Advanced HBase

Big Data Analytics Series: Lecture 15

Dr. Mansaf Alam
Associate Professor
Department of Computer Science,
Jamia Millia Islamia, New Delhi

Managing HBase

- Cloudera Manager, can manage and configure several aspects of HBase.
- Cloudera Manager needs certain extra steps to set up and configure HBase service.

Ref[1 &3]

Creating the HBase Root Directory

- **Minimum Required Role:** Cluster Administrators (it is also providing by Full Administrator)
- When you are going to add the HBase service, a root directory for HBase in HDFS is created automatically by **Add Service** wizard.
- When you leave the **Add Service** wizard or it does not finish, you can create the root directory outside the wizard by following these steps:
  - Select **Create Root Directory** from the **Actions** menu in the **HBase > Status** tab.
  - Click on **Create Root Directory** and again to confirm it.

Ref[3]

Graceful Shutdown

- **Minimum Required Role:** Operator, it is also provided by Configurator, Cluster Administrator, Full Administrator.
- The graceful shutdown of an HBase RegionServer permits the regions hosted by that RegionServer to be moved to other RegionServers before ending RegionServer.

Ref[3]

Cloudera Manager offers the following:

- Configuration choices to do a graceful shutdown of either an HBase RegionServer or the entire service.
- To increase the speed of a rolling restart, HBase service, set the Region Mover Threads property to a advanced value.
- This enhances the number of regions that can be moved in parallel, but places extra strain on the HMaster. In most cases, Region Mover Threads should be set to 5 or lower.

Ref[3]

Data Inserting using HBase Shell

- To insert data in an HBase table, the following commands and methods can be used :

(i) put command,

(ii) add() method of **Put** class, and

(iii) put() method of **HTable** class.

Ref[2]

- Let us consider an example, we are going to insert the following table in HBase.

Table Name: Emp

Column Families

Row Key	Personal data		Professional Data	
EmpId	EName	EState	Position	Salary
1	Arshad	UP	Associate Professor	170000
2	Kashish	Delhi	Asstt. Professor	150000
3	Monica	Punjab	Professor	200000

put command is used to insert rows into a table.

Syntax

hbase(main):006:0> put ’<tablename>’,’row1’, ’<colfamily:colname>’,’<value>’

Exampe: First Row insertion

Let us insert the first row values into the Emp table as given below.

hbase(main):006:0> put ‘Emp','1',‘Personal Data:EName',‘Arshad'

0 row(s) in 0.6600 seconds

hbase(main):006:0> put ‘Emp','1',‘Personal Data:EState',‘UP‘

0 row(s) in 0.0410 seconds

hbase(main):007:0> put 'emp','1',‘Professional Data:Position',‘Associate Professor‘

0 row(s) in 0.0240 seconds

hbase(main):008:0> put 'emp','1',‘Professional Data:Salary',‘170000‘

0 row(s) in 0.0240 seconds

You can Insert the remaining rows using the put command in the same way.

If you have inserted the whole table in HBase table, you can see the following output.

hbase(main):022:0> scan ‘Emp' ROW COLUMN+CELL

1 column=Personal Data:EName, timestamp=1417524185058, value=Arshad

1 column=Personal Data:EState, timestamp=1417524216501, value=UP

1 column=Professional Data:Position, timestamp=1417524232601, value=Associate: Professor

1 column=Professional Data:Salary, timestamp=1417524244109, value=170000

2 column=Personal Data:EName, timestamp=1417524556125, value=Kashish

2 column=Personal Data:EState, timestamp=1417524574905, value=Delhi

2 column=Professional Data:Position, timestamp=1417524592204, value=Asstt:Professor

2 column=Professional Data:Salary, timestamp=1417524604221, value=150000

3 column=Personal Data:EName, timestamp=1417524672067, value=Monica

column=Personal Data:EState, timestamp=1417524681780, value=Punjab

3 column=Professional Data:Position, timestamp=1417524693187, value=Professor

3 column=Professional Data:Salary, timestamp=1417524702514, value=200000

Data inserting Using Java API

- We can insert data into HBase Table using the **Put** class method **add()**.
- We can save it using the **put()** method of the **HTable** class.
- All these classes belong to the **org.apache.hadoop.hbase.client** package.

Ref[2]

The following are the steps to insert data in a HBase Table:

Step 1:Instantiate the Configuration Class

- The **Configuration** class is used to adds HBase configuration files to its object.
- We can create a configuration object using the **HbaseConfiguration** class, **create()** method that is given below.

```
hbase(main):008:0> Configurationconf
=HbaseConfiguration.create();
```

Step 2:Instantiate the HTable Class

- We have a class called **HTable**, an execution of Table in HBase.
- This class is used to communicate with a single HBase table.
- If we instantiate this class, it accepts configuration object and table name as parameters. We can instantiate **HTable** class as given below:

```
hbase(main):008:0> HTable hTable = new HTable(conf, tableName);
```

Step 3: Instantiate the PutClass

- To insert data into a table of HBase, the **add()** method and its variants are used.
- This method belongs to Put, so instantiate the put class.
- This class needs the row name we want to insert the data into, in string format.
- We can instantiate the Put class as given below:

```
hbase(main):008:0> Put p = new
Put(Bytes.toBytes("row1"));
```

Step 4: Insert Data

- Put class, add() method is used to insert data. It needs 3 byte arrays representing column family, column qualifier (column name), and the value to be inserted, respectively.
- Insert data into the table of **Hbase** using the **add()** method as follows:

```
hbase(main):008:0>
p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes("column name"),Bytes.toBytes("value"));
```

Step 5: Save the Data in Table

- After inserting the essential rows, save the changes by adding the put instance to the **put() method** of HTable class as follows:

```
hbase(main):008:0> hTable.put(p);
```

Step 6: Close the HTable Instance

- We have to close the HTable after inserting data into the HBase Table, close the HTable instance using the close() method as given below:

```
hbase(main):008:0> hTable.close();
```

Data Updating using HBase Shell

- We can update an existing cell value using the **put** command of HBase.
- To update so, we have to follow the same syntax and mention your new value as follows:

hbase(main):006:0> put ‘table name’,’row ’,'Column family:column name',’new value’

The afresh given value changes the existing value, updating the row.

Ref[2]

Example

Let us consider a table in HBase called E**mp** with the following data.

hbase(main):006:0> scan ‘Emp' ROW COLUMN + CELL row1 column = Personal:EName, timestamp = 1418051555, value = Arshad row1 column = Personal:EState, timestamp = 1418275907, value = UP row1 column = Professional:Position, timestamp = 14180555,value = Associate Professor row1 column = Professional:Salary, timestamp = 1418035791555,value = 170000

1 row(s) in 0.0100 seconds

The following HBase command will update the State value of the employee named ‘Kashish’ to UP.

hbase(main):006:0>put ‘Emp','row1',‘Personal:EState',‘UP‘

0 row(s) in 0.0400 seconds

The updated table looks as follows where you can observe the state of Kashish has been changed to ‘UP’.

hbase(main):006:0>scan ‘Emp' ROW COLUMN + CELL row1 column = Personal:EName, timestamp = 1418035791555, value = Kashish row1 column = Personal:EState, timestamp = 1418274645907, value = UP row1 column = Professional:Position, timestamp = 141857555,value = Asstt. Professor row1 column = Professional:Salary, timestamp = 1418039555, value = 150000

1 row(s) in 0.0100 seconds

Data Updating Using Java API

- **put()** method can be used to update the data in a particular cell.
- There are various steps to update an existing cell value of a table.
- The steps are given on next slide.

Ref[2]

Step 1: Instantiate the Configuration Class

- **Configuration** class is used to adds HBase configuration files to its object.
- **create()** method of the **HbaseConfiguration** class can be used create a configuration object, which is given below:

hbase(main):006:0>Configuration conf = HbaseConfiguration.create();

Step 2: Instantiate the HTable Class

- **HTable** is class, which is an implementation of Table in **HBase**.
- This class is used to communicate with a single HBase table.
- When instantiating this class, it takes the configuration object and the table name as parameters.
- We can instantiate the HTable class as given below:

```
hbase(main):006:0> HTable hTable = new HTable(conf, tableName);
```

Step 3: Instantiate the Put Class

- To insert data into HBase Table, the **add()** method and its variants are used.
- This method belongs to **Put**, therefore instantiate the **put** class.
- This class requires the row name you want to insert the data into, in string format.
- You can instantiate the **Put** class as shown below.

```
hbase(main):006:0>Put p = new
Put(Bytes.toBytes("row1"));
```

Step 4: Update an Existing Cell

- **Put** class, **add()** method is used to insert data.
- It needs 3 byte arrays that representing column family, column qualifier (column name), and the value to be inserted, respectively. Insert data into HBase table, we can use **add()** method as given below:

Syntax

```
hbase(main):006:0>p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes("column name"),Bytes.toBytes("value"));
```

Example

```
hbase(main):006:0>p.add(Bytes.toBytes(“Personal") , Bytes.toBytes(“EState"),Bytes.toBytes("Delhi"));
```

Step 5: Save the Data in Table

- After insertion of essential rows, we can save the changes by adding the put instance to the **put()** method of the HTable class as follows:

hbase(main):006:0>hTable.put(p);

Step 6: Close HTable Instance

After insertion data in HBase Table, we can close the **HTable** instance using the close() method which is given below:

hbase(main):006:0>hTable.close();

Reading Data using HBase Shell

We can use **get** command and **get()** method of **HTable** class to read data from a table in HBase. Using **get** command, we can get a single row of data at a time.

syntax:

hbase(main):006:0>get '<table name>','row1'

Ref[2]

Example

The given below example shows how to use the get command. Let us scan the first row of the **Emp** table.

```
hbase(main):006:0> get ‘Emp', '1' COLUMN
CELL Personal : EState timestamp =
1417521848375, value = UP Personal : EName
timestamp = 1417521785385, value = Arshad
Professional: Position timestamp =
1417521885277, value = Associate Professor
Professional: Salary timestamp =
1417521903862, value = 170000

4 row(s) in 0.0270 seconds
```

Reading a Specific Column

The following is the syntax to read a specific column with the help **get** method.

hbase(main):006:0> get 'table name', ‘rowid’, {COLUMN ⇒ ‘column family:column name ’}

Example

hbase(main):006:0> get ‘Emp', 'row1', {COLUMN ⇒ ‘Personal:EName'} COLUMN CELL Personal:EName timestamp = 1418035791555, value = Arshad

Ref[2]

1 row(s) in 0.0080 seconds

Data Reading Using Java API

get() method of the HTable class is used to read data from an HBase table. This method needs an instance of the **Get** class. There are various steps are used to retrieve data from the HBase table, which is given on next slide.

Ref[2]

Step 1: Instantiate the Configuration Class

Configuration class adds HBase configuration files to its object. **create()** method of the **HbaseConfiguration** class is used to create a configuration object which is given below:

hbase(main):006:0> Configuration conf = HbaseConfiguration.create();

Step 2: Instantiate the HTable Class

- We have a **HTable** class and that is an implementation of Table in HBase. This class communicate with a single HBase table. While instantiating this class, it accepts the configuration object and the table name as parameters. We can instantiate the HTable class as below.

hbase(main):006:0> HTable hTable = new HTable(conf, tableName);

Step 3: Instantiate the Get Class

We can retrieve data from the table of HBase using the method **get()** which class of **HTable**. This method extracts a cell from a given row. It needs a **Get** class object as parameter. We can Create it as follows:

```
hbase(main):006:0> Get get = new
Get(toBytes("row1"));
```

Step 4: Read the Data

- When we retrieving data, we can get a single row by id, or get a set of rows by a set of row ids, or scan an whole table or a rows subset.
- We can retrieve an HBase table data using the add method variants in **Get** class.
- To get a specific column from a specific column family, use the given below method:

get.addFamily(personal) To get all the columns from a specific column family, use the method: get.addColumn(personal, name)

Step 5: Get the Result

- We can get the result by passing our **Get** class instance to the **HTable** class, **get** method . This method returns the **Result** class object, which holds the requested result. We can use **get()** method as follows:

hbase(main):006:0> Result result = table.get(g);

Step 6: Values Reading from the Result Instance

- The **Result** class offers the **getValue()** method to read values from its instance. We can use this as given below:

```
hbase(main):006:0> byte [] value =
result.getValue(Bytes.toBytes(“Personal"),Bytes.
toBytes(“EName")); byte [] value1 =
result.getValue(Bytes.toBytes(“Personal"),Bytes.
toBytes(“EState"));
```

Deleting a Specific Cell in a Table

To delete a specific cell in a table, we can use the **delete** command.

Syntax

hbase(main):006:0> delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’

Example

Here we are going to deleting the state .

hbase(main):006:0> delete ‘Emp', '1', ‘Personal Data:EState', 1417521848375 Ref[2]

- 0 row(s) in 0.0060 seconds

Deleting All Cells in a Table

Using the “deleteall” command,we can delete all the cells in a row.

Syntax

hbase(main):006:0> deleteall ‘<table name>’, ‘<row>’,

Example

Here is an example of “deleteall” command, where we are deleting all the cells of row1 of Emp table.

hbase(main):006:0> deleteall ‘Emp','1'

0 row(s) in 0.0240 seconds

We can verify the table by using the command **scan.** A snapshot of the table after deleting the table is as follows:

hbase(main):006:0> scan ‘Emp' ROW COLUMN + CELL 2 column = Personal Data:Ename, timestamp = 1417524574905, value = Arshad

2 column = personal data:EState, timestamp = 1417524556125, value = UP

2 column = Professional Data:Position, timestamp = 1417524204, value = Associate Professor

column = Professional Data:Salary, timestamp = 1417524604221, value = 170000

3 column = Personal Data:Name, timestamp = 1417524681780, value = Kashish

3 column = Personal Data:EState, timestamp = 1417524672067, value = Delhi

3 column = Professional Data:Position, timestamp = 1417523187, value = Asstt. Professor

3 column = Professional Data:Salary, timestamp = 1417524702514, value = 150000

Data Deleting Using Java API

We can use **delete()** method of the **HTable** class to delete data from table of an Hbase. The steps for deleting data from a table are given below:

Step 1: Instantiate the Configuration Class

The **Configuration** class is used to adds HBase configuration files to its object. We can used **create()** method of **HbaseConfiguration** class to create a configuration object. Ref[2]

```
Configuration conf = HbaseConfiguration.create ();
```

Step 2: Instantiate the HTable Class

HTable, an implementation in Hbase Table. This class is used to communicate with a single table of HBase. When instantiating this class, it accepts the configuration object and the table name as parameters. We can instantiate the HTable class as follows:

hbase(main):006:0>HTable hTable = new HTable(conf, tableName);

Step 3: Instantiate the Delete Class

- Instantiate the **Delete** class by passing the rowid of the row of table which is to be deleted, in format of byte array. We can also pass timestamp and Rowlock to this constructor.

hbase(main):006:0>Delete delete = new Delete(toBytes("row1"));

Step 4: Select the Data to be Deleted

We can delete data using the delete methods of the **Delete** class. There are various delete methods of this class. Select columns or column families to be deleted using those methods. Let us see this examples that show the usage of Delete class methods.

```
hbase(main):006:0>delete.deleteColumn(Bytes.
toBytes(“Personal"), Bytes.toBytes(“EName"));
delete.deleteFamily(Bytes.toBytes(“Professional
"));
```

Step 5: Delete the Data

Now we can delete the selected data by passing the **delete** instance to the **delete()** method of **HTable** class as follows:

hbase(main):006:0>table.delete(delete);

Step 6: Close the HTableInstance

We have to closed Htable Instance after deleting the data.

hbase(main):006:0>table.close();

Scaning using HBase Shell

We can view the data in HTable using **scan** command.

Syntax:

hbase(main):006:0>scan ‘<table name>’

Example

Here we are reading the **emp** table.

Ref[2]

hbase(main):006:0>scan ‘Emp' ROW COLUMN + CELL 1 column = Personal Data:EName, timestamp = 1417521848375, value = Arshad

1 column = Personal Data:EState, timestamp = 1417521785385, value =UP

1 column = Professional Data:Position, timestamp = 1417585277,value = Associate Professor

1 column = Professional Data:Salary, timestamp = 1417521903862, value = 170000

1 row(s) in 0.0370 seconds

count

We can use count command to count the number of rows of a table.

Syntax:

hbase(main):006:0>count ‘<table name>’

After deleting the first row, Emp table will have two rows. We can verify it as follows:

hbase(main):006:0>count ‘Emp'

2 row(s) in 0.090 seconds ⇒ 2

truncate

This command disables drops and recreates a table.

Syntax

hbase(main):006:0>truncate 'table name'

Example

Here we have to truncate E**mp** table.

hbase(main):006:0>truncate 'emp'

Truncating 'one' table: - Disabling table... - Truncating table...

0 row(s) in 1.5950 seconds

We use scan command to verify and we will get a table with zero rows.

hbase(main):006:0>scan ‘Emp’ ROW COLUMN + CELL

0 row(s) in 0.3110 seconds

HBase Security

- Securing an HBase cluster is a one-way operation, and moving from a secure to an unsecure Configuration should not be attempted without contacting Cloudera support for guidance.
- We can grant and revoke permissions to users in HBase. There are three commands for security purpose: **grant, revoke,** and **user_permission**.

Ref[2]

grant

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

Syntax

hbase(main):006:0>grant <user> <permissions> [<table> [<column family> [<column; qualifier>]]

We can grant 0 or additional privileges to a user from the set of RWXCA, where

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

Example

We grants all privileges to a user named ‘Samiya’.

hbase(main):006:0>grant ‘Samiya', 'RWXCA'

revoke

To revoke a user's access rights of a table using **revoke** command.

Syntax

hbase(main):006:0>revoke <user>

The following command revokes all the permissions from the user ‘Samiay’.

- **hbase(main):006:0>**revoke ‘Samiya'

user_permission

- **user_permission** command is used to list all the permissions for a specific table.

Syntax

hbase(main):006:0>user_permission ‘tablename’

The following command lists all the user permissions of ‘Emp’ table.

hbase(main):006:0>user_permission ‘Emp'

References

1. https://www.cloudera.com/documentation/enterprise/5-9-x/PDF/cloudera-hbase.pdf
2. https://www.tutorialspoint.com/hbase/hbase_quick_guide.htm
3. https://www.cloudera.com/documentation/enterprise/5-7-x/topics/cm_mc_hbase_service.html

Thank you