Northeastern University

Final Project Report

Topic: Web Log Analysis

Engineering of Big Data
Academic Year: Summer 2020
INFO 7250
Professor: Mr. Yusuf Ozbek

by:
Pranav Khiste
NUID: 001057866
Submission Date: August 14, 2020

Motivation:

The Log data can be used to analyze and understand the behavior of any web service as well as users accessing that service. To find out patterns, system errors or exceptional user activities, web logs plays crucial role. These insights are leveraged across the enterprise level and can be used for business intelligence purpose.

Implementation:

To achieve this goal, I implemented Data Ingestion Pattern to load data into the system using MongoDB and Pig. For further analysis, I used various MapReduce algorithms.

Technologies Used:

- Libre Access
- MongoDB
- Apache Hadoop
- Apache Pig
- Tableau

Data Source: access-1.log

Link: https://northeastern.instructure.com/courses/7136/files/608290

Data Snippet:

```
127.0.0.1 - - [15/Oct/2011:11:49:11 -0400] "GET / HTTP/1.1" 200 44
127.0.0.1 - - [15/Oct/2011:11:49:11 -0400] "GET /favicon.ico HTTP/1.1" 404 209
129.10.135.165 - - [15/Oct/2011:11:59:10 -0400] "GET / HTTP/1.1" 200 6
129.10.135.165 - - [15/Oct/2011:11:59:10 -0400] "GET /favicon.ico HTTP/1.1" 404 209
129.10.65.240 - - [15/Oct/2011:12:05:58 -0400] "GET / HTTP/1.1" 200 6
129.10.65.240 - - [15/Oct/2011:12:05:58 -0400] "GET /favicon.ico HTTP/1.1" 404 209
129.10.65.240 - - [15/Oct/2011:12:07:25 -0400] "GET /favicon.ico HTTP/1.1" 404 209
146.115.62.108 - - [15/Oct/2011:12:57:44 -0400] "GET / HTTP/1.1" 200 6
146.115.62.108 - - [15/Oct/2011:12:57:45 -0400] "GET /favicon.ico HTTP/1.1" 404 209
146.115.62.108 - - [15/Oct/2011:12:57:45 -0400] "GET /favicon.ico HTTP/1.1" 404 209
129.10.135.165 - - [15/Oct/2011:13:02:34 -0400] "GET / HTTP/1.1" 200 6
129.10.135.165 - - [15/Oct/2011:13:02:34 -0400] "GET /favicon.ico HTTP/1.1" 404 209
64.134.67.129 - - [15/Oct/2011:15:53:08 -0400] "GET / HTTP/1.1" 200 6
64.134.67.129 - - [15/Oct/2011:15:53:08 -0400] "GET /favicon.ico HTTP/1.1" 404 209
```

After Preprocessing :

	A	B	C	D	E	F	G	H	I
1	IP	TIMESTAMP	ZONE	METHOD	RESOURCE	PROTOCOL	STATUS CODE	OBJECT SIZE	
2	127.0.0.1	15/Oct/2011:1	400	GET	/	HTTP/1.1	200	44	
3	127.0.0.1	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
4	129.10.13	15/Oct/2011:1	400	GET	/	HTTP/1.1	200	6	
5	129.10.13	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
6	129.10.65.	15/Oct/2011:1	400	GET	/	HTTP/1.1	200	6	
7	129.10.65.	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
8	129.10.65.	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
9	146.115.6	15/Oct/2011:1	400	GET	/	HTTP/1.1	200	6	
10	146.115.6	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
11	146.115.6	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
12	129.10.13	15/Oct/2011:1	400	GET	/	HTTP/1.1	200	6	
13	129.10.13	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
14	64.134.67.	15/Oct/2011:1	400	GET	/	HTTP/1.1	200	6	
15	64.134.67.	15/Oct/2011:1	400	GET	/favicon.ico	HTTP/1.1	404	209	
16	182.72.44.	15/Oct/2011:1	400	GET	/w00tw00t.a	HTTP/1.1	400	226	
17	88.146.16	16/Oct/2011:1	400	GET	/scripts/setu	HTTP/1.1	404	215	
18	88.146.16	16/Oct/2011:1	400	POST	/scripts/setu	HTTP/1.1	404	215	
19	88.146.16	16/Oct/2011:1	400	GET	/phpmyadmi	HTTP/1.1	404	226	
20	88.146.16	16/Oct/2011:1	400	GET	/phpMyAdmi	HTTP/1.1	404	226	
21	88.146.16	16/Oct/2011:1	400	GET	/mysql/scrip	HTTP/1.1	404	221	

<u>Attributes used:</u>

- IP address
- Timestamp
- Time zone
- HTTP Method
- Resource
- HTTP Protocol
- Status Code
- Object size in bytes

INDEX

Starting with MongoDB:

Step 1: Start Mongo demon
cmd: systemctl start mongod

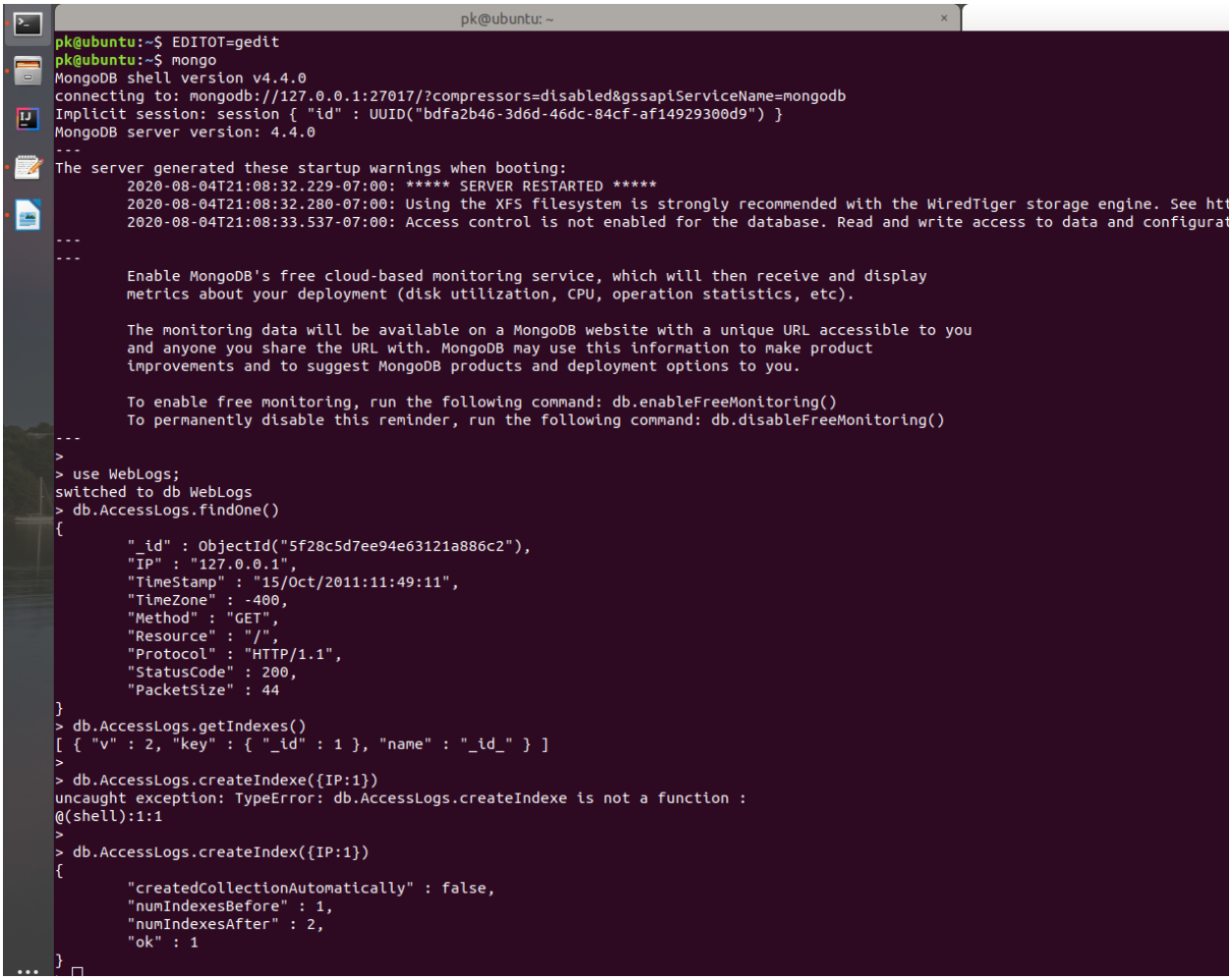
Step 2: Start Mongodb
cmd: mongo

Step 3: To import dataset, use mongo-import utility.

cmd: mongoimport --db=WebLogs --collection=AccessLogs --type=csv --headerline --file=/home/pk/Downloads/access-1.log

To export data from mongodb, use mongo-export utility.

cmd: mongoexport --db WebLogs --collection MostActiveIP --type=csv --fields _id,value --out /home/pk/Desktop/Big Data Project/2.Mongo/output/ActiveIPs.csv


pk@ubuntu: ~
pk@ubuntu:~$ EDITOT=gedit
pk@ubuntu:~$ mongo
MongoDB shell version v4.4.0
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("bdfa2b46-3d6d-46dc-84cf-af14929300d9") }
MongoDB server version: 4.4.0

The server generated these startup warnings when booting:
2020-08-04T21:08:32.229-07:00: ***** SERVER RESTARTED *****
2020-08-04T21:08:32.280-07:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See htt
2020-08-04T21:08:33.537-07:00: Access control is not enabled for the database. Read and write access to data and configurat
---

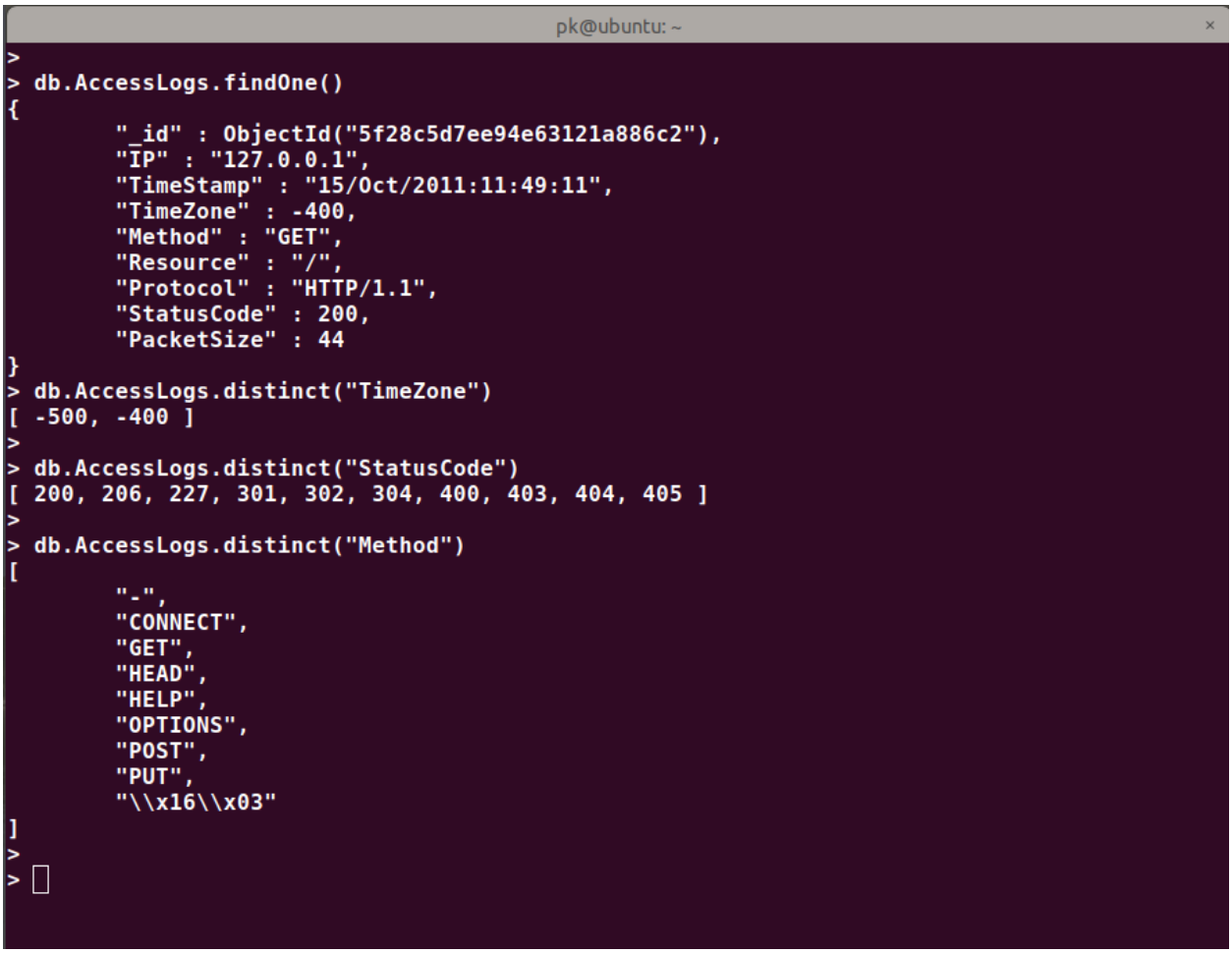
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

>
> use WebLogs;
switched to db WebLogs
> db.AccessLogs.findOne()
{
"_id" : ObjectId("5f28c5d7ee94e63121a886c2"),
"IP" : "127.0.0.1",
"TimeStamp" : "15/Oct/2011:11:49:11",
"TimeZone" : -400,
"Method" : "GET",
"Resource" : "/",
"Protocol" : "HTTP/1.1",
"StatusCode" : 200,
"PacketSize" : 44
}
> db.AccessLogs.getIndexes()
[{ "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" }]
>
> db.AccessLogs.createIndexe({IP:1})
uncaught exception: TypeError: db.AccessLogs.createIndexe is not a function :
@(shell):1:1
>
> db.AccessLogs.createIndex({IP:1})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}

Analysis 1: Unique fields from the attributes [MongoDB]

To find out unique fields from the attributes like total time zones available in the dataset, distinct functionality can be used.

> db.AccessLogs.distinct("TimeZone")
[-500,-400]



Conclusion:

- **This dataset contains users from 2 different locations.**
- **All types of HTTP methods are consumed by the user.**
- **Status code does not contain series of 500 that means servers are working fine all the time.**

Analysis 2: Most active IP address [MapRedcue- MongoDB]

Map function:
```
var map = function(){
    emit(this.IP,1);
    };
```

Reduce Function:
```
var reduce = function(key,values){
    var count = 0;
    values.forEach(function(i){
        count +=i;
    });
    return count;
};
```

> db.AccessLogs.mapReduce(map,reduce, {out: "MostActiveIP"})

```
pk@ubuntu: ~
>
> var map = function(){
... emit(this.IP,1);
... };
>
> var reduce = function(key,values){
... var count = 0;
... values.forEach(function(i){
... count +=i;
... });
... return count;
... };
>
> db.AccessLogs.mapReduce(map,reduce, {out: "MostActiveIP"})
{ "result" : "MostActiveIP", "ok" : 1 }
>
> show collections;
AccessLogs
MostActiveIP
>
> db.MostActiveIP.find().sort("value":-1)
uncaught exception: SyntaxError: missing ) after argument list :
@(shell):1:35
> db.MostActiveIP.findOne()
{ "_id" : "24.128.196.105", "value" : 5 }
>
```

Result:

```
>
> db.MostActiveIP.find().sort("value":-1)
uncaught exception: SyntaxError: missing ) after argument list :
@(shell):1:35
> db.MostActiveIP.findOne()
{ "_id" : "24.128.196.105", "value" : 5 }
>
>
> db.MostActiveIP.find().sort({"value":-1})
{ "_id" : "155.33.18.236", "value" : 4958 }
{ "_id" : "207.248.55.246", "value" : 3719 }
{ "_id" : "10.15.10.129", "value" : 2812 }
{ "_id" : "10.15.10.135", "value" : 2108 }
{ "_id" : "129.10.65.240", "value" : 1501 }
{ "_id" : "107.20.213.124", "value" : 1279 }
{ "_id" : "168.144.67.144", "value" : 760 }
{ "_id" : "50.63.154.43", "value" : 662 }
{ "_id" : "74.63.242.249", "value" : 638 }
{ "_id" : "72.158.153.33", "value" : 638 }
{ "_id" : "188.143.122.191", "value" : 637 }
{ "_id" : "118.102.182.196", "value" : 637 }
{ "_id" : "184.168.22.231", "value" : 636 }
{ "_id" : "189.126.103.45", "value" : 286 }
{ "_id" : "129.10.222.165", "value" : 279 }
{ "_id" : "213.144.108.194", "value" : 266 }
{ "_id" : "194.78.179.211", "value" : 252 }
{ "_id" : "127.0.0.1", "value" : 196 }
{ "_id" : "108.7.144.71", "value" : 195 }
{ "_id" : "202.91.240.186", "value" : 168 }
```

Analysis 3: Most Demanded Resource [MapRedcue- MongoDB]

Map function: var map = function(){
emit(this.Resource,1);
};

Reduce Function: var reduce = function(key,values){
var count = 0;
values.forEach(function(i){
count +=i;
});
return count;
};

> db.AccessLogs.mapReduce(map,reduce, {out: " MostDemandedResource"})

Result:

```
>
> db.AccessLogs.mapReduce(map,reduce, {out: "MostDemandedResource"})
{ "result" : "MostDemandedResource", "ok" : 1 }
>
> db.MostDemandedResource.find().sort({"value":-1})
{ "_id" : "/favicon.ico", "value" : { "Count" : 2604 } }
{ "_id" : "/", "value" : { "Count" : 1093 } }
{ "_id" : "/yusuf/images/color_yellow.png", "value" : { "Count" : 612 } }
{ "_id" : "/yusuf/images/color_magenta.png", "value" : { "Count" : 611 } }
{ "_id" : "/yusuf/images/color_blue.png", "value" : { "Count" : 611 } }
{ "_id" : "/yusuf/images/color_cyan.png", "value" : { "Count" : 611 } }
{ "_id" : "/yusuf/images/plus.png", "value" : { "Count" : 610 } }
{ "_id" : "/yusuf/images/adddoc.png", "value" : { "Count" : 607 } }
{ "_id" : "/yusuf/images/email.png", "value" : { "Count" : 606 } }
{ "_id" : "/yusuf/images/quicknote.png", "value" : { "Count" : 605 } }
{ "_id" : "/yusuf/images/calendar.png", "value" : { "Count" : 605 } }
{ "_id" : "/yusuf/images/discussion.png", "value" : { "Count" : 605 } }
{ "_id" : "/yiran/finalP2.zip", "value" : { "Count" : 500 } }
{ "_id" : "/yusuf/projectsummary.txt", "value" : { "Count" : 495 } }
{ "_id" : "/yusuf/discussions.txt", "value" : { "Count" : 448 } }
{ "_id" : "/yusuf/quicknotes.txt", "value" : { "Count" : 441 } }
{ "_id" : "/yusuf/biogen.html", "value" : { "Count" : 414 } }
{ "_id" : "/yusuf/images/a.jpg", "value" : { "Count" : 404 } }
{ "_id" : "/yusuf/images/b.jpg", "value" : { "Count" : 404 } }
{ "_id" : "/yusuf/images/c.jpg", "value" : { "Count" : 404 } }
Type "it" for more
> it
{ "_id" : "/yusuf/images/biogen_logo.png", "value" : { "Count" : 289 } }
{ "_id" : "/flow/", "value" : { "Count" : 277 } }
{ "_id" : "/mysql/backup.pdf", "value" : { "Count" : 204 } }
{ "_id" : "/mvc/login/", "value" : { "Count" : 157 } }
```

Starting with Hadoop:

Step 1: Start Hadoop services
cmd: > cd $HADOOP_HOME/sbin
> ./start-all.sh

Step 2: Copy dataset to HDFS from Local
Cmd: > cd $HADOOP_HOME/bin
> Hadoop fs -copyFromLocal /Fromfilepath /ToCopyLocation

Analysis 4: Total Data consumed by each IP [MapRedcue- Hadoop]

To achieve this goal, I used normal MapReduce algorithm.

Mapper:
IP address is emitted as a key along with Object size as val.

Reducer:
For each key, total size of objects is calculated and returned with each key.

Result:

```
pk@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ hadoop fs -tail /project/output/TotalData1/part-r-00000
2020-08-13 17:42:29,216 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
48
91.213.197.3    18852
91.225.98.87    436
91.241.13.27    0
91.57.108.189   6
92.240.68.152   3460
92.240.68.153   1811
92.42.51.165    0
92.52.78.238    0
92.68.44.161    0
93.114.41.133   1864
93.152.157.19   0
93.174.93.52    619
93.187.141.154  23403
93.190.138.111  1231
93.63.221.11    0
93.63.226.228   0
93.81.45.207    412
94.101.85.203   7594
94.102.51.246   66
94.103.146.231  7594
94.127.67.61    7594
94.127.68.85    6642
94.129.139.190  60416483
94.142.155.123  428
94.176.167.251  6
94.198.184.134  0
94.23.195.105   1102
94.23.205.180   9712
94.23.235.146   9712
94.23.45.14     0
94.23.61.43     9712
94.23.7.115     9712
94.90.115.82    210
95.0.87.28      226
95.110.208.32   226
95.142.103.184  5647
95.142.8.34     0
95.172.16.234   6
```

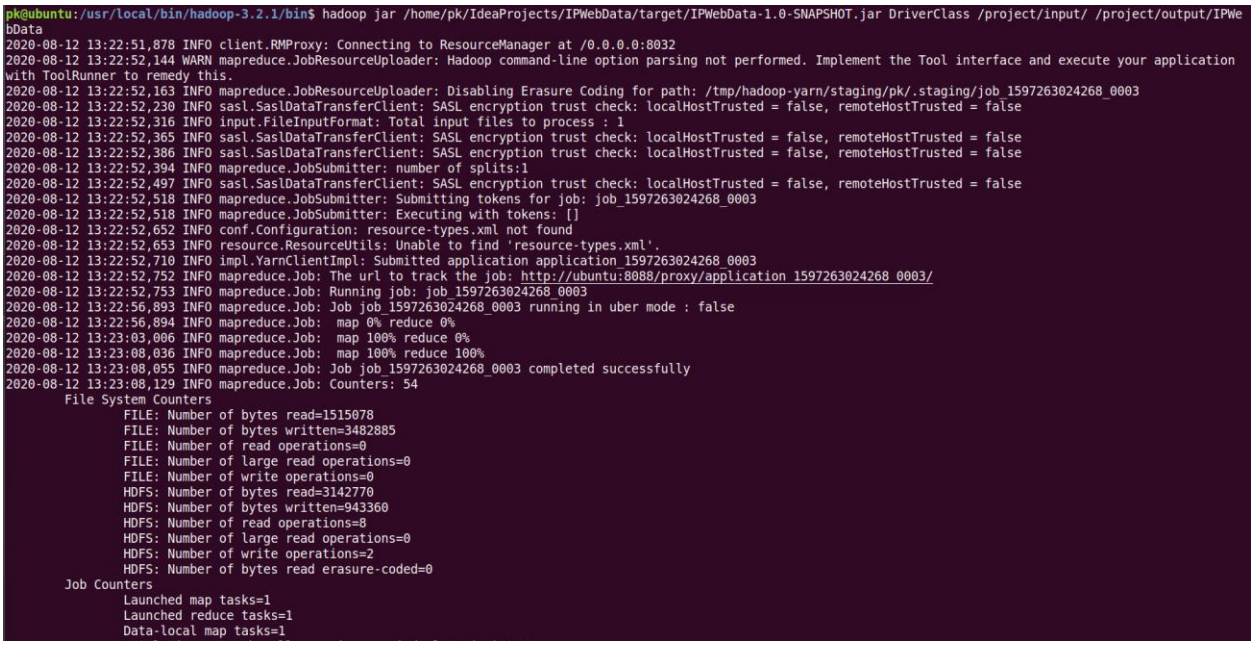
Implemented using: DriverClass, MapperClass, ReducerClass.

Analysis 5: Total data consume by the user per resource [MapRedcue- Hadoop]

To achieve this goal, I used **Composite Key Writable Object** in MapReduce algorithm.

Mapper:
IP address with resource is passed to the composite key class.
Object of composite key writable class is emitted as a Key along with object size in bytes

Reducer:
For each key, total size of objects is calculated and returned with each key.
Calculation is performed on each reducer separately as grouping of keys is done by partitioner class.



Result:

```
pk@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$
pk@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ hadoop fs -head /project/output/IPWebData/part-r-00000
2020-08-13 17:41:00,246 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
178.93.71.53     www.google.com:443     470
50.78.12.253     www.fbi.gov:80 235
87.224.70.142    www.fbi.gov:80 235
75.144.213.41    www.fbi.gov:80 235
213.154.210.10   optimum.in.ua:443      235
62.80.178.47     optimum.in.ua:443      235
213.154.210.10   optimum.in.ua:443      235
121.28.97.242    http://www.yahoo.com/  6
112.84.255.116   http://www.yahoo.com/  6
121.28.97.242    http://www.yahoo.com/  6
219.234.148.130          http://www.wikipedia.org/      6
222.205.9.250    http://www.wikipedia.org/      6
222.205.1.155    http://www.wikipedia.org/      6
222.205.9.250    http://www.wikipedia.org/      12
222.205.16.229   http://www.wikipedia.org/      6
123.234.227.103          http://www.wikimedia.org/      6
112.20.203.168   http://www.wikimedia.org/      6
119.33.8.132     http://www.wikimedia.org/      6
58.19.126.17     http://www.whu.edu.cn/ 12
58.48.47.47      http://www.w3.org/     6
175.161.175.21   http://www.springerlink.com/home/main.mpx      211
113.120.56.238   http://www.springerlink.com/   6
58.17.52.14      http://www.springerlink.com/   6
221.7.193.70     http://www.springerlink.com/   6
```

Implementation: MapperClass, DriverClass, ReducerClass, Composite Key Writable, Writable comparator, Natural Key Partitioner

Conclusion:

- Data consumed by IP address associated with each resource is calculated.
- This analysis is useful to find out **User Behavior towards application**.

Analysis 6: Total number of times resource accessed per hour.

To achieve this goal, I used Filtering Pattern in MapReduce algorithm.

Mapper:
Resource and hour is emitted as a key along with count.

Reducer:
For each key, total count is calculated.

Result:

```
pk@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$
pk@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ hadoop fs -tail /project/output/HighTrafficWeb/part-r-00000
2020-08-13 17:38:36,339 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
rigin=home&_acct=C000228598&_version=1&_urlVersion=1&_userid=10&md5=3581c309ecfeef0212f0d7f92bcb7f52    17      1
http://www.sina.com.cn/    01   2
http://www.sina.com.cn/    04   1
http://www.sina.com.cn/    05   1
http://www.sina.com.cn/    20   1
http://www.sina.com.cn/    23   1
http://www.springerlink.com/    00      1
http://www.springerlink.com/    07      1
http://www.springerlink.com/    08      1
http://www.springerlink.com/    19      1
http://www.springerlink.com/home/main.mpx    18 1
http://www.w3.org/    22        1
http://www.whu.edu.cn/    03    2
http://www.wikimedia.org/    05 1
http://www.wikimedia.org/    09 1
http://www.wikimedia.org/    13 1
http://www.wikipedia.org/    03 1
http://www.wikipedia.org/    11 1
http://www.wikipedia.org/    16 1
http://www.wikipedia.org/    20 3
http://www.yahoo.com/    06     1
http://www.yahoo.com/    13     2
optimum.in.ua:443    00 1
optimum.in.ua:443    04 1
optimum.in.ua:443    19 1
www.fbi.gov:80    08    1
www.fbi.gov:80    18    1
www.fbi.gov:80    21    1
www.google.com:443    00        1
www.google.com:443    21        1
```

Implementation: MapperClass, DriverClass, ReducerClass.

Conclusion:

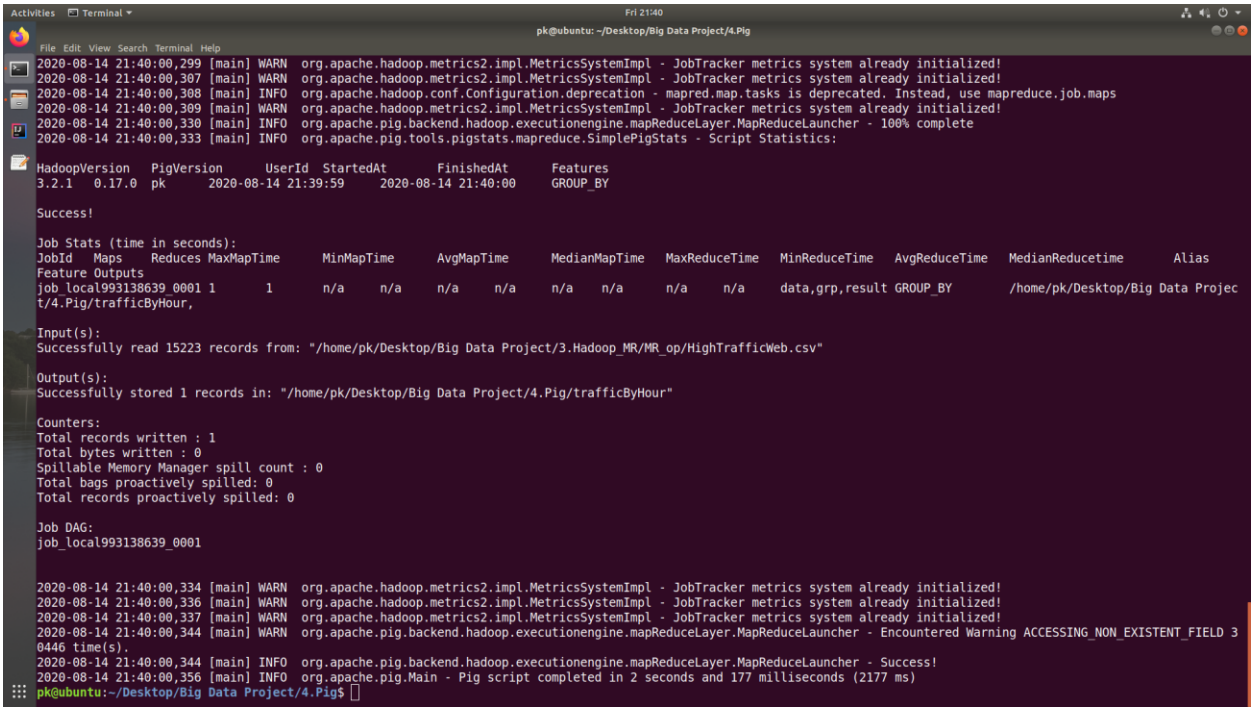
- For each resource, number of time that resource is accessed by user per hour is calculated.
- This analysis can be useful to maintain the traffic load for the web resource.

<u>Analysis</u> 7: Maximum traffic per hour at resource.

To achieve this goal, I used pig script to perform complex data transformation. Input given to this script is the output of Analysis 6 that is the output of mapreduce performed on Hadoop.

Methods used: Pig **Group By, Aggregate (MAX)**

Step 1: Load the input file
Step 2: Group all the data using **Group all** then it will form the tuple.
Step 3: Use FOREACH to generate required values, use MAX() to store maximum value.
Step 4: Store the result into the file.



Input format: Resource, hour, count
//phpMyAdmin-2.5.4/index.php,11,3
//phpMyAdmin-2.5.4/index.php,12,2

Output format: { (www.google.com:443,21,1), (www.google.com:443,0,1)…}

Conclusion: So, this analysis is performed to find out maximum number of times resource is accessed per hour.

- This analysis can be used to analyze traffic load per hour and it is useful to optimized the system.

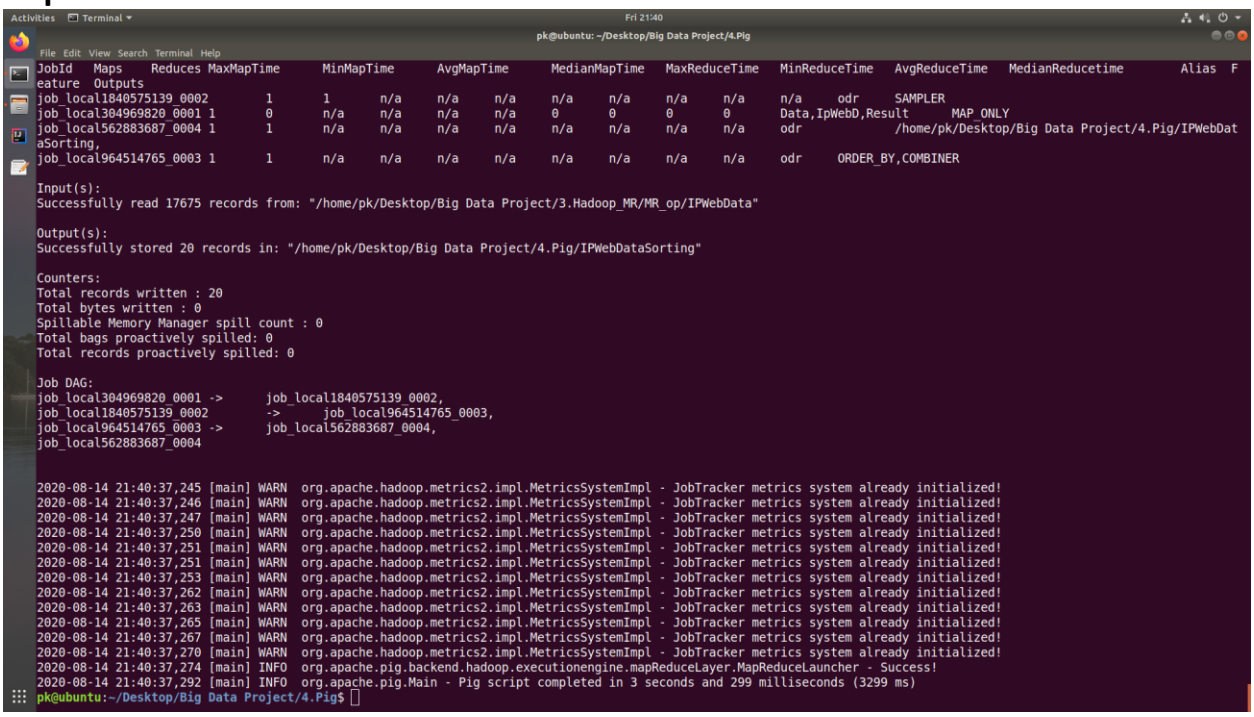
<u>Analysis</u> 8: Top 100 data consumed IP addresses.

To perform sorting of emitted values in Hadoop map-reduce is complex to code. This can be achieved in pig by writing small code.

Methods used: Pig **FILTER, ORDER BY, LIMIT**

Step 1: Load the input file
Step 2: Use FOREACH to generate required values.
Step 3: Filter data by giving size condition.
Step 4: Use ORDER BY on the values with DESC parameter
Step 5: Apply LIMIT to specify the number of outputs required.

Implementation:



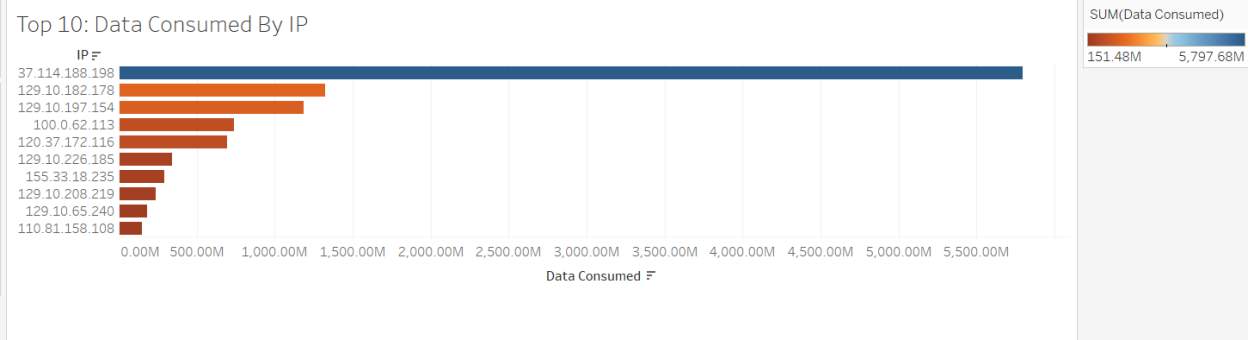
Input format: IP, Resource, Total Data consumed

Output Format: IP, Resource, Max. data consumed
37.114.188.198 /sqlserver/sqlserver.part1.rar 1502707021
(Sorting is performed in Descending order)

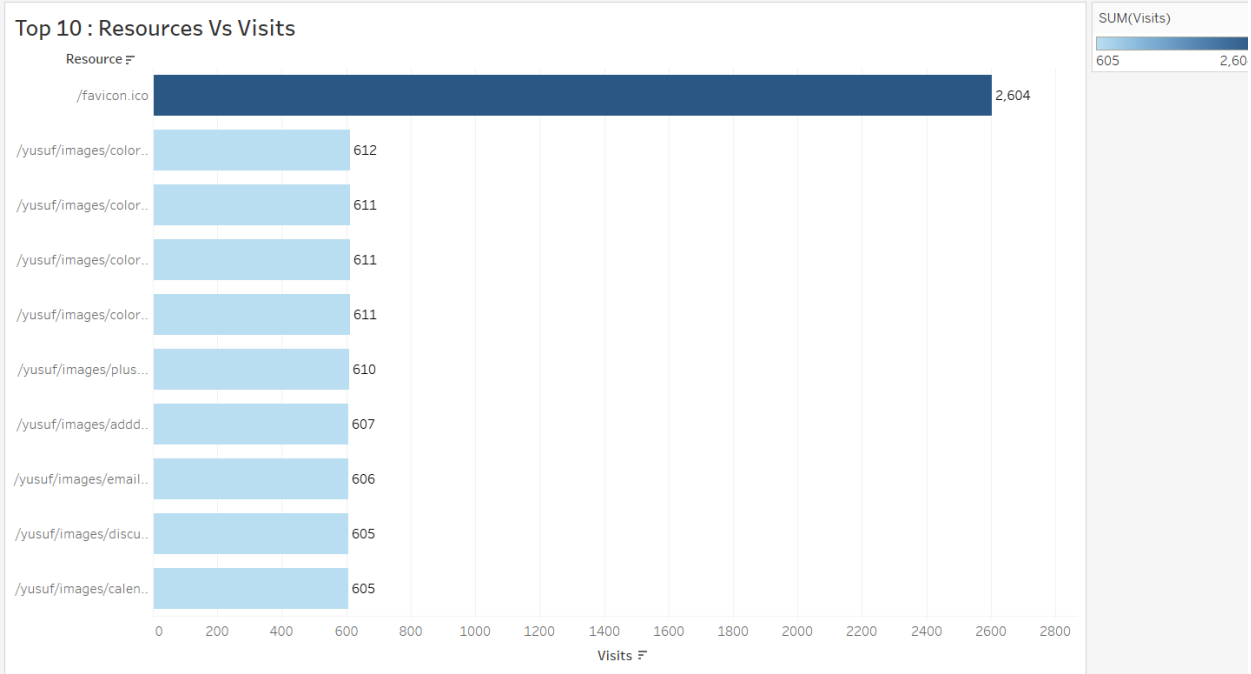
Visualization

- Data Visualization is performed with the help of **Tableau**.

1. **Top 10 highest data consumed IPs:**
   Input: Output of analysis 4.



2. **Top 10 Resources getting maximum of visits.**
   Input: Output of analysis 6.



Codes

MongoDB MapReduce:

1. Most Active IP

Mapper:

```
var map = function(){
emit(this.IP,1);
};
```

Reducer:

```
var reduce = function(key,values){
var count = 0;
values.forEach(function(i){
count +=i;
});
return count;
};
```

Cmd: db.AccessLogs.mapReduce(map,reduce, {out: "MostActiveIP"})

2. Most Demanded Resource

Mapper:

```
var map = function(){
emit(this.Resource,1);
};
```

Reducer:

```
var reduce = function(key,values){
var cnt = 0;
values.forEach(function(i){
cnt +=i;
});
return {"Count": cnt };
};
```

```
db.AccessLogs.mapReduce(map,reduce, {out: "MostDemandedResource"})
```

Hadoop MapReduce:

1. Analysis 4: (Traffic Load)

Driver class:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class DriverClass {
  public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {

    Configuration conf = new Configuration();
    Job = Job.getInstance(conf, "DataConsume");

    job.setJarByClass(DriverClass.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);

    job.setMapperClass(MapperClass.class);
    job.setReducerClass(ReducerClass.class);

    job.setOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);

    System.exit(job.waitForCompletion(true) ? 0:1);


  }
}
```

Mapper Class:

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class MapperClass extends Mapper<LongWritable, Text, Text, LongWritable> {
   Text ip = new Text();
   LongWritable data = new LongWritable();

   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String line = value.toString();
      String [] Tokens = line.split(",");
      int len = Tokens.length;
      //127.0.0.1,15/Oct/2011:11:49:11,-400,GET,/,HTTP/1.1,200,44

      if(Tokens[len-1].trim().matches("-?\\d+")){
         data.set(Long.parseLong(Tokens[len-1]));
      }
      else {
         data.set(0);
      }
      ip.set(Tokens[0]);
      context.write(ip,data);
      // Key: IP , Values: bytes of data
   }
}
```

Reducer Class:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
public class ReducerClass extends Reducer<Text, LongWritable,Text,LongWritable> {
   LongWritable result = new LongWritable();
   @Override
   protected void reduce(Text key, Iterable<LongWritable> values, Context context) throws
IOException, InterruptedException {
      long Total = 0;
      for (LongWritable val: values) {
         Total += val.get();
      }
      result.set(Total);
      context.write(key,result);
   }
}
```

2. Analysis 5: (IPWebData)

Driver class:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class DriverClass {

  public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
    Configuration config = new Configuration();
    Job = Job.getInstance(config, "DataFormating");

    job.setJarByClass(DriverClass.class);

    job.setGroupingComparatorClass(NaturalGroupingKeyComparator.class);
    job.setSortComparatorClass(CompositeKeyComparator.class);
    job.setPartitionerClass(NaturalKeyPartitioner.class);

    job.setMapOutputKeyClass(CompositeKeyWritable.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(MapperClass.class);
    job.setReducerClass(ReducerClass.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    Path outDir = new Path(args[1]);
    FileOutputFormat.setOutputPath(job, outDir);

    job.setNumReduceTasks(1);

    FileSystem fs = FileSystem.get(job.getConfiguration());
    if(fs.exists(outDir)){
      fs.delete(outDir, true);
    }
```

```
        job.waitForCompletion(true);

    }
}
```

Mapper Class:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MapperClass extends Mapper<LongWritable, Text, CompositeKeyWritable, IntWritable> {

    IntWritable data = new IntWritable();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

        // 127.0.0.1,15/Oct/2011:11:49:11,-400,GET,/,HTTP/1.1,200,44

        String line = value.toString();
        String[] tokens = line.split(",");

        Integer len = tokens.length;
        String Ip = null;
        String Web = null;

        //String[] timestamp = tokens[1].split(":");
        //String date=null;

        try {
            if (tokens.length == 8) {
                if(tokens[len-1].trim().matches("-?\\d+")){
                    Ip = tokens[0];
                    Web = tokens[4];
                    data.set(Integer.parseInt(tokens[len-1]));
                }
                else {
                    Ip = "Unknown";
                    Web = "N/A";
                    data.set(0);
                }
            } else{
                Ip = "Unknown";
                Web = "N/A";
                data.set(0);
            }
```

```
        } catch (Exception ex) {
            //Do Nothing
        }

        CompositeKeyWritable obj = new CompositeKeyWritable(Ip, Web);

        context.write(obj, data);
    }
}
```

Reducer Class:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<CompositeKeyWritable, IntWritable, Text, IntWritable> {

    IntWritable result = new IntWritable();
    Text text = new Text();
    @Override
    protected void reduce(CompositeKeyWritable key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable val : values){
            sum += val.get();
        }
        text.set(key.getIp() + "\t" + key.getWeb());
        result.set(sum);
        context.write(text, result);
    }
}
```

CompositeKeyComparator Class:

```
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import java.awt.*;

public class CompositeKeyComparator extends WritableComparator {

    public CompositeKeyComparator(){
        super(CompositeKeyWritable.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        CompositeKeyWritable ckw1 = (CompositeKeyWritable) a;
```

```
        CompositeKeyWritable ckw2 = (CompositeKeyWritable) b;

        int result = -1 * ckw1.getWeb().compareTo(ckw2.getWeb());
        return result;
    }
}
```

CompositeKeyWritable Class:

```
import org.apache.hadoop.io.WritableComparable;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class CompositeKeyWritable implements WritableComparable<CompositeKeyWritable> {

    String Ip;
    String Web;

    public CompositeKeyWritable(){
    }

    public CompositeKeyWritable(String Ip, String Web) {
        this.Ip = Ip;
        this.Web = Web;
    }

    public String getIp() {
        return Ip;
    }

    public void setIp(String Ip) {
        this.Ip = Ip;
    }

    public String getWeb() {
        return Web;
    }

    public void setWeb(String Web) {
        this.Web = Web;
    }

    public void readFields(DataInput in) throws IOException {
        Ip = in.readUTF();
        Web = in.readUTF();
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(Ip);
        out.writeUTF(Web);
```

```
    }

    public int compareTo(CompositeKeyWritable o) {
        int result = this.Web.compareTo(o.getWeb());
        return (result < 0 ? -1 : (result == 0 ? 0 : 1));
    }

    @Override
    public String toString() {
        return Ip + " \t " + Web;
    }
}
```

NaturalGroupingKeyComparator class:

```
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class NaturalGroupingKeyComparator extends WritableComparator {

    public NaturalGroupingKeyComparator(){
        super(CompositeKeyWritable.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        CompositeKeyWritable ckw1 = (CompositeKeyWritable) a;
        CompositeKeyWritable ckw2 = (CompositeKeyWritable) b;

        int result = ckw1.getIp().compareTo(ckw2.getIp());
        return result;
    }
}
```

NaturalKeyPartitioner class:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class NaturalKeyPartitioner extends Partitioner<CompositeKeyWritable, IntWritable> {

    @Override
    public int getPartition(CompositeKeyWritable key, IntWritable value, int noOfPartitions) {
        return key.getIp().hashCode() % noOfPartitions;
    }
}
```

3. Analysis 6: (HighTrafficWeb)

Driver class:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class DriverClass {
   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {

      Configuration conf = new Configuration();
      Job = Job.getInstance(conf, "Peak Hours");

      job.setJarByClass(DriverClass.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      job.setMapOutputKeyClass(Text.class);
      job.setMapOutputValueClass(IntWritable.class);

      job.setMapperClass(MapperClass.class);
      job.setReducerClass(ReducerClass.class);

      job.setOutputKeyClass(Text.class);
      job.setMapOutputValueClass(IntWritable.class);

      System.exit(job.waitForCompletion(true) ? 0:1);


   }
}
```

Mapper Class:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {
   Text Web = new Text();
   IntWritable one = new IntWritable(1);
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String line = value.toString();
      String [] Tokens = line.split(",");
      //127.0.0.1,15/Oct/2011:11:49:11,-0400,GET,/,HTTP/1.1,200,44
      // Key: web , hour ; val: count
      String [] timestamp = Tokens[1].split(":");
      String hour = timestamp[1];
      String input = Tokens[4] +"    " +  hour;
      Web.set(input);
      context.write(Web,one);
   }
}
```

Reducer Class:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable,Text,IntWritable> {

   IntWritable result = new IntWritable();

   @Override
   protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
      int count = 0;

      for (IntWritable val: values) {
         count += val.get();
      }
      result.set(count);

      context.write(key,result);
   }
}
```

PIG SCRIPTS

Analysis 7: Maximum traffic per hour at resource.

File: maxTrafficHour.pig
Execution: pig -x local maxTrafficHour.pig

Script:

```
data = LOAD '/home/pk/Desktop/Big Data Project/3.Hadoop_MR/MR_op/HighTrafficWeb.csv' AS (Resource:chararray,hour:chararray, count:int);

grp = GROUP data ALL;

result = FOREACH grp GENERATE (data.Resource, data.hour), MAX(data.count) as cnt;

Store result into '/home/pk/Desktop/Big Data Project/4.Pig/trafficByHour';
```

Analysis 8: Top 100 data consumed IP addresses.

File: IPWebDataSort.pig
Execution: pig -x local IPWebDataSort.pig

Script:

```
Data = LOAD '/home/pk/Desktop/Big Data Project/3.Hadoop_MR/MR_op/IPWebData';

IpWebD = FOREACH Data GENERATE (chararray) $0 as IP, (chararray) $1 as Resource, (int) $2 as Data_size;

Result = FILTER IpWebD BY Data_size > 51200;

odr = ORDER Result BY Data_size DESC;

lmt = LIMIT odr 100;

Store lmt into '/home/pk/Desktop/Big Data Project/4.Pig/IPWebDataSorting';
```