# Big Data Analytics
## HomeWork-2

Anjani Swetha Settipalli
bigd24

**Question1:**

**1.1 Problem Description:**

Given dataset contains questions asked on www.stackexange.com and the corresponding answers to the questions. In this problem we are going to bulk import the data into hbase table. The schema of the table has to designed and the data needs to be uploaded in the table using Map reduce job.

**1.2 Solution Strategy:**

1. Create a table schema with the required number of column families.

The Hbase bulk load process consists of two steps:

- Data Preperation through Map reduce job
- Completing the Data Load.

The Map Reduce job generates Hbase data files (Hfiles) from input data using HfileOutputFormat. This writes the output which is like the format of Hbase Internal Storage which can be loaded efficiently in Hbase.
In Hbase , usually tables are broken into number of regons. HfileOutputFormat must be configured such that Hfile it generates fits in a region. Hadoop's TotalOrderPartitioner is used to map output to the key ranges of the regions in the HBase table.
ConfigureIncrementalLoad() function in HfileOutputFormat which sets up TotalOrderPartitioner based on current region and boundaries of the table.

In order to import HFiles into Hbase table, we use a command line tool called CompleteBulkLoad.

**Data Preparation through Map reduce job:**

**Mapper:**

1. The column families of the table is selected based on the given dataset. The given dataset contains the user information who post the questions and answers in the stackexchange.com. As there is user information in the dataset we decide on a column family for User. The other information used in the dataset is the post information. Hence all the post information is grouped into column family called 'Post'.
2. Parse the given dataset as key value pairs, Replace all special characters in the dataset except '='. This would help us to easily find the values of the corresponding key. Now the dataset is in the form
3. <key1=”value1” key2=”value2” key3 =”value3”……..>

4. In the next step remove any white spaces in between the keys and values using trim ().
5. Now I replaced all quotation marks with \ which would help me to tokenize the given dataset.
6. If the returned key value is of user information like CreationDate, LastEditDate, LastActivityDate, LastActivityDate, OwnerUserId then these key values are placed as the column values in User.
7. The rest of the columns are considered as the posted information of the user and are added to the Post column family of the table.
8. Now the values are inserted in the table with two column families 'Post' and 'User'.

**Reducer:**

1. There is no need to write a reducer as HFileOutputFormat.configureIncrementalLoad() as used in the driver code sets the reducer and partitioner for us.

**Driver :**

1. The driver class takes two values as inputs. First input would be our input dataset and the second is the output path to which output file is to be written.
2. It creates a connection to the Hbase table 'bigd24-hbase-sample' , and uses HfileOutput format to create the Hfiels and finally uses these Hfiles to upload the data into Htable.

**1.3 Results:**

**Execution Steps:**

1. The table is created in the HBase shell using the following command

   Create ‘bigd24-hbase-sample’, ‘Post’, ‘User’

2. Create the store files using the following command

   HADOOP_CLASSPATH=`hbase classpath` time yarn jar edgar-airquality.jar com.sample.bulkload.hbase.HbaseBulkLoad /bigdata-hw2 '/home/cosc6376/bigd24/bulkload/build/lib/bulkoutput'

3. Bulk upload into the table using the following command

HADOOP_CLASSPATH=`hbase classpath` time yarn jar /opt/hbase/0.98.9/lib/hbase-server-0.98.9-hadoop2.jar completebulkload '/home/cosc6376/bigd24/bulkload/build/lib/bulkoutput' 'bigd24-hbase-sample'

**Output:**

The data is inserted into 'bigd24-hbase-sample'.

**1.4 Description of Resources Used:**

20 SUN X2100 nodes (shark01 - shark20)

- 2.2 GHz dual core AMD Opteron processor
- 2 GB main memory

3 SUN X2200 nodes (shark25 - shark28)

- two 2.2 GHz quad core AMD Opteron processor (8 cores total)
- 8 GB main memory

Network Interconnect

- 96 port 4xInfiniBand SDR switch (gift from Cisco Systems)
- 48 port Linksys GE switch

Storage

- 20 TB Sun StorageTek 6140 array (/home shared with crill)
- 4 TB distributed PVFS2 storage (/pvfs2)

**1.5 Measurements performed:**

The above Map reduce job is run different times to measure the performance of the given algorithm is measured. The minimum execution time across the three runs is considered as the performance time of the given algorithm.

1. **Time taken to create store files:**
   The following are the measurements of the execution time for creating the HFiles.

```
real    1m42.513s
user    0m12.720s
sys     0m1.400s

real    1m56.244s
user    0m12.272s
sys     0m1.416s

real    1m43.363s
user    0m12.856s
sys     0m1.340s

real    2m6.558s
user    0m13.124s
sys     0m1.208s

real    1m31.160s
user    0m12.576s
sys     0m1.172s
```

We consider the real time out of different timings given after executing the first command given in the execution steps.

| No of runs | Time |
|---|---|
| 1 | 1m42.513s |
| 2 | 1m56.244s |
| 3 | 1m43.363s |
| 4 | 2m6.558s |
| 5 | 1m31.160s |

The execution time might because of network traffic. We consider minimum time among them

as the execution time of the given algorithm i.e., 1m42.513s

## 2. Execution time for Bulk Upload

The time taken for uploading the bulk load is measured as follows

1.
real 0m7.181s
user 0m8.764s
sys 0m0.864s

2.

real 0m7.166s
user 0m8.776s
sys 0m1.028s

3.
real 0m7.425s
user 0m9.600s
sys 0m0.920s

4.
real 0m7.401s
user 0m9.440s
sys 0m1.012s

5.

real 0m7.469s
user 0m9.580s
sys 0m0.892s

We consider real time among the above mentioned measurements

| No of runs | Time |
|---|---|
| 1 | 0m7.181s |
| 2 | 0m7.166s |
| 3 | 0m7.425s |
| 4 | 0m7.401s |

| | |
|---|---|
| 5 | 0m7.469s |

From the above measurements we can observe that the minimum time taken by the algorithm for bulk upload is 0m7.166s. There might be changes in the execution times because of network traffic in the cluster.

**Question2:**

**2.1 Problem Description:**

Given dataset contains questions asked on www.stackexange.com and the corresponding answers to the questions. For each question there will be some answers that are accepted. The accepted answer ID for a particular question is mentioned in the question row. For every question there is a corresponding Answer Count which represents number of answers given for a particular question. In this problem we are going to find the average number of answers per question using Hbase.

**2.2 Solution Strategy:**

**Driver:**

1. The driver class takes a single parameter as an argument which is the output path. The Hbase table is passed as an input.
2. We instantiate scan object to scan everything for each row.
3. In order to determine the average answer count we need to get the question row and then the answer count corresponding to the question. Hence we narrow down our scan to take only the family column that contains the corresponding information, which is 'Post'.
4. Filter is applied on the given column family to get only the question rows, i.e., the rows which has PostTypeId = 1.
5. Now the job is created and the required Mapper and reducer classes are set.

**Mapper:**

1. The input of the given mapper are the rows whose PostTypeId = 1.
2. Answer count is extracted from the given set of records.
3. The key “Avg ” and the value of Answer count is sent as the key value pair to the reducer.

**Reducer:**

1. As combiner is optional I did not go for combiner, instead used reducer directly. The reducer receives the key value pairs from the mapper.
2. For each key obtained from mapper it goes through all the values of the key “Avg” and sums up all the Answers counts send by the mapper.
3. Counter is used in order to count the number of questions

4. Finally average is calculated by dividing the total answer counts with number of questions to get an average.

The key value in the reducer remains the same, i.e., "Avg" and the value is set to calculated average value found in the reducer

**2.3 Result:**

**Execution steps:**

The given program is takes a single argument and is executed as follows:

HADOOP_CLASSPATH=`hbase classpath` time yarn jar edgar-airquality.jar AverageCalculator /home/cosc6376/bigd24/AverageCalculator/build/lib/avgoutput

**Output:**

| | |
|---|---|
| Average | 3.7441607 |

**2.4 Description of Resources Used:**

Please refer to 1.4 Description of Resources Used.

**2.5 Measurements Performed:**

**1.** The given program is run for several times and the performance of the algorithm is noted. It is taken as the minimum time taken by the algorithm among the several runs.
We consider System time given among different times.

real 0m53.696s
user 0m12.580s
sys 0m1.308s

real 0m46.868s
user 0m14.068s
sys 0m1.300s

real 0m52.842s
user 0m12.376s
sys 0m1.296s

real 0m59.486s
user 0m12.308s
sys 0m1.232s

real 0m49.596s
user 0m13.016s
sys 0m1.376s

| No of runs | Time |
|---|---|
| 1 | 0m53.696s |
| 2 | 0m46.868s |
| 3 | 0m52.842s |
| 4 | 0m59.486s |
| 5 | 0m49.596s |

2. The given dataset is run for without using the Hbase , with the help of the code using in the first homework and the performance of the algorithm is compared when used with Hbase.

real 1m13.296s
user 0m9.192s
sys 0m0.916s

real 1m10.205s
user 0m9.412s
sys 0m0.876s

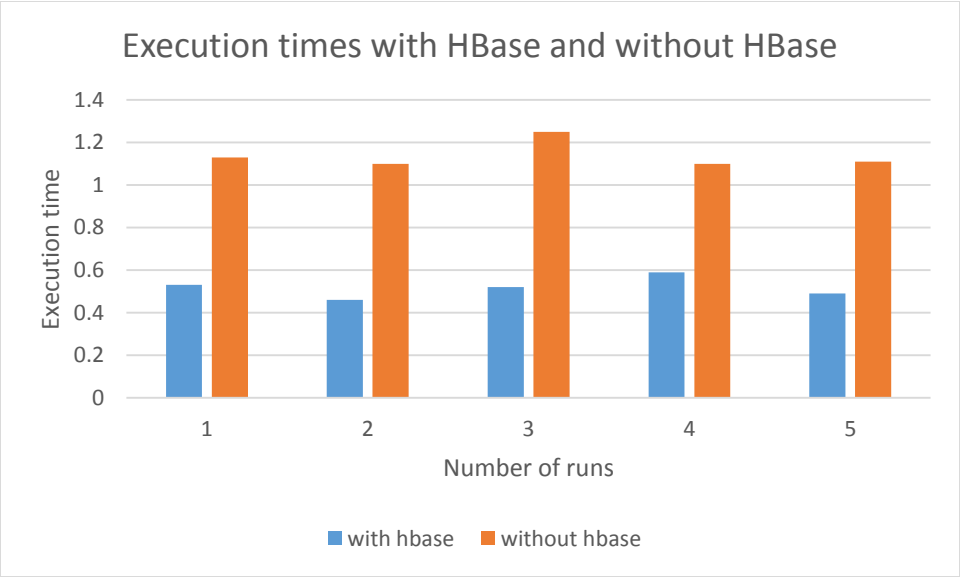
real 1m25.485s
user 0m9.320s
sys 0m0.940s

real 1m10.978s
user 0m9.560s
sys 0m0.876s

real 1m11.283s
user 0m9.192s
sys 0m0.892s

| No of runs | Time |
|---|---|
| 1 | 1m13.296s |
| 2 | 1m10.205s |
| 3 | 1m25.485s |
| 4 | 1m10.978s |
| 5 | 1m11.283s |

Now we draw a box plot comparing the execution times when used with hbase on top of hdfs and only hdfs without hbase.




From the above plot we can observe that execution times when used with hbase on top of hdfs is significantly low compared with only hdfs without using hbase .This is because HDFS lacks random read and write access. On the other hand hbase can be used for random, real/time read access to our data. The access time is reduced to great extent when we use Hbase because we are able to access the required column from the required column family and we are not processing the entire data. This greatly reduces the execution time of hbase.

References:
htrtp://pstl.cs.uh.edu/resources/shak