Session: 13 Introduction to HBase- Overview of HBase

HBase is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (HDFS). HBase provides a fault-tolerant way of storing sparse data sets, which are common in many big data use cases. It is well suited for real-time data processing or random read/write access to large volumes of data.

HBase does not support a structured query language like SQL. An HBase system is designed to scale linearly. It comprises a set of standard tables with rows and columns.

HBase works well with Hive, a query engine for batch processing of big data, to enable fault-tolerant big data applications. **HBase created by Apache Software Foundation.** The Initial release 28 March 2008. HBase applications are written in Java.

Why is HBase fast?

HBase is a column-oriented, non-relational database. This means that data is stored in individual columns, and indexed by a unique row key. This architecture allows for rapid retrieval of individual rows and columns and efficient scans over individual columns within a table. Cells are the smallest units of HBase tables, holding the data in the form of tuples.

Why ZooKeeper is used in HBase?

ZooKeeper acts as the bridge across the communication of the HBase architecture. It is responsible for keeping track of all the Region Servers and the regions that are within them. Monitoring which Region Servers and HMaster are active and which have failed is also a part of ZooKeeper's duties.

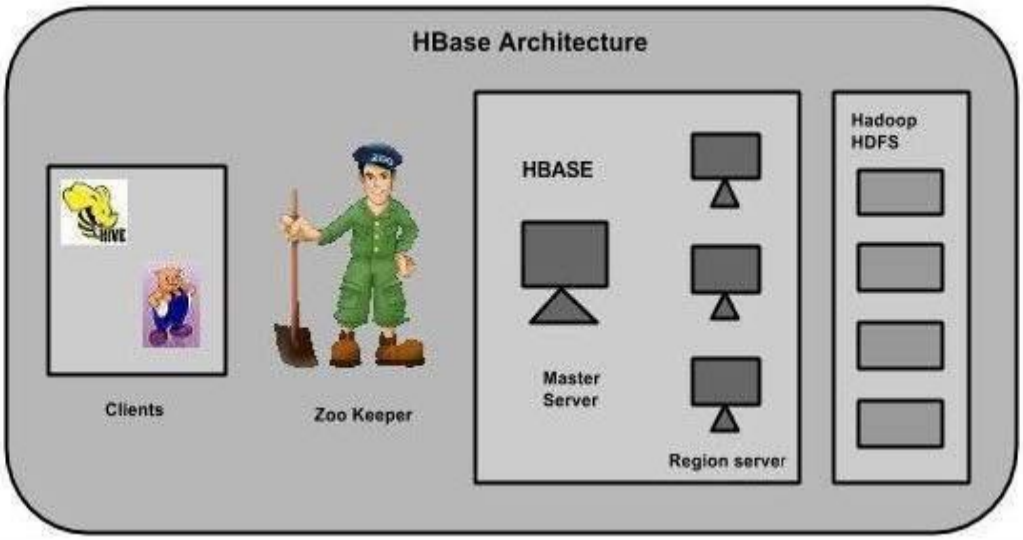
HBase has a number of features like:

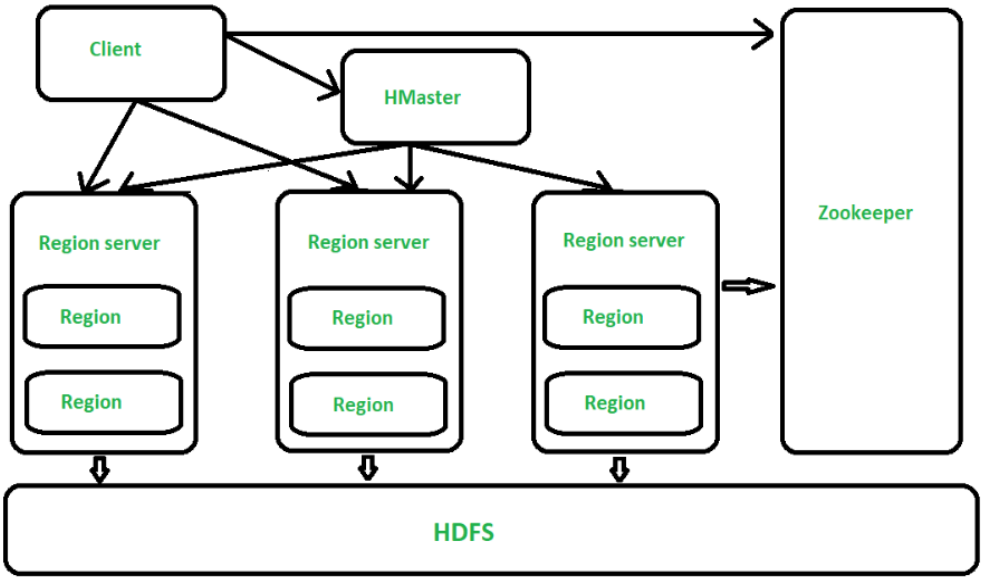
- **Scalable:** HBase allows data to be scaled across various nodes as it is stored in HDFS.
- **Automatic failure support:** Write ahead Log across clusters are present that provides automatic support against failure.
- **Consistent read and write**: HBase provides consistent read and write of data.

1. **HBase architecture**

   Tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.

   HBase architecture has 3 main components: **A.HMaster B. Region Server C.Zookeeper.**





1. HMaster –

The implementation of Master Server in HBase is HMaster. It is a process in which regions are assigned to region server as well as DDL (create, delete table) operations. It monitor all Region Server instances present in the cluster. In a distributed environment, Master runs several background threads. HMaster has many features like controlling load balancing, failover etc.

2.Region Server –

Regions are nothing but tables that are split up and spread across the region servers. HBase Tables are divided horizontally by row key range into Regions. Regions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families.

Region Server runs on HDFS DataNode which is present in Hadoop cluster. Regions of Region Server are responsible for several things, like handling, managing, executing as well as reads and writes HBase operations on that set of regions. The default size of a region is 256 MB.

3. Zookeeper –

It is like a coordinator in HBase. It provides services like maintaining configuration information, naming, providing distributed synchronization, server failure notification etc. Clients communicate with region servers via zookeeper.

Advantages of HBase –

- Can store large data sets
- Database can be shared
- Cost-effective from gigabytes to petabytes
- High availability through failover and replication

Disadvantages of HBase –

- No support SQL structure
- No transaction support
- Sorted only on key
- Memory issues on the cluster

Features of HBase architecture:

A.Distributed and Scalable: HBase is designed to be distributed and scalable, which means it can handle large datasets and can scale out horizontally by adding more nodes to the cluster.

B**.Column-oriented Storage:** HBase stores data in a column-oriented manner, which means data is organized by columns rather than rows. This allows for efficient data retrieval and aggregation.

C.Hadoop Integration: HBase is built on top of Hadoop, which means it can leverage Hadoop's distributed file system (HDFS) for storage and MapReduce for data processing.

Consistency and Replication: HBase provides strong consistency guarantees for read and write operations, and supports replication of data across multiple nodes for fault tolerance.

Built-in Caching: HBase has a built-in caching mechanism that can cache frequently accessed data in memory, which can improve query performance.

Compression: HBase supports compression of data, which can reduce storage requirements and improve query performance.

Flexible Schema: HBase supports flexible schemas, which means the schema can be updated on the fly without requiring a database schema migration.

HBase is extensively used for online analytical operations, like in banking applications such as real-time data updates in ATM machines, HBase can be used.

2. **Installation**

Session-14-15-The HBaseAdmin and HBase Security

o Various Operations on Tables

There are various commands that we can use for data manipulation on the HBase table:

o HBase general command and shell

General Commands

- status - Provides the status of HBase, for example, the number of servers.
- version - Provides the version of HBase being used.
- table_help - Provides help for table-reference commands.
- whoami - Provides information about the user.

These are the commands that operate on the tables in HBase.

- create - Creates a table.
- list - Lists all the tables in HBase.
- disable - Disables a table.
- is_disabled - Verifies whether a table is disabled.
- enable - Enables a table.
- is_enabled - Verifies whether a table is enabled.
- describe - Provides the description of a table.
- alter - Alters a table.
- exists - Verifies whether a table exists.
- drop - Drops a table from HBase.

- drop_all - Drops the tables matching the 'regex' given in the command.

Java Admin API - Prior to all the above commands, Java provides an Admin API to achieve DDL functionalities through programming.

Under **org.apache.hadoop.hbase.client** package, HBaseAdmin and HTableDescriptor are the two important classes in this package that provide DDL functionalities.

Data Manipulation Language

- **put** - Puts a cell value at a specified column in a specified row in a particular table.
- **get** - Fetches the contents of row or a cell.
- **delete** - Deletes a cell value in a table.
- **deleteall** - Deletes all the cells in a given row.
- **scan** - Scans and returns the table data.
- **count** - Counts and returns the number of rows in a table.
- **truncate** - Disables, drops, and recreates a specified table.

Java client API - Prior to all the above commands, Java provides a client API to achieve DML functionalities, CRUD (Create Retrieve Update Delete) operations and more through programming, under org.apache.hadoop.hbase.client package.

HTable Put and Get are the important classes in this package.

java client API for HBase

To perform CRUD operations on HBase tables we use Java client API for HBase. Since HBase has a Java Native API and it is written in Java thus it offers programmatic access to DML (Data Manipulation Language).

i. Class HBase Configuration

This class adds HBase configuration files to a Configuration. It belongs to the **org.apache.hadoop.hbase package.**

ii. Method

static org.apache.hadoop.conf.Configuration create() to create a table in HBase.

o Admin API

HBase is written in java. it provides java API to communicate with HBase. Java API is the fastest way to communicate with HBase.

Class HBaseAdmin:HBaseAdmin is a class representing the Admin.

This class belongs to the org.apache.hadoop.hbase.client package.You can get the instance of Admin using **Connection.getAdmin() method.**

Methods and Description

S.No.	Methods and Description
1	**void createTable(HTableDescriptor desc):** To Create a new table.
2	**void deleteColumn(byte[] tableName, String columnName) :**Deletes a column from a table.
3	**void deleteColumn(String tableName, String columnName):** Delete a column from a table.
4	**void deleteTable(String tableName)** -Deletes a table.

5. **HTableDescriptor addFamily(HColumnDescriptor family)**

Adds a column family to the given descriptor

6. **HTableDescriptor(TableName name)**

Constructs a table descriptor specifying a TableName object.

CRUD operations: Working with HBase Client API

CRUD stands for create, read, update, and delete.

1 Put Method

To update any record HBase uses 'put' command. To update any column value, users need to put new values and HBase will automatically update the new record with the latest timestamp.

put1.addColumn(Bytes.toBytes("education"), Bytes.toBytes("Type"),Bytes.toBytes("Regular"));

2.Read= scan & get

3. delete

o Client API

by using the API code, we can perform the operation on HBase table. To create the table through Api we use the following code-

```
package createTable;

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.TableName;
import org.apache.hadoop.hbase.client.*;
import java.io.IOException;
public class createTable
{
	public static void main(String [] args) throws IOException{
		Configuration conf = HBaseConfiguration.create();
		Connection con = ConnectionFactory.createConnection(conf);
		Admin ad = con.getAdmin();
		HTableDescriptor ht = new HTableDescriptor(TableName.valueOf("emp1"));
		ht.addFamily(new HColumnDescriptor("education"));
		ht.addFamily(new HColumnDescriptor("post"));
	if(!ad.tableExists(ht.getTableName()))
	{
		System.out.println("Creating table");
		ad.createTable(ht);
		System.out.println("Done");
	}
	}
}
```

CRUD Operation:

```
package createTable;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Table;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.Delete;

public class crud {
public static void main(String[] args) throws IOException {
Configuration conf = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(conf);
Table table = connection.getTable(TableName.valueOf("emp"));
try {
Put put1 = new Put(Bytes.toBytes("row1"));
Put put2 = new Put(Bytes.toBytes("row2"));
Put put3 = new Put(Bytes.toBytes("row3"));

// add the data in HBase table
put1.addColumn(Bytes.toBytes("education"), Bytes.toBytes("Type"),Bytes.toBytes("Regular"));
put2.addColumn(Bytes.toBytes("education"), Bytes.toBytes("Course"),Bytes.toBytes("MCA"));
```

```
put3.addColumn(Bytes.toBytes("education"), Bytes.toBytes("Duration"),Bytes.toBytes("03 years"));
put1.addColumn(Bytes.toBytes("post"), Bytes.toBytes("Technical"),Bytes.toBytes("Java"));
put2.addColumn(Bytes.toBytes("post"), Bytes.toBytes("Department"),Bytes.toBytes("MMG"));
put3.addColumn(Bytes.toBytes("post"), Bytes.toBytes("Salary"),Bytes.toBytes("50000"));
table.put(put1);
table.put(put2);
table.put(put3);

System.out.print("Operation Done");
}

finally
{
table.close();
connection.close();
}
}
}
```

Delete the data:

```
package createTable;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Delete;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.Result;

public class deletedata{
public static void main(String args[])throws IOException{
Configuration config = HBaseConfiguration.create();
HTable table=new HTable(config, "emp");
Delete del=new Delete(Bytes.toBytes("row1"));
del.deleteColumns(Bytes.toBytes("education"),Bytes.toBytes("Type") );
table.delete(del);
System.out.println("details-deleted");
table.close();
}
}
```

HBase – Scan, Count and Truncate

The **scan** command is used to view the data in HTable. The syntax is: scan '<table name>'

Truncate command is used to This command disables drops and recreates a table.

HBase Security

There are three commands for security purpose: **grant, revoke, and user_permission.**

1.grant

The grant command grants specific rights such as read, write, execute, and admin on a table to a certain user.

hbase> grant <user> <permissions> [<table> [<column family> [<column; qualifier>]]

We can grant zero or more privileges to a user from the set of RWXCA, where

- **R - represents read privilege.**
- **W - represents write privilege.**
- **X - represents execute privilege.**
- **C - represents create privilege.**
- **A - represents admin privilege.**

2.revoke

The revoke command is used to revoke a user's access rights of a table.

hbase> revoke <user>

3.user_permission

This command is used to list all the permissions for a particular table. The syntax of user_permission is as follows:

hbase>user_permission ‘tablename’