes a call to hit the cluster using HBase API and fetches the data from each row in the tables and it writes to the directory in HDFS. This tool is very high in performance. It also provides the convenience of copying the data to anywhere like onto the remote cluster too by setting up a valid remote connection. Similarly to restore the data we have a tool named Import which can capture the data which has previously copied by the Export table.

3.5.4 Copy Table

Copy table is also used to copy the data from the tables. Similar to Export, Copy table also uses Map Reduce jobs and makes a call using the HBase API. However the difference is in Copy Table the data is directly copied into another table which is present in the same cluster so that it can be used as a local to the cluster. There is a performance backdrop with this table since it copies every individual row into the destination file using put requests which could lead to fill up the memstore in destination folders.

Chapter 4

HBase in Real time

4.1 Background

In the above section we have discussed the advantages of the HBase over the traditional RDBMS for large data-driven applications. In real time HBase has been widely used in for various applications like facebook, Twitter, Adobe, Yahoo etc. All these applications handle numerous data read/write requests. In this section we shall present in detail about one of these applications i.e. Facebook and why HBase has been used for it.

4.2 Workload challenges

In 2009, Facebook has decided to build its messenger app which is a single system which supports multiple messages from different platforms like email, text messages, Facebook account messages etc. In order to store all these different modes of communication it should also maintain a single archive and to handle such large data Facebook needed new database infrastructure. It was the largest engineering project which Facebook has handled till then and it had a team of around 15 engineers. Till then Hadoop was using MySQL for user data, the Cassandra data base for the inbox search and the Haystack for photos. They have been using Hadoop in combination with Hive for analytics and data storage. However for some of the latest applications those have been arisen at Facebook required very high write throughputs, low latency and random performance to read. In terms of random read performance, MySQL storage engines were performing well but they were not performing well in terms of the latency and random write outputs. It was a difficult task for the MySQL engines to scale up in order to balance the load and time efficiency. MySQL engines were also seemed to be an overhead for the organization since the hardware was expensive.

Apart from the performance issues, there were some work load challenges and requirements which engineers at Facebook had to think while developing the messenger application. They were as follows:

- Undoubtedly the moment this application has gone live in production around hundreds of million people will be using it and large amounts of data which might scale up to many petabytes should be handled with rigid uptime. Thus Elasticity plays a key role and the capacity of the storage systems should be increased enormously in order to balance the load and utilization.
- The primary goal of the messenger app was to persist all the messages from multiple platforms like email, text messages and Facebook chat messages under one tree. In order to store these messages for each user separately a new threading modal has to be designed.
- The other major challenge for the engineers is the migration of the data where the existing user messages have to be moved into the new data modal. For the quick migration random access and ability to do large scans will be needy.
- There will be a lot of messages in each user"s mail box and these messages should not be deleted unless done by the user. Thus the messages will be increasing daily and large set of database tables should be maintained with a schema that is indexed by the users with their messages.

- With these workloads as the number of rows increases, write performance will be decreased deliberately for storage engines like MySQL and it also results in random I/O operations.
- There will be a high write throughput as there will be millions of messages exchanged between the users every day and also the messages of each user will be stored in an archive for reference. If a user wants to read those messages there might be a latency issue while retrieving large number of messages from the database with the engines like MySQL.
- The other concern was the recovery from the disaster. The application should be in a stable way so that it can provide efficient service with high uptime to the users in all conditions like software or hardware failures and upgrades. The service should also be in a position to manage the loss of data from a data center and to serve it from other data centers within a short period of time. The developers were also thinking to stabilize the service in such a way that if any individual center gets failed (though it happens very rarely) there should be no downtime.

Thus after going through all these challenges and requirements, engineers at Facebook decided to go with the HBase and Hadoop as the technologies which can lay strong foundation of storage for these kinds of huge data-driven applications. The decision was made based upon the performance of the HBase which has high write-throughput and had consistent key-value storage. They have believed it and they were confident that it can address all the challenges which were not solved till then. Though they had some draw backs with the HDFS Name Node failures, they were confident to overcome this by developing a well-built name node whoch would be helpful for their in-house operations too. Storage of data in large HBase/HDFS clusters also lead to the failure of the system and it was not achieving the objective of fault isolation. To overcome this they have come up with an idea of storing the data in small clusters. By that time there was an HBase community which had lot of forums and blogs to discuss and helped lot of developers who want to implement HBase in their applications. Thus the engineers at Facebook were confident in achieving the goal.

4.3 Implementation of HDFS

HDFS is a file system which can store large files which has massive amounts of data which can scale up to terabytes. It has been designed in a specific pattern such as write-once, read-many-times pattern. However as we discussed earlier, there were some issues with the Name node failure. Name node acts as a master node inn HDFS and it is the one and only node which acts as a master. Thus whenever the name node is failed, the whole HDFS cluster will be down until the name node got recovered. This draw back bothered a lot of engineers in the market who were dependent on the HDFS particularly for the applications whose uptime should be all day. The engineers at Facebook too had this issue but it was only once since they have started implementation of the HDFS. Since their hardware was reliable in most cases and software had a good end to end testing before the deployment into production they have faced only once where the Name node has crashed. However they have made some enhancements to the HDFS architecture in order to overcome this single point of name node failure.

4.3.1 Name Node (Master) and Data nodes

In HDFS cluster there are two kinds of nodes which function in form of master-worker where the name node is the master and the data nodes are acted as worker nodes. It manages the filesystem namespace. It has all the file names and the meta data of all the files and directory in

HDFS. This information is stored insistently in name space file and all the transaction logs in the edit log. Name node also has an idea of all the data nodes on which the blocks of a file are location, though it does not store the locations since the information about the locations is restored from the data nodes when the system has started. Data nodes acts as the worker nodes where when a request has been made by the client via name node then they store or retrieve the blocks of a file based upon the requirement and they update about the blocks they have stored to the name node. Thus for a cold-start of NameNode has to deal with two things. First one is to read the name space file (also known as file system image) and applying the transaction logs and then it saves the new file system image to the disk. Second one is to processing of all information of the blocks which they get from the data nodes so that they can recover all the block locations present in the cluster. Thus the Biggest HDFS cluster they have has about 150 million files and above mentioned two stages take equal time to complete their tasks.[3] On an average for a cold-start it takes about 45 to 50 minutes.

As discussed earlier there was a growing concern on what if the master node (NameNode) gets failed. To overcome this Hadoop has provided a secondary name node which acts as a kind of backup name node. The secondary name node usually runs on the separate machine since it requires lot of memory and CPU as the same the NameNode has. The backup node avoids reading the name space file (file system image) from the disk again if the name node got failed but still it has to process all the block location of the files which come from the data nodes. Thus it takes around 20 minutes which is not at all good for an application which needs a 24/7 uptime. Thus the engineers at Facebook were not happy with it as the recovery time was high. To overcome this they came up with a new node which is called as Avatar Node.

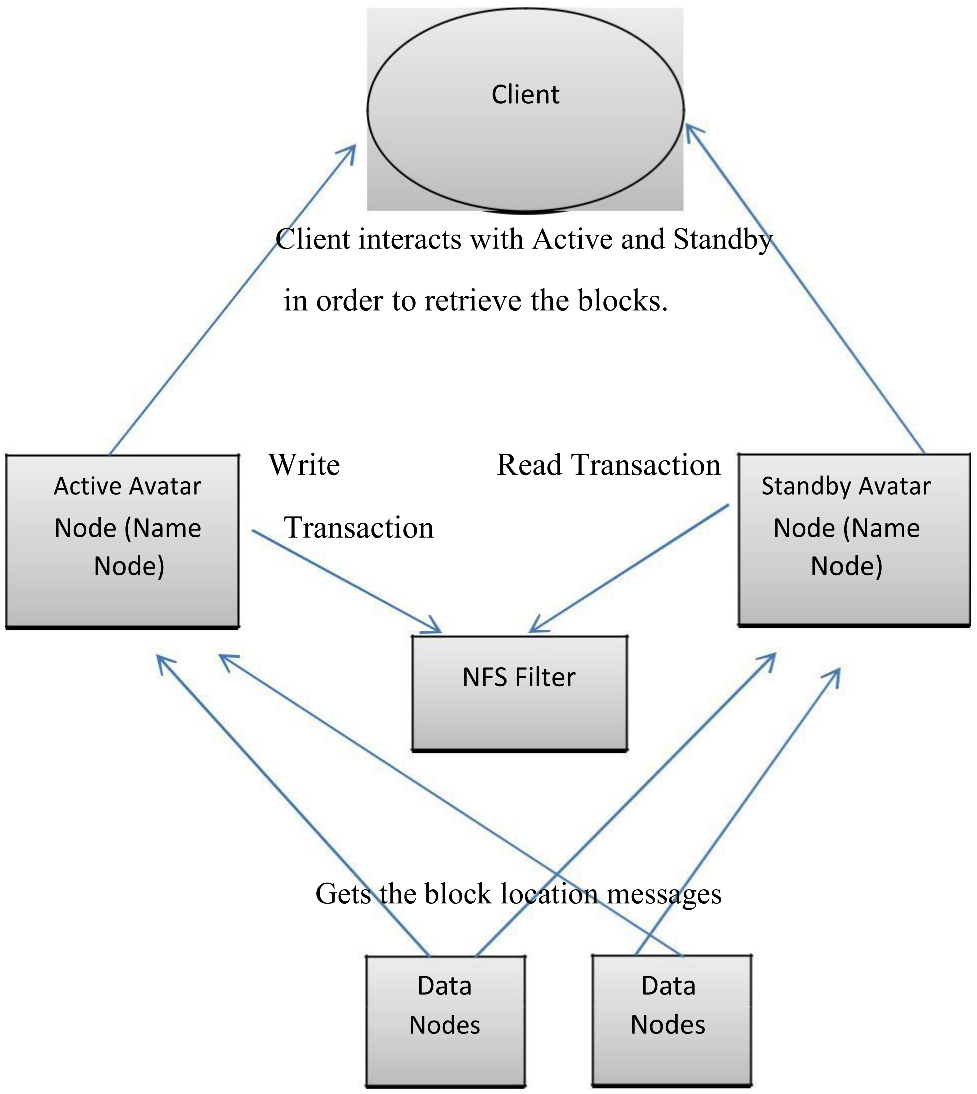

Client
Client interacts with Active and Standby in order to retrieve the blocks.
Active Avatar Node (Name Node)
Write Transaction
Read Transaction
Standby Avatar Node (Name Node)
NFS Filter
Gets the block location messages
Data Nodes
Data Nodes

Figure 8 – HDFS model [3]

4.3.2 Avatar Node

They came up with the new architecture where the HDFS cluster will have two nodes. One is the Active Avatar Node and the other one is the Standby Avatar Node. Previously the HDFS cluster used to have only one Name Node. In this case the Avatar Node acts as a wrapper. Moreover it acts like a wrapper around the Name Node. At Facebook all HDFS clusters use Network File system in order to store the copy of the fsimage and another copy of transaction logs (also known as edit logs). As we discussed earlier, the Name Node (Avatar Node in this case) writes all the transactions to the transaction log which is stored in the NFS file system. Then the Stand by Avatar Node reads all these transaction logs from the NFS file system and it starts storing these transaction logs into a separate name space. In this way it maintains a separate namespace which is almost similar to the actual name space which is maintained by the Active Avatar Node. It also takes care of check pointing the Active Avatar Name Node and creates a new fsimage so that the stand by name node no more acts as the secondary NameNode. Since it has all the information which the Active Avatar Node (Primary Name Node) had it takes very less time (in seconds) to recover if the Active Avatar Node got failed.

Previously the data node used to interact only with the Name Node if there were any request from the clients. But in this case it interacts with both Active Avatar and Stand by Avatar as well. Thus the Stand by will have all the recent information about the blocks of the files so that it can serve the application if the Active Node got failed by any instance.

4.3.3 HDFS Transaction logging

Usually HDFS accounts the newly allocated blocks with its id‟s to the transaction log only when the file is closed or flushed. Since the engineers at Facebook doesn‟t want much failover, they have written new transactions to the edit logs (transaction logs) about the block allocation so that the Standby can get to know about the newly allocated blocks. Thus it allows client to continue writing to the files from where it has stopped before the failover. Moreover they have formatted the edit logs to have transaction length, id and checksum so that the stand by Node can read the whole transaction.

4.3.4 Distributed Avatar File System

Engineers at Facebook were putting in lot of efforts to overcome the failover events and to provide better efficiency. They have developed a Distributed avatar File system(DAFS) which acts as a layer on the top of the client which provides clear access to the HDFS in the case of fail over. It has been integrated with the Zookeeper distribution service which holds a zNode which has the physical address of the Avatar Node. Thus when a request has been made by the client, DAFS file system checks with the respective zNode for the Avatar Node and it directs all its calls/requests to it. If a request has went through any network error, then the DAFS requests the Zookeeper for a change of primary node. Thus they have designed it in such a way that the zNode will have the physical address of the new Avatar Node and the DAFS will make its all requests/calls to this new Avatar Node. Thus they have made the fail over event more transparent for an application.

4.3.5 Enhancements for Real time Work Load

HDFS usually has high throughput time but he response time was very low. In case of any errors or failures, the response time was very high and this was a major challenge for HDFS. To overcome this issue we have come up with some enhancements:

Remote Procedure Call (RPC) Timeout: To send RPC"s Hadoop generally uses tcp connections. Whenever there is a tcp-socket timeout, instead of declaring a RPC time out the client sends an alert to the RPC server and the client will wait for the response until the server is dead. However this wait has sometimes been a continuous one and it is not a good thing for a real-time application. Hence they have come up with a different strategy. Thus when client makes a RPC to Data Node and if it gets failed instead of waiting for it it"s better to fail fast and try for a different Data Node. Thus they have added an extra feature of specifying an RPC time out whenever a server has started with an RPC session.

Recover Lease: Usually HDFS supports only a single writer for a file and a lease will be handled by the NameNode to get adjusted with the semantic. There are many instances where an application fails to open a file which was not closed properly earlier. In order to succeed in this aspect HDFS-append has been called repeatedly on the log file. The append usually trigger‟s a soft lease to expire. Thus in order to withdraw the log file‟s lease, the application has to wait for the soft lease period which is almost for a minute by default. Thus it takes a lot of time and also an additional cost is imposed because of the HDFS-append operation which usually takes times for a write pipeline establishment and it will be more in the case of error and it includes more than one Data Node for it. Thus to avoid this they came with an API called "Recover Lease" which revoke‟s a file lease clearly. When the Name Node receives the recover lease request it starts the recovery process and before that it will change the lease holder of the file to itself. It returns the status update to the application if it was success or not so that it can attempt to read from that file.

Local Replicas: At some instances in order to improve the performance and efficiency application stores data in the HDFS. However for reading and writing latency has been a major concern in HDFS. If the reading and writing has been done to a local machine then the latency was pretty less. To overcome this latency issue they have made a major enhancement to HDFS client such that it identifies that there is a local replica of data and it uses that local replica for reading instead of involving the Data Node. In this way they have enhanced the performance.

4.4 Implementation of HBase at Facebook

Developers at facebook had made some enhancements to HBase as per their requirements of the application in order to achieve the performance efficiency and durability. Some of their enhancements have been discussed briefly below:

4.4.1 ACID Properties:

As discussed earlier, HBase does not obey some of the ACID properties. However the engineers at Facebook were expecting some of the ACID properties to be followed by their database systems. The engineers had a strong belief in the consistency of HBase and moreover its MVCC (like the Read Write Consistency Control (RWCC)) architecture which has provided strong isolation and the HLog which has provided sufficient stability. However they still decided to make some changes so that the HBase followed to some of the ACID properties like the Atomicity, Consistency etc. which they needed.

Atomicity is capability of the database to ensure that in a transaction either all the tasks should be performed or none of them should be performed. Thus a transaction is said to be atomic if it follow the above rule. In HBase achieving the row level atomicity was the primary goal. Though the RWCC structure had given some guarantee, this was not possible due to the Node failure. As discussed earlier in HBase the Region servers are responsible for the read/write requests from the client. Thus when there are numerous entries into a single row, while writing these transactions to the HLog if the region server got failed then only partial writings will be made into the transaction. Thus either the complete transaction will not be failed or performed successfully. Thus it did not obey the Atomicity property. To overcome this they have come up with the new concept of log transaction which is known as WALEdit where each transaction will be fully completed or not at all written.

Consistency guarantees that only valid data which obeys all the rules and properties is written into the data base. When a particular transaction fails which results in an invalid data to be entered into the database, then it stops the invalid data and reverts back to the previous state in order to follow the rules. As discussed earlier, in order to achieve performance and low latency developers had made some enhancements to HDFS in order to identify the local replica of data and use that instead of involving the data node. In the similar way, HDFS has provided some replication to HBase in order to handle the consistency that HBase has guaranteed. While writing the transactions, HDFS has set up a pipeline kind of connection in order to ensure that all the replicas should send an ACK once they get the data. In this way when HBase does not get an ACK or an ACK of data not being sent properly then it does not continue further. Since all the write transaction logs are present in HLog, when HBase doesn"t get any positive ACK, it will roll back the HLog and obtains new blocks. In this way it achieves consistency.

4.4.2 Availability Enhancements

Online Upgrades: One of the concerns was the cluster downtime. The downtime was too high and one of the major reasons was due to the system maintenance. Initially they have identified that Regionservers which are responsible to serve regions were taking minutes of time to stop when a stop request has been issued to them. This is due to the long compaction (compressing unused data) cycles. To overcome this, instead of these long compact cycles they have made them interruptible so that they can favor region server downtime. This move has reduced the downtime to seconds which indeed reduced the shutdown time of the cluster. Another concern was with the complete Start and Stop of the cluster by HBase for software upgrades. Instead of full start and stop, they have added some rolling restart scripts to perform the upgrades for one server at a time. When a region server has been stopped, the master node again assigns regions to a region server automatically and thus it reduces the downtime. Though many defects were raised due to this rolling restart scripts which related to reassignment and region off lining, they have overcome by rewriting the Master which integrated with Zookeeper.

Log Splits:

When a region server dies, usually the master node would split all its Hlogs and replayed them before the region can be reopened for read and writes. Since there were HLogs per server this was a very time consuming process. To overcome this they have started utilizing zookeeper for splits across the region servers where the master organizes a distributed log split.

4.4.3 Performance Improvements

In order to enhance the data insertion in HBase they have started focusing on sequential writes. As discussed earlier, a data transaction is applied to a MemStore which is an in-memory cache. When this MemStore reaches a maximum point then then the transaction data is copied into an existing HFile where these files are immutable and sorted based upon the key/value pairs contained in it. However sometimes instead of writing into an existing HFile they are written into new HFiles. A large number of read requests will be issued on these HFiles for results. Thus in order to improve the read performance these HFiles have to be compacted (or compressing) these HFiles by removing the unwanted data which has been created during updates.

Implementation of Compaction: Compaction is a process where we compress the unused data in the database files which has been created during the updates in order to improve the efficiency. Though the older versions are removed a metadata of these files is kept to use in case of any conflict during replication.

Usually the read performance is based upon the amount of files present. This can be managed well with the help of a proper compaction algorithm. If the compaction algorithm is not written well, it might affect the network efficiency too. Thus the developers at Facebook had put in lot of efforts in order to write an efficient compaction algorithm. Previously Data Compactions was done in two different ways like Minor or major. Minor compactions were done for the files which were smaller in magnitude but they do not delete process or get rid of the unused data. This has resulted in HFiles with larger magnitude than necessary. However in the major compaction used to delete the unused data and it used to compact all the major HFiles unconditionally. Thus due to this draw back in the minor compaction iit had an impact on the cache efficiency of the blocks and it had an impact on further compactions too. Thus they have done some modifications by merging the two different approaches into one single code path to make the files small in magnitude which resulted in getting rid of the unwanted data. Once this problem got resolved the next concern was the latency. Due to the existing compaction algorithm the latency was around 25 milliseconds which seemed to be high for the developers at Facebook. This was happening due to the existing algorithm where the minor compaction was happening unconditionally for the first 4 HFiles provided a prompt has been reached for the minor compaction of the first three HFiles. Thus they have come up with a modification to the algorithm by stopping the unconditional compaction for files which are above the certain size and skip it completely if good enough files were not found. Thus, in this way they have dropped the latency.

Improvisation of read performance: Read performances can be improved by keeping the files very less in number with the help of compaction which results in reduced random I/O operations too. Moreover developers at Facebook had an idea of skipping particular files for some queries so that the I/O operations can be reduced in order to improve the read performance. They have used Bloom Filters for this mechanism. They provide an efficient time-constant method to query if a particular row or a column is present in HFile. Bloom filters are added to the end of the HFile and through the process of folding the filters are cached in memory. Thus when we query for a particular row or column it first looks in the cache bloom filter for the HFile and in this way it skips some of the files.

4.5 Production Deployment and challenges

In this section we are going to discuss briefly about some of the operational experiences they have faced during the testing and in production.

4.5.1 Testing

While designing the HBase as per their needs, they were primarily concerned about the durability of the open source code of the HBase and also ensure the same if there are any future changes. In order to fulfill this, they wrote an HBase testing program which generates data which can be written into HBase. Thus the tester writes all this generated data into HBase and also read the data in order to verify if the data has been added or not. They have also selected and killed some of the processes in the cluster and verified the returned database transactions were successfully written or not. This helped them to test a lot of issues.

4.5.2 Region Splitting (Manual v/s Automatic)

HBase offers a feature called automatic splitting where when the region size is increased then the region is automatically split into 2 regions. However at Facebook they have developed and implemented the manual splitting instead of the automatic one. While creating the table, they manually presplit the table into certain regions of equal size. When the size of the region got increased, they start rolling splits in these regions. The main reason to opt for manual splitting is the data grows uniformly across all regions. With the auto splitting there might be a huge flow of split and compactions on all the region servers at same time as the size of the data increases uniformly in all regions at same time. Whereas with the manual splitting, they can control the splits across the time thereby controlling the network IO load across the regions which results in decreasing the production load too. Thus as per their experience, they came to a conclusion that automatic splitting is not appropriate for the applications which has uniform distribution of data.

4.5.3 Schema changes

One of the major drawbacks of HBase is it does not support online schema changes to an existing table. If we want to add a new column family to an existing table we can"t do it right away. For this we need to stop the access and disable the table, then add the new column families and again give access to table for operations. In order to perform this operation, they have to stop the workload and they were not in a position to do that. Thus in order to overcome this draw back they have created and added some column-families to the table before going live itself so that they can be used in future if there is any need.

4.5.4 Data Imports

Efficiently importing the existing legacy message data into the HBase without any latency was one more concern. They have done this job by issuing data base put requests from Hadoop Job. Since these put requests were sent across the servers it would saturate the IO of the network and it has also created severe latency issues in the initial release. The developers were not happy with this kind of latency since it might show severe impact on production workload. Thus they came up with a bulk import method where it partitions large amount of data into several regions using the map job and then it used to compress the map job output by GZIP compressing and the reducer job writes data directly to compressed HFiles.

4.5.5 Network IO traffic

Once the HBase was launched live in production, after a couple of months they came to know that the Network IO traffic was increased. They have used some statistics tools and log scraping to estimate the network IO on one region server for a 24 hour period. They came to know that due to the major compactions and the minor compactions the network IO traffic was increasing. The traffic across these two compactions was around 85%. They have increased the compaction interval from daily basis to weekly basis in order to control the network IO traffic and they were successful. They also got benefitted by excluding some of the column families from being logged to the HLog.

4.6 Future implementations at Facebook

HBase is still in early stages at Facebook and it was implemented only in messenger app initially. Since it got successful in production they were plans to implement HBase in other areas as well at Facebook. Some of the use cases were Operational Data Storage, Puma(Real time analytics), search indexing etc. They were also focusing on some of the areas of HBase where improvisation is need like for load balancing in the regions, capacity to recover quickly to network partitions, rolling restarts for software upgrades etc. Also there has been some research going on the cross data center replication since these applications serve the same data actively across different data centers.

Chapter 5

HBase vs RDBMS

5.1 Background

Since long time "HBase vs RDBMS" has been a debatable topic for many of the enthusiasts who are working in this area. In this paper I just tried to give a glance on both the data bases and its implementations, performance for similar kind of issues.

As mentioned earlier, HBase is an open source data base with a distributed column-oriented built on the top of HDFS. It is mainly used for large data sets to give read/write access in real time applications. It has been mainly designed in such a way that it can handle scalability issues. It can hold very large tables like billions of rows X millions of columns and it can be horizontally segregated and replicated across thousands of product nodes habitually. Various data structure serializations, storage and retrieval of the data have been showcased by the table schemas in HBase. Whereas RDBMS is based upon the relational model of data and it follows the Codd"s 12 rules which were proposed by E.F. Codd in 1970. It categorizes data into multiple tables which has rows and columns and each row has a unique key so that querying can be done using this primary key. The data base is totally concerned on rows and it strictly follows the Acid properties. SQL language has been used for writing the queries to implement the CURD operations. In terms of flexibility and features RDBMS is the perfect choice for small and medium scale applications. It has efficient open source features like the PostgreSQL and MySQL. However coming to the larger data base applications in terms of data, random access and read/write operations RDBMS couldn"t fulfill its requirements. There was a lot of performance deficiency and it eventually lead to the breaking of the ACID properties, Codd‟s 12 and some traditional data base properties for which the RDBMS was famous for.

5.2 Analysis

To have a better understanding a comparison let us compare both the databases in terms of some features and characteristics:

Let us assume there is a service launched and we are using RDBMS for it. For launching of this public service, will be hosted on a MySQL instance which has a well-defined schema and operated remotely. As service starts getting widespread there will be a lot of reads hitting the data base and we need to memcache (It is a distributed memory caching system used to enhance the performance of the websites by caching the data so that maximum number of hits to an external API) the queries. As there is lot of data to be cached and there are lot of hits coming in durability of the transaction is tough to maintain with the vertical distribution. Thus if it does not obey the durability and consistency then there is no point of following ACID properties for which the RDBMS is famous for. As per the RDBMS characteristics the scaling is done in a vertical approach. Thus if there are too many hits to the database we need to increase the power of CPU and RAM which is an expensive thing. It is also not a good option to handle too much of data on one server just by increasing its power. If there is any server crash due to the data load then that‟s not good as the down time is not acceptable in customer point of view. Thus to handle these sort of issues it‟s better to go for multiple nodes which is no longer a vertical approach in terms of scaling. In some instances there could be a chance of writes getting slower and in order to avoid this we need to drop the secondary indexes and go for the primary key look ups which no longer makes this RDBMS. Since we have larger hits we need to denormalize the data in order to enhance the performance of the data base and to reduce the complexity of querying the joints. Thus there are lots of drawbacks in handling the large data on a single node with the vertical approach.

Now let us assume launching a service which has large data to be read and write on a NoSQL data base like HBase. To launch the service, it will be hosted on a shared, remote NoSQL instance. Here the large data is stored more sequentially into rows and columns using the primary indexes. As the data is increased it will be split across all the multiple nodes or regions which are available. So there is dependency on a single node or a server. The scaling is done as a process of automation. Thus when we add a node to the cluster which we point to and then run the region server so that it automatically balances the load on all the regions. In RDBMS we used to increase the power of CPU and RAM in order to distribute the data which was a costly affair. But here in HBase data is distributed on multiple nodes which are a hardware which we can get relatively for a lower price in the market. As the data is distributed on multiple nodes there is nothing to worry if one of the nodes got crashed. Thus for more efficiency and to avoid the tension of server crash or downtime it"s better to go for HBase in order to handle large sets of data in terms of scalability, efficiency and speed.

Chapter 6

Future work

Most researches in HBase community are going on the data security and access levels as it is a sensitive issue which can cost a huge loss for business and companies like Intel, Yahoo, Facebook etc are also trying to make valuable contributions towards it. Every year in May all the developers, researchers, speakers, business leaders who are a part of HBase committee will meet at the HBase conference. Multiple presentations and discussions on current big data issues, new use cases and features will take place. Also the backdrop of the schema design where the column-families have to be pre-defined was one more concern and the members of HBase are working towards a possible fix. This year the HBASECON 2015 is going to take place at California where all the people who are connected with HBase will meet and discuss the progress and contributions towards HBase. We can listen some user experiences of running HBase in production. I assume HBase is going to play a key role in terms of big data and analytics in coming days and it has been used in various fields like finance, bio informatics etc. Also many top data-driven companies like twitter, Facebook, Mozilla, and Adobe etc have already started using it.