Project Final Report

# Team Members :

1. Jayant Jitesh
2. Ayush Bhatia

# Introduction:

In this project we worked on the StackOverflow dataset. Stackoverflow is one of the largest websites for technical discussion and we wanted to use this data to perform analysis which may be helpful to the users of this website in some way.

**Analysis Task1:**

In this analysis we find the top contributors in each category(tags). Once we have the top contributors we can suggest users to tag those top contributors in the questions. We think that this information may help users to get answers to questions quickly by tagging such contributors in the question. Of course the tagging functionality has to be implemented by the website to support this and is not in our scope.

**Analysis Task2:**

Find the response time to questions for each category(tags). Our hypothesis is “higher the frequency of category lesser is the response time” where response time is measured as the time difference between the question posted and the first answer posted. We will determine if the data supports the hypothesis or not. This analysis can be used to understand what are the popular tags and get an insight if there is any relation between the popularity and response time .

Apart from that we will analyze the performance of theta join using HBase and Hive. For that we will do the analysis 1 using HBase and Hive then compare their performance. We will explore various ways of using HBase as index to compare their respective performances.

**Highlights:**

* Challenges with data extraction or data extraction steps.
* Hbase
* Hive
* Maybe some analysis results.

**PENDING ITEMS**

intro

highlights

result discussion

log files

table for log file names and location.

conclusion and future works

# Data Description:

We are working with stackoverflow dataset. We came across this dataset in the data suggestion given in the project on blackboard. The dataset is freely available at https://archive.org/details/stackexchange . This is an anonymous dump of user-contributed content on the [Stack overflow network](http://stackexchange.com/). The stack-overflow data set contains the data related to post, user details and comments. The data set we are working on is as follow:

**Dataset Size**

stackoverflow.com-Posts 22.7 GB

stackoverflow.com-Users 662 MB

The “Posts” dataset contain the information about the post on the site. They can be either questions or answers. The “Users “data set contains the data related to a particular user like the id, reputation, age etc.

**stackoverflow.com-Posts Data**

<row Id="7" PostTypeId="2" ParentId="4" CreationDate="2008-07-31T22:17:57.883" Score="193" Body="body text" OwnerUserId="9" LastEditorUserId="967315" LastEditDate="2012-10-14T11:50:16.703" LastActivityDate="2012-10-14T11:50:16.703" CommentCount="0" />

<row Id="8" PostTypeId="1" AcceptedAnswerId="162" CreationDate="2008-07-31T23:33:19.290" Score="35" ViewCount="2834" Body="&lt;p&gt;Are there any conversion tools for porting from &lt;strong&gt;Visual J#&lt;/strong&gt; code to &lt;strong&gt;C#&lt;/strong&gt;?&lt;/p&gt;&#xA;" OwnerUserId="9" LastEditorUserId="464552" LastEditorDisplayName="Rich B" LastEditDate="2012-11-12T17:24:03.030" LastActivityDate="2013-03-29T06:32:49.430" Title="Tool for Converting Visual J# code to C#?" Tags="&lt;c#&gt;&lt;code-generation&gt;&lt;j#&gt;&lt;visualj#&gt;" AnswerCount="3" CommentCount="1" FavoriteCount="1" ClosedDate="2013-06-03T04:00:25.587" />

**Schema Description of Posts dataset:**

- Id : Row id

- PostTypeId

- 1: Question

- 2: Answer

- ParentID (only present if PostTypeId is 2)

- AcceptedAnswerId (only present if PostTypeId is 1)

- CreationDate : Date and time of post creation

- Score : Score of the post counted as the up votes

- ViewCount : Total number of views

- Body : Body of the post

- OwnerUserId : Posted by the user

- LastEditorUserId : Last edited user id

- LastEditorDisplayName : Last edited user display name

- LastEditDate : Lat date of edit

- LastActivityDate : Last date of any activity performed

- ClosedDate

- Title

- Tags

- AnswerCount

- CommentCount

- FavoriteCount

**stackoverflow.com-Users Data**:

<row Id="8" Reputation="945" CreationDate="2008-07-31T21:33:24.057" DisplayName="Eggs McLaren" LastAccessDate="2012-10-15T22:00:45.510" WebsiteUrl="" Location="" AboutMe="&lt;p&gt;This is a puppet test account I use to validate &quot;regular user&quot; stuff on the site&lt;/p&gt;&#xD;&#xA;&lt;p&gt;-- &lt;a href=&quot;http://stackoverflow.com/users/1/jeff-atwood&quot; rel=&quot;nofollow&quot;&gt;Jeff Atwood&lt;/a&gt;" Views="3243" UpVotes="12" DownVotes="9" AccountId="6" />

<row Id="9" Reputation="7116" CreationDate="2008-07-31T21:35:26.517" DisplayName="Kevin Dente" LastAccessDate="2014-01-18T01:15:27.643" WebsiteUrl="http://weblogs.asp.net/kdente" Location="Oakland, CA" AboutMe="&lt;p&gt;Independent software engineer&lt;/p&gt;&#xA;" Views="2514" UpVotes="36" DownVotes="4" Age="43" AccountId="7" />

**Schema Description of Users data:**

- Id : Unique id

- Reputation : Reputation of the user calculated by up votes and down votes

- CreationDate: User account creation date

- DisplayName: Display name of user

- LastAccessDate: Last login date of a user

- WebsiteUrl : personal website url

- Location :

- Age

- AboutMe : Description about the user

- Views : Total number of profile views

- UpVotes

- DownVotes

# Task Description and /OR Technical discussion:

1. **Data Extraction:** It contains two sub tasks:
   1. **Parsing XML data set:** In this task we parsed the xml data by using SAX java parser and wrote the required fields to a CSV file. We have used the SAX parser since the file was huge in size.

For this we wrote a simple java program and ran it on the local system.

* 1. **Normalizing the dataset:** We needed this task to normalize the data for further use. Since the answer post didn’t have the tags and multiple tags were associated with a single question post. We decide to duplicate the posts for multiple tags (one post for each tag) and populate the tags in answer post by applying the join between answer posts and question posts based on the “ParentId” of answer post and “PostId” for question post.

For this we created a Map Reduce task for this applying theta-join.

**Pseudo code:**

Intermediate Key: PostId, Record (one row of posts dataset)

Map (Object key, Text value)

{

If value.type==’Question’

emit ( PostId, Record)

else

emit ( ParentPostId, Record)

}

Reduce (PostId, Iterable<Record> values)

{

Map<Records> records=new Map<Records>()

//tags may have multiple tag

String tags;

for each Record r in values

if r.ParentId==0

tags=r.tag;

records.add(r);

for each Record r in records

for each tag t in tags

r.tag=t;

emit(r.PostId,r)

}

**Input Data:**

7,1,0,2008-07-31 22:17:57.883,0,9,,<c#><sql>,0

12,2,7,2008-07-31 22:17:57.883,0,9,,,0

**Output Data:**

7,1,0,2008-07-31 22:17:57.883,0,9,,c#,0

7,1,0,2008-07-31 22:17:57.883,0,9,,sql,0

12,2,7,2008-07-31 22:17:57.883,0,9,,c#,0

12,2,7,2008-07-31 22:17:57.883,0,9,,sql,0

2. **Hbase -Analysis 1:** For this task we needed to insert the data into HBase, apply the join on users and posts data to calculate the count and finally compute top contributors. We considered the following design options for this:

1. **Using Single table:** We created a single table for storing the posts and users data. We added a flag to the row key to identify the type of data. Based on the data type we populated different columns. All the subtasks were as follows:
   1. **Populating the data in HBase table:** We use a map-only task for this.

**Pseudo code:**

PostMap (Object key, Text value)

{

For each Record r in value

Row.RowKey=r.PostId+”posts”+TimeStamp

Row.data=r

Write(“posts”,r)

}

UserMap(Object key, Text value)

{

For each Record r in value

Row.RowKey=r.UserId+”users”

Row.data=r;

Write(“posts”,row)

}

**b. Calculating the contribution:**For calculating the count we created a map reduce that apply join on the users data and the posts data where user id of users in same as the post’s user id.

**Pseudo code:**

Map (Row r)

{

if r.flag==”users”

emit(r.userId,(r.flag,r.userName))

else

emit(r.postOwnerId,(r.flag,r.tag))

}

Reduce(Object key, Iterable value)

{

Map userContributions;

String userName

for each row r in values

if r.flag==”users”

userName=value.userName

else

userContributions.put(r.tag,count++)

for each record r in userContributions

write((userName+r.tag),r.count)

}

**c. Finding top-k contributors:** We used the top-k design pattern for this (As mentioned in the “Top-K Records” slide of lecture 5).

1. **Using two tables:** We created separate table for storing the posts and users data. All the subtasks were as follows:
   1. **Populating the data in HBase table:** We use a map-only task for this.

**Pseudo code:**

PostMap (Object key, Text value)

{

For each Record r in value

Row.RowKey=r.PostId+TimeStamp

Row.data=r

Write(“posts”,r)

}

UserMap(Object key, Text value)

{

For each Record r in value

Row.RowKey=r.UserId

Row.data=r;

Write(“users”,row)

}

**b. Calculating the contribution:** For calculating the count we created a map reduce that apply join on the users data and the posts data where user id of users in same as the post’s user id. We considered two options:

1. Using MultipleTableInputFormat: We were able to run this on local system but since it was included in later versions of HBase we were not able to run on AWS.

2. Scanning one table when required: In this approach, We provided the posts table as the input to the map reduce task. The user table is used to get the user-name so we scan the user table for the given user-id during reduce to get the user-name.

**Pseudo code:**

Map (Row r)

{

emit(r.postOwnerId,r.tag)

}

**Reduce:**

setUp()

{

hTable=new HTable(“users”);

}

Reduce(Object key, Iterable value)

{

String userName=hTable.get(key);// key is user id;

Map userContributions;

for each row r in values

userContributions.put(r.tag,count++)

for each record r in userContributions

write((userName+r.tag),r.count)

}

cleanUp()

{

hTable.close()

}

**c. Finding top-k contributors:** We used the top-k design pattern for this (As mentioned in the “Top-K Records” slide of lecture 5).

**3. Hive - Analysis 1 and Analysis 2**

Following are the steps followed to work on analysis1

**Analysis1**

**Step1:** Create table postdata and load the data into the table.

**Step2**: Create table user and load the data into the table.

**Step3**:Join the postdata table with the user table on userid of user table and owner id of postdata table.

**Step4**:select the tag from postdata table, username from user, count of the userid group by the tag from postdata table and username from the user table and order by the count of user in descending order from step3.

**Step5**:Assign row\_number ,partition by tag and order by tag and count of user step4.

**Step6**: select tag,username and the count of user from step5 and select the row number between 1 to N(top n) ,which gives the top users with count within each category.

Following are the steps followed to work on analysis2

**Analysis2**

**Step1:** Create table postdata1 and postdata2 and load the data to two tables.

**Step2**: Join the postdata table(table1) with the postdata table (table2) on id of table1 with parentid of table2 and tag of table1 with tag of tabble2.

**Step3**:Select the id from table1, parentid from table2, tag from table1 and timestamp from the Step2.

**Step4**:Assign the row number ,partition by parentid and tag and order the time by ascending order to get the most recent time to Step2.

**Step5**:From Step4 get the tag, count of tag and the time difference and select the row\_number as 1 , group by the tag and order by the count of the tag in descending order.

For self join , same steps were performed except that only a single table of postdata was used.

**Design options:** Our goal with Hive was to get the correct implementation for the two analysis proposed and then try out new things to learn more about Hive.

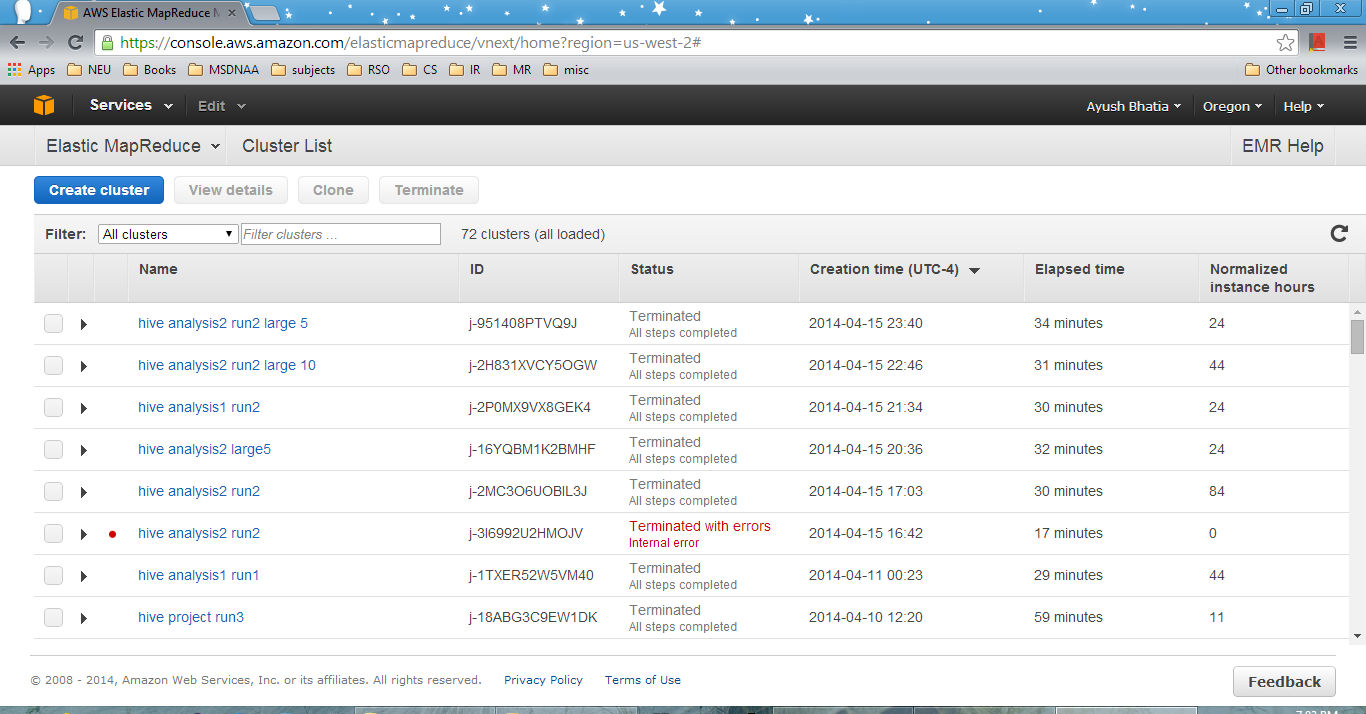
a)We could have used either row number or rank function. row\_number was used because rank is a user defined function and we first wanted to use the built in functions. We later realized that rank is supported as a built in function from Hive 0.11 onwards. We then tried using rank and there was no difference in performance between rank and row\_number . In our analysis2 we needed to get the first row for within each category to get the first response and it did not matter if we used row\_number or rank . In the analysis1 we used row\_number because row\_number continues to assign unique numbers in case of a duplicate value whereas rank assigns same numbers to two values if they are same.

b)We used a variation of query to find the difference and average in a subquery to see if there was any improvement in the performance or reduction in the number of mapreduce jobs but did not see any change .

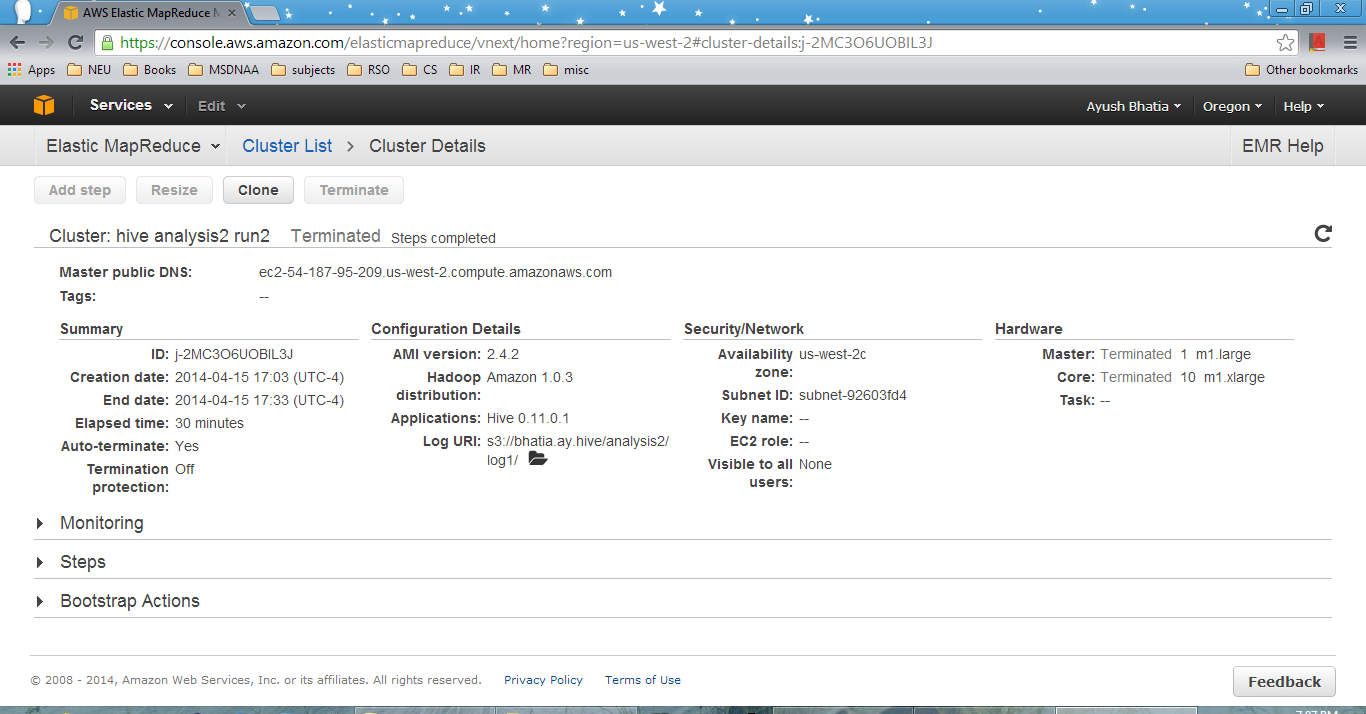
c)For analysis2 we tried two variations. We ran a query with self join and another query with two tables with the same data and join on them to see if there was a significant difference in running time between the two approaches and which one is better. There was some difference in running time but it was negligible.

# Hive Screenshots

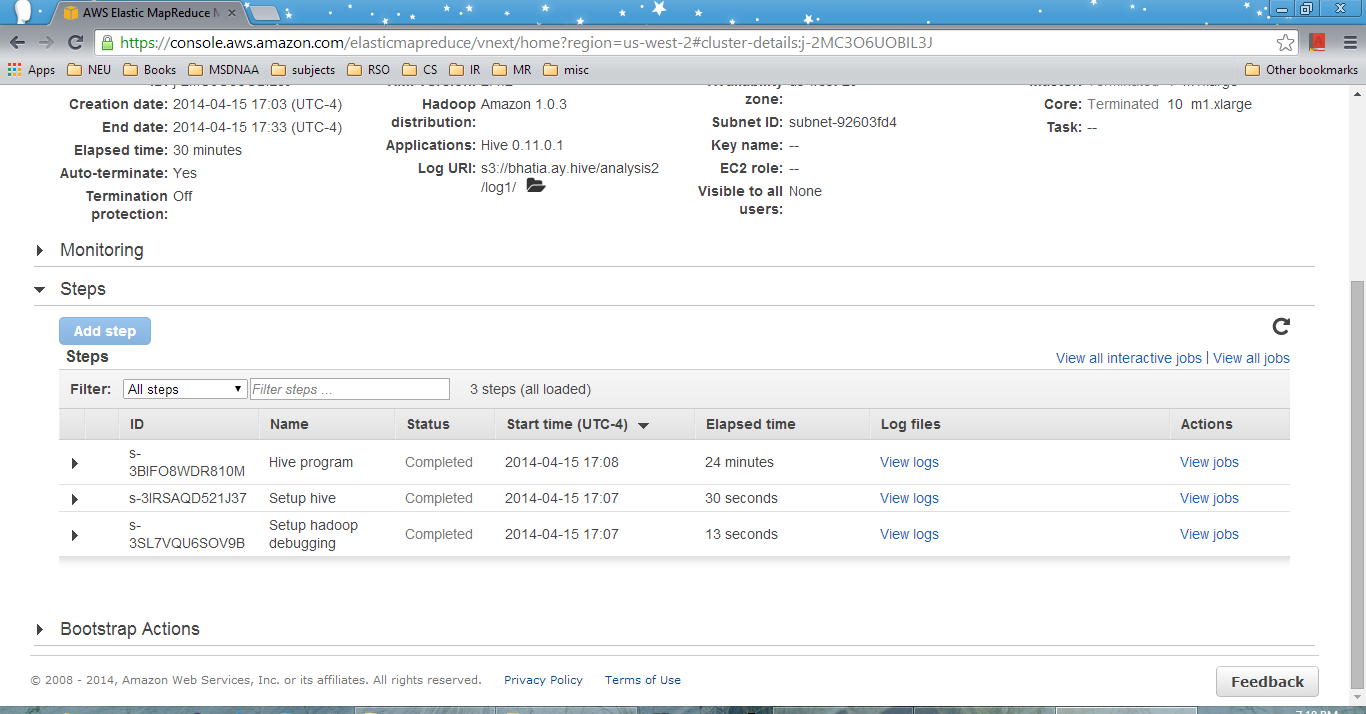
Screenshot of Hive job runs for various configurations. The screenshot shows a list of clusters used to run hive jobs.



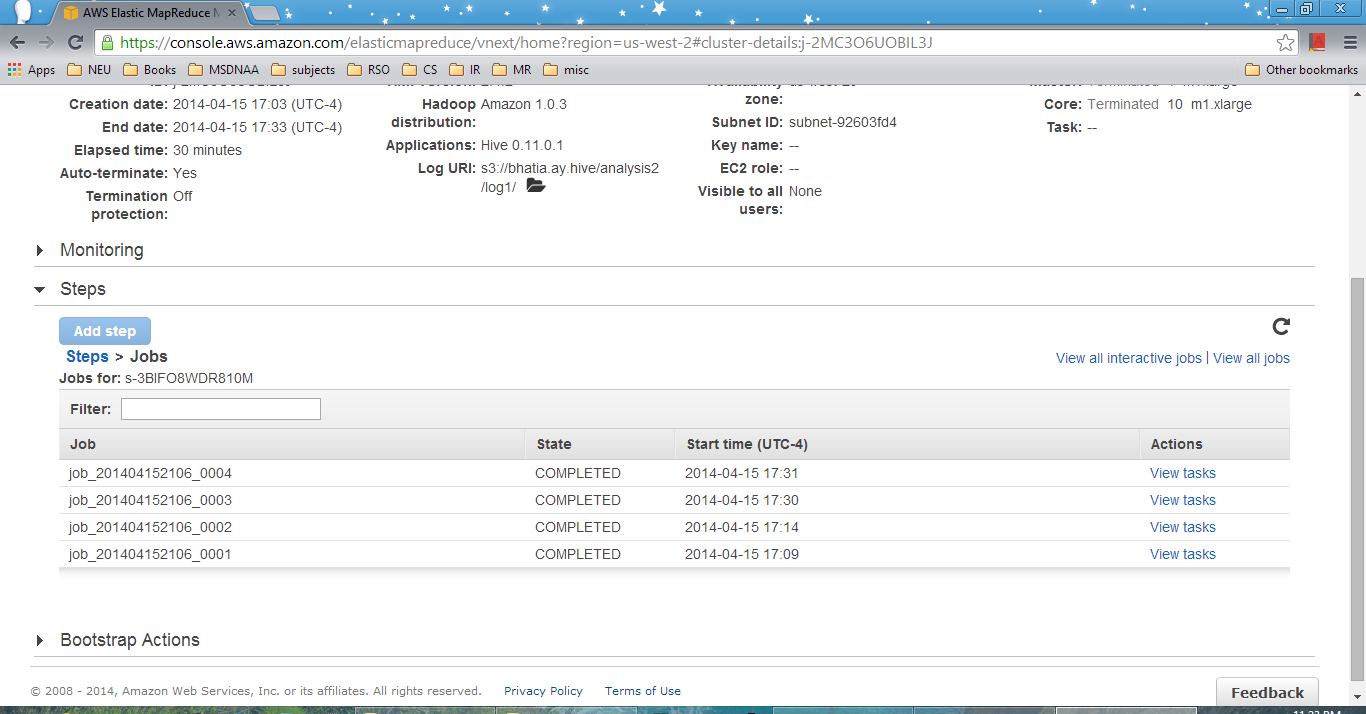
Screenshot showing successful run of a Hive job



Screenshot showing successful run of steps inside a Hive job



Screenshot showing job completion inside hive program



# Results discussion and visualization/graphs :

**Job Run time and CPU time Comparison Table:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **configurations** | **Running Time** | | | **CPU Time** | | |
|  | **5 Large** | **10 Large** | **15 large** | **5 Large** | **10 Large** | **15 large** |
| **Hbase Analysis 1 with two tables** | **NA** | **1325** | **1142** | **NA** | **1880.61** | **1705.05** |
| **Hbase Analysis 1 with single tables** | **NA** | **1324** | **1185** | **NA** | **1652.77** | **1727.2** |
| **Hive Analysis 1 with two tables** | **1279** | **1211** | **NA** | **3416.82** | **3399.2** | **NA** |
| **Hive Analysis 2 with self-join** | **1416** | **1282** | **NA** | **4336.67** | **4248.85** | **NA** |
| **Hive Analysis 2 without self- join** | **1416** | **1241** | **NA** | **4632.2** | **4833.24** | **NA** |

**Graphs:**

Figure1: Graph showing average response time VS Tags (along with the count)

Figure2: Graph showing average response time VS Tags (along with the count)

Figure 3**:** Graph showing top 5 contributors in each category along with the contribution of each user for that category.

**Discussion and comparison:**

In Analysis 1, we were able to find top 5 contributors for each hashtags. Figure 3 is showing top 5 for some of the popular categories. For performance comparison we found that the results came from hive and HBase are identical as expected. The running time for Hive is almost same to the running time of HBase Map Reduce task. The real difference can be seen for populating the data into Hive and HBase using java MpaReduce. Populating the data using MapReduce took 39 minutes while Hive task took 90 seconds. Apart from that Hive task is more CPU intensive as compared to the HBase map reduce task.

In Analysis2, the results were similar to our expectation. Our hypothesis that “higher the frequency of category lesser is the response time” is valid for most of the cases however for some cases (see figure 2) like “ouputcache” has count 400 and response time around 2650 seconds while “aspnetdb” has count 89 and response time around 1000 seconds. It can be observed that the response time doesn’t depend only on frequency it depends on some other factors like topic is new or old, popularity etc.

# Hive Setup Challenges on AWS:

1. Hive has a good documentation but the documentation lacks examples so it takes more time understanding the documentation. https://cwiki.apache.org/confluence/display/Hive/GettingStarted is a good starting point for Hive and was helpful in setting up Hive in local machine.

This is a good link to get started on AWS but there are other problems that we faced as detailed below. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/CLI\_CreatingaJobFlowUsingHive.html. Hive can be run in either interactive mode or batch mode. We ran Hive in Batch mode. We tried different combinations on EMR 12 times until it worked. The problems and description are given below.

2. To specify input, it can be given in INPUT parameter while creating the job flow, but for more than one input you need to specify it in the arguments field. For example if there is just one input table ,it can be specified using {$INPUT} in the script and this is replaced by actual INPUT file that is provided in the INPUT parameter when creating Hive job flow. When you need another file as input you need to specify another variable in the script ,for example {$INPUT1} in script and then give the actual location on s3 in the arguments field while creating job flow. The syntax is -d INPUT1=s3://location of the file on s3 . We faced this issue when we wanted to use two table and then found out that the other variables can be passed as arguments in the job .

3. The output file where the output is written is to be given in the script. It did not work for me if specified in the OUTPUT using a variable such as {$OUTPUT} . The output should be a folder and the location has to be on s3. For example the syntax is INSERT OVERWRITE DIRECTORY 's3n://location' .

4. While creating a table on EMR you need to give the location also where it will be stored even though it creates a table. In the script you need to mention the s3 location . For example if the INPUT is a CSV file called XYZ.CSV then the script must specify the location where the table will be created from the input on s3. Here input is the INPUT that you specify in the EMR console while creating Hive job. For example the syntax is create table tablename(table columns) "stored as textfile location 's3://location' " . We were unable to run the job until we included "stored as textfile location 's3://location' ".

5. There are two ways of storing the output result of a hive query.

a) Write it to a file using INSERT OVERWRITE DIRECTORY. You need to specify the complete s3 location.

b) Create a table first and then write the output in the table. For example create a table with the columns along with the data types and then insert the output of the query using INSERT INTO table name.

6. By default the hive output is separated by ^A and "ROW FORMAT DELIMITED FIELDS TERMINATED BY ','" in the script does not work for writing the output file. If ',' would have been supported then it would have been easier to open the output file directly in CSV format. We used notepad++ to change ^A to , and then open in xls . However "ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' " works on the local Hive setup without any problem.

7. We wanted to try a UDF in Hive but after several tries we could not get it working on local Hive setup . The UDF that we tried is called rank and as mentioned before we later realized that it was not present in earlier versions of Hive but was included as a standard function in hive 0.11.

8. Since the number calculation can go beyond double data type, decimal data type is used because we were getting incorrect results when we used INT datatype for calculations. Decimal is made available only from 0.11 in hive and we used it in our query.

9. Even though Hive is similar to SQL there are many differences for example it dint have CHAR dataype until Hive 0.13.0. Sometimes the query looks correct and fails and then you later realize that it is not present in that version of Hive.

10. While creating cluster make sure that Pig is not installed because Pig is selected by default . There is no problem as such even if Pig is selected and you run hive but I think it may get included in the bill so make sure to uncheck it.

# Major Problems

1. In analysis 2 we had to find the time difference to calculate the response time. The time is in the format "2008-07-31T21:33:24.057 " . While Hive has separate functions to find difference between year, month, hours there is no function to find the timestamp difference between two timestamps. After several tries we found that the difference can be calculated just by subtracting the two timestamps but before that the timestamps has to be cast to double or decimal .

2. We tried using a UDF for rank function which is similar to row\_number but it dint not work correctly in the local setup and so we did not run the same on EMR. Later we found that rank is supported in Hive 0.11 and but could not find support for that in the EMR documentation.

3. Hive 0.11.0 introduced several windowing and analytic functions and we tried to use rank, row\_number,over with partition and order by and were successful. We tried other functions such as FIRST\_VALUE,LAST\_VALUE and were unable to run successfully because of lack of documentation. This link has some documentation for windowing and analytic but it is not clear and there are no examples. Also we did not find any answer related to it in any other forum. <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+WindowingAndAnalytics#LanguageManualWindowingAndAnalytics-PARTITIONBYwithonepartitioningcolumn,oneORDERBYcolumn,andnowindowspecification>

4. Sometimes when choosing a design decision for Hive it may be very easy to make a mistake. For Example we have two queries below

select count(distinct (Col)) from table;

select count(\*) from (select distinct (Col) from table)

The first one looks simple and looks like it may run faster but that is not the case. In the first query , map sends each value to the reduce and single reduce counts them all. In the second case map splits the vales to many reducers and each reduce generates it list and final job counts size of each list.

5. In the post data all the tags were combined in one column. For our analysis we needed to have one tag in one row. So we duplicated the row for each tag.

6. Answer post didn't have the tag field populated. For both the analysis we need to have tags in answer post also. We came with two different options:

1. For all the analysis we can apply the join between the questions and answers posts to get the tag information.
2. We can have the join in the starting and all the analysis can use the modified dataset.

Second options seemed better and we went with that.

7. We tried using the TableMultipleInput but since it was added after HBase 94.6. We were not able to run on AWS. On AWS it required Hadoop 2.2.0 and migrating to newer Hadoop started creating problems on AWS. So we decided not to use the TableMultipleInput format.

# CONCLUSION:

After comparing the running time of HBase and Hive we found that both of them almost have same running time. Although HBase data population through MapReduce takes a lot of time as compared to Hive data population (For 10 large machines HBase took 42 minutes while Hive took 2 minutes). Once the data is loaded both of them behave same.

Considering the overall performance and effort, Hive seems to be a better choice for the Join operations as compared to HBase. But Hive task was more CPU intensive as compared to the HBase task.

While working on AWS we found a helpful guide of best practices and it would have been really useful had we found out this at the starting of course or the project. The link is

http://media.amazonwebservices.com/AWS\_Amazon\_EMR\_Best\_Practices.pdf

# FUTURE WORK:

In future works we can try to use the MultipleTableInput format and see how it affects the performance. Currently loading the data into HBase is done by Map Reduce program and it was very time consuming. In future works different options can be explored to load the data into HBase table like PigLatin, bulk loading using command line interface etc. We tried improving the hbase population task by using batch option. Instead of writing one entry in each map call we created a list and committing it after it reaches certain limit (we set it to 500). Although it didn’t improve the performance much future works may explore more performance tuning. For this project we used Hive in a batch mode. In future works the interactive approach can also be explored.

We can extend this project to find out

# Table of source folder and logs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Configuration** | **Source Folder** | **Log folder inside logs** | | |
|  |  | **5 Large** | **10 Large** | **15 large** |
| **Hbase Analysis 1 with two tables** | **Hbase\_Analysis1\_V1** | **NA** | **Hbase\_Analysis1\_V1\10Large** | **Hbase\_Analysis1\_V1\15Large** |
| **Hbase Analysis 1 with single tables** | **Hbase\_Analysis1\_V2** | **NA** | **Hbase\_Analysis1\_V2\10Large** | **Hbase\_Analysis1\_V2\15Large** |
| **Hive Analysis 1 with two tables** |  | **hive\analysis1\_5large.txt** | **hive\analysis1\_10large.txt** | **NA** |
| **Hive Analysis 2 with self join** |  | **hive\analysis2V1\_5large.txt** | **hive\analysis2V1\_10large.txt** | **NA** |
| **Hive Analysis 2 without self join** |  | **hive\analysis2V2\_5large.txt** | **hive\analysis2V2\_10large.txt** | **NA** |