Analyzing the frequently viewed videos from a YouTube log dataset using Apache Hive

Samirana Aacharya[1], Bamrah Jagjit Kaur[2], Bandari Sharath Chandra[3], B. Vijaya Lakshmi[4]

Assistant Professor [1], B.tech (pursuing)[2,3,4], GNITC, Department Of CSE[1,2,3,4]

[1]acharya501@gmail.com, [2]jagjitkaur456@gmail.com, [3]sharath.bandari25@gmail.com, [4] vijayalakshmi.sonu999@gmail.com

ABSTRACT

Opinion mining, which extracts meaningful opinion information from large amounts of social multimedia data, has recently arisen as a research area. In particular, opinion mining has been used to understand the true meaning and intent of social networking site users. It requires efficient techniques to collect a large amount of social multimedia data and extract meaningful information from them. Therefore, in this paper, we propose a method to extract sentiment information from various types of unstructured social media text data from social networks by using a parallel Hadoop Distributed File System (HDFS) to save social multimedia data and using MapReduce functions for sentiment analysis. The proposed method has stably performed data gathering and data loading and maintained stable load balancing of memory and CPU resources during data processing by the HDFS system. The proposed MapReduce functions have effectively performed sentiment analysis in the experiments. Finally, the sentiment analysis results of the proposed system are very close to those of manual processes.

1. INTRODUCTION

In this we will be discussing the concept of analysing YouTube videos using Map Reduce and Hive, also how hive can be used in place of Map reduce. At times, there are situations where you have to manipulate your java coding which can make the program more complex. So, in such situations we can use Map Reduce. In this paper, we are going to analyse a YouTube log data set and obtain the list of top 10 videos based upon the rating. Big data has lately gained more popularity and there's still much more to discover in it. In Relational Database System in order to extract large amount of data it takes large amount of time and complexity arises that's why we use Map reduce and hive. In our project first we will convert java code and Hadoop library files into a .jar file. Then the given YouTube log data set will be exported to HDFS. There after executing the query the output will be stored in HDFS. The output contains n number of video names along with the average of their rating. Now we start hive to obtain a list of top 10 videos.

2. EXISTING SYSTEM

In the existing technology, we used traditional enterprise systems to analyze, store and transport the data. In Relational Database System in order to extract large amount of data it takes large amount of time and complexity arises. In the existing technology, we used traditional enterprise systems to analyze, store and transport the data. These databases usually have a centralized server to store and process data. Moreover, centralized system creates too much of a bottleneck (i.e. a bottleneck is one process in a chain of processes, such that its limited capacity reduces the capacity of the whole chain) while processing multiple files simultaneously. Google solved this issue by introducing an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to a cluster of computers. Later, the results are collected at one place and integrated to form the final result dataset.

DISADVANTAGES:

- Complexity is high
- Runtime for query processing is high

3. PROPOSED SYSTEM

MAP REDUCE

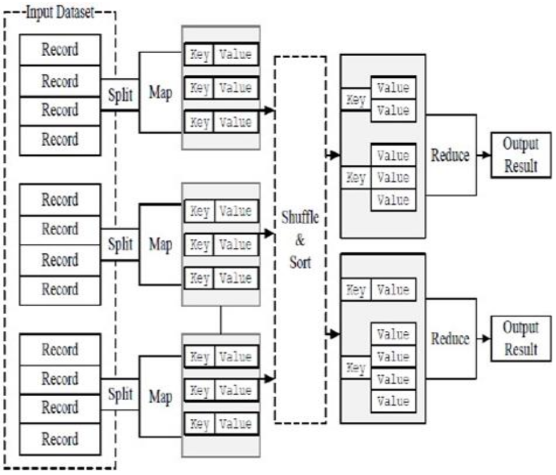
MapReduce is a programming model and software framework for distributed processing originally developed by google in 20004 proposed to facilitate and reduce the processing of vast amounts of data on enormous clusters of commodity hardware in a reliable, fault-tolerant manner. The job of MapReduce is to split the input data set into separate blocks which are taken care by map tasks in a parallel way. The framework sorts the outputs of maps, which acts as input to the reduce tasks. Typically, both the input and output of the job are stored in a file-system. The framework has certain characteristics like scheduling tasks, monitoring them and re-executes the failed tasks.

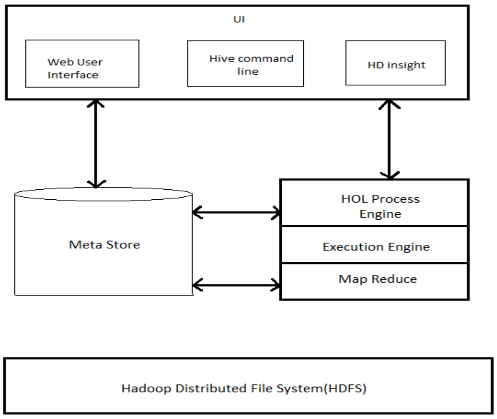
The MapReduce acts as a master slave architecture same as in HDFS. There are two types of nodes in MapReduce, they are Task-tracker and Job-tracker. Task-tracker acts as master node and Job-tracker as slave. The task-tracker splits the whole program into number of individual program and assign it to the workers. The worker compute each program individually and transfers the results back to task-tracker, Job-tracker runs with name node, receives the users job, decides on how many tasks will run (number of mappers) and decides on where to run each mapper by considering its location. Whereas Master pings workers periodically to detect failures.

The MapReduce algorithm consists of two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The reduce task is to take output from the map as an input and combines those data tuples (key-value pair) into a smaller set of tuples.

The reduce task is always performed after the map job





HIVE

Hive is an abstraction of MapReduce. It uses its own query language HQL (Hive Query Language), it is similar to SQL. Hive can be used instead of MapReduce. MapReduce uses java coding which can be sometimes complicated and confusing so instead of writing those n lines of java code we can write single line query using hive. Generally, in hive it stores metadata in an embedded apache then it is stored in database, and also some other client/server databases such as MySQL can also use TEXTFILE, SEQUENCEFILE, ORC and RCFILE are the four file formats sustained by hive. Hive is mainly designed of traditional data warehousing tasks not for online transaction processing. Hive is designed for Online Analytical Processing (OLAP). In hive schema is stored in a database and data is processed in HDFS. HQL is provided by SQL type language for querying, and also familiar, fast, scalable, and extensible.

This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	The interaction between user and HDFS can be created by using data warehouse infrastructure software known as hive. Hive Web UI, Hive command line, and Hive HD Insight are the user interfaces supported by hive.
Meta Store	Metadata or schema of tables, databases, columns in a table, their data types, and HDFS mapping are stored in the database servers respectively which are choose by hive.
HiveQL Process Engine	HQL and SQL are similar for querying on schema info on the Megastore. It can be replaced traditionally for MapReduce program. In place of using MapReduce program java, we can

	use a query for MapReduce job and process it
Execution Engine	Hive Execution Engine is the conjunction part of HQL process Engine and MapReduce. The result generated by Execution engine by processing the query and query is similar to MapReduce.
HDFS or HBASE	For storing data into file system, HBASE or Hadoop distributed file system are the techniques used.

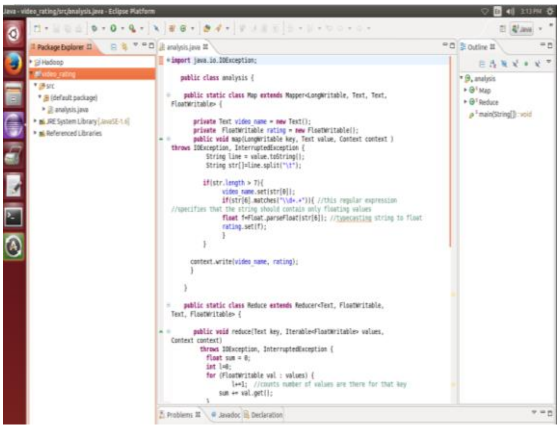
4. RESULTS AND ANALYSIS

RESULTS

All Hadoop sub-projects such as Map Reduce Hive, Pig, and HBase support Linux operating system. Therefore, you need to install any Linux flavored OS.

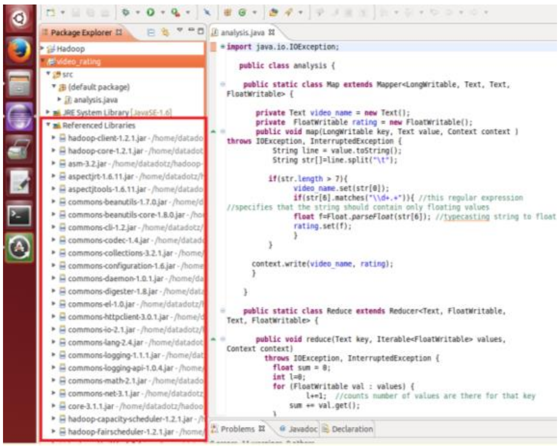
SNAPSHOTS

Creating a java project



```
import java.io.IOException;

  public class analysis {

   public static class Map extends Mapper<LongWritable, Text, Text,
FloatWritable> {

      private Text video_name = new Text();
      private  FloatWritable rating = new FloatWritable();
      public void map(LongWritable key, Text value, Context context )
throws IOException, InterruptedException {
          String line = value.toString();
          String str[]=line.split("\t");

         if(str.length > 7){
              video_name.set(str[0]);
              if(str[6].matches("\\d+.+")){ //this regular expression
//specifies that the string should contain only floating values
              float f=Float.parseFloat(str[6]); //typecasting string to float
              rating.set(f);
              }
         }

     context.write(video_name, rating);
     }

   }

   public static class Reduce extends Reducer<Text, FloatWritable,
Text, FloatWritable> {

      public void reduce(Text key, Iterable<FloatWritable> values,
Context context)
        throws IOException, InterruptedException {
          float sum = 0;
          int l=0;
          for (FloatWritable val : values) {
                l+=1;  //counts number of values are there for that key
             sum += val.get();
```

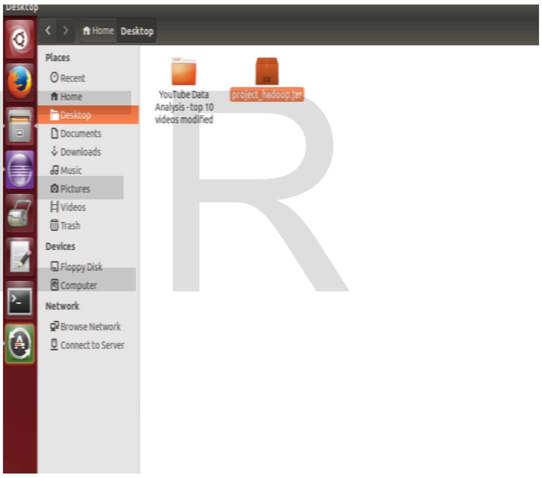
Add Hadoop Library files



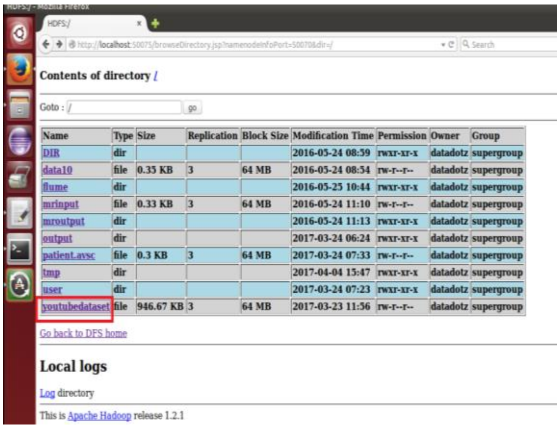
Package Explorer
Hadoop
video_rating
src
(default package)
analysis.java
JRE System Library [JavaSE-1.6]
Referenced Libraries
hadoop-client-1.2.1.jar - /home/datado
hadoop-core-1.2.1.jar - /home/datadot
asm-3.2.jar - /home/datadotz/hadoop-
aspectjrt-1.6.11.jar - /home/datadotz/
aspectjtools-1.6.11.jar - /home/datado
commons-beanutils-1.7.0.jar - /home/d
commons-beanutils-core-1.8.0.jar - /ho
commons-cli-1.2.jar - /home/datadotz/
commons-codec-1.4.jar - /home/datado
commons-collections-3.2.1.jar - /home/
commons-configuration-1.6.jar - /home
commons-daemon-1.0.1.jar - /home/da
commons-digester-1.8.jar - /home/dat
commons-el-1.0.jar - /home/datadotz/
commons-httpclient-3.0.1.jar - /home/
commons-io-2.1.jar - /home/datadotz/
commons-lang-2.4.jar - /home/datadot
commons-logging-1.1.1.jar - /home/dat
commons-logging-api-1.0.4.jar - /home/
commons-math-2.1.jar - /home/datado
commons-net-3.1.jar - /home/datadotz
core-3.1.1.jar - /home/datadotz/hadoo
hadoop-capacity-scheduler-1.2.1.jar - /
hadoop-fairscheduler-1.2.1.jar - /home/
analysis.java
Problems
Javadoc
Declaration

```
import java.io.IOException;

  public class analysis {

   public static class Map extends Mapper<LongWritable, Text, Text,
FloatWritable> {

      private Text video_name = new Text();
      private  FloatWritable rating = new FloatWritable();
      public void map(LongWritable key, Text value, Context context )
throws IOException, InterruptedException {
          String line = value.toString();
          String str[]=line.split("\t");

         if(str.length > 7){
              video_name.set(str[0]);
              if(str[6].matches("\\d+.+")){ //this regular expression
//specifies that the string should contain only floating values
              float f=Float.parseFloat(str[6]); //typecasting string to float
              rating.set(f);
              }
         }

     context.write(video_name, rating);
     }

   }

   public static class Reduce extends Reducer<Text, FloatWritable,
Text, FloatWritable> {

      public void reduce(Text key, Iterable<FloatWritable> values,
Context context)
        throws IOException, InterruptedException {
          float sum = 0;
          int l=0;
          for (FloatWritable val : values) {
                l+=1;  //counts number of values are there for that key
             sum += val.get();
```

Converting java code and Hadoop library files into a jar file



Moving Dataset to HDFS



HDFS:/ - Mozilla Firefox
HDFS:/
http://localhost:50075/browseDirectory.jsp?namenodeInfoPort=50070&dir=/
Search

Contents of directory /

Goto : / go

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
DIR	dir				2016-05-24 08:59	rwxr-xr-x	datadotz	supergroup
data10	file	0.35 KB	3	64 MB	2016-05-24 08:54	rw-r--r--	datadotz	supergroup
flume	dir				2016-05-25 10:44	rwxr-xr-x	datadotz	supergroup
mrinput	file	0.33 KB	3	64 MB	2016-05-24 11:10	rw-r--r--	datadotz	supergroup
mroutput	dir				2016-05-24 11:13	rwxr-xr-x	datadotz	supergroup
output	dir				2017-03-24 06:24	rwxr-xr-x	datadotz	supergroup
patient.avsc	file	0.3 KB	3	64 MB	2017-03-24 07:33	rw-r--r--	datadotz	supergroup
tmp	dir				2017-04-04 15:47	rwxr-xr-x	datadotz	supergroup
user	dir				2017-03-24 07:23	rwxr-xr-x	datadotz	supergroup
youtubedataset	file	946.67 KB	3	64 MB	2017-03-23 11:56	rw-r--r--	datadotz	supergroup

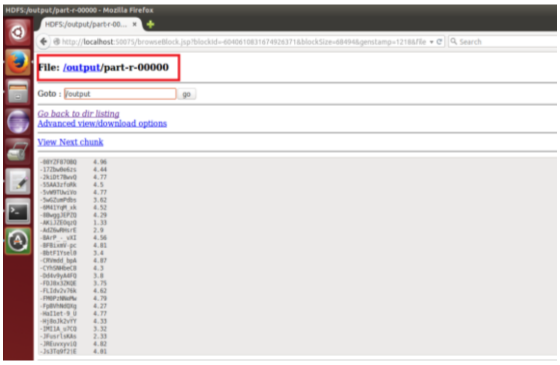
Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.2.1

Executing the command



HDFS:/output/part-r-00000 - Mozilla Firefox

HDFS:/output/part-r-00...

http://localhost:50075/browseBlock.jsp?blockId=-6040610831674926371&blockSize=68494&genstamp=1218&file

File: /output/part-r-00000

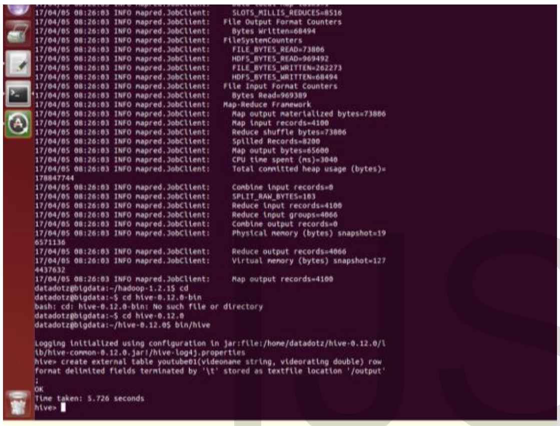
Goto : /output go

Go back to dir listing
Advanced view/download options

View Next chunk

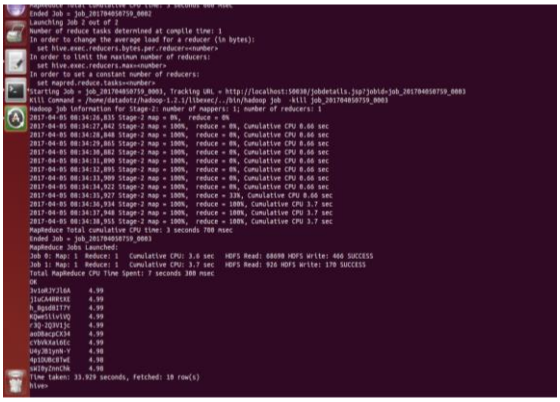
```
-08YZF870BQ    4.96
-17Zbw0e6zs    4.44
-2kiDt78wvQ    4.77
-55AA3zfoRk    4.5
-5vW9TUwiVo    4.77
-5wGZumPdbs    3.62
-6M41YqM_xk    4.52
-8BwggJEPZQ    4.29
-AKiJZEOqzQ    1.33
-AdZ6wRHsrE    2.9
-BArP_-_vXI    4.56
-BFBixmV-pc    4.81
-BbtF1Ysel0    3.4
-CRVmdd_bpA    4.87
-CYhSNHbeC8    4.3
-Dd4v9yA4FQ    3.8
-FDJ8x3ZKQE    3.75
-FLIdv2v76k    4.62
-FM0PzNNwMw    4.79
-FpBVhNdQXg    4.27
-HaIlet-9_U    4.77
-Hj8oJk2vYY    4.33
-IMI1A_u7CQ    3.32
-JFusrlsKAs    2.33
-JREuvxyviQ    4.82
-Js3Tq9f21E    4.81
```

Creating table in Hive



```
17/04/05 08:26:03 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=8516
17/04/05 08:26:03 INFO mapred.JobClient:   File Output Format Counters
17/04/05 08:26:03 INFO mapred.JobClient:     Bytes Written=68494
17/04/05 08:26:03 INFO mapred.JobClient:   FileSystemCounters
17/04/05 08:26:03 INFO mapred.JobClient:     FILE_BYTES_READ=73806
17/04/05 08:26:03 INFO mapred.JobClient:     HDFS_BYTES_READ=969492
17/04/05 08:26:03 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=262273
17/04/05 08:26:03 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=68494
17/04/05 08:26:03 INFO mapred.JobClient:   File Input Format Counters
17/04/05 08:26:03 INFO mapred.JobClient:     Bytes Read=969389
17/04/05 08:26:03 INFO mapred.JobClient:   Map-Reduce Framework
17/04/05 08:26:03 INFO mapred.JobClient:     Map output materialized bytes=73806
17/04/05 08:26:03 INFO mapred.JobClient:     Map input records=4100
17/04/05 08:26:03 INFO mapred.JobClient:     Reduce shuffle bytes=73806
17/04/05 08:26:03 INFO mapred.JobClient:     Spilled Records=8200
17/04/05 08:26:03 INFO mapred.JobClient:     Map output bytes=65600
17/04/05 08:26:03 INFO mapred.JobClient:     CPU time spent (ms)=3040
17/04/05 08:26:03 INFO mapred.JobClient:     Total committed heap usage (bytes)=
178847744
17/04/05 08:26:03 INFO mapred.JobClient:     Combine input records=0
17/04/05 08:26:03 INFO mapred.JobClient:     SPLIT_RAW_BYTES=103
17/04/05 08:26:03 INFO mapred.JobClient:     Reduce input records=4100
17/04/05 08:26:03 INFO mapred.JobClient:     Reduce input groups=4066
17/04/05 08:26:03 INFO mapred.JobClient:     Combine output records=0
17/04/05 08:26:03 INFO mapred.JobClient:     Physical memory (bytes) snapshot=19
6571136
17/04/05 08:26:03 INFO mapred.JobClient:     Reduce output records=4066
17/04/05 08:26:03 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=127
4437632
17/04/05 08:26:03 INFO mapred.JobClient:     Map output records=4100
datadotz@bigdata:~/hadoop-1.2.1$ cd
datadotz@bigdata:~$ cd hive-0.12.0-bin
bash: cd: hive-0.12.0-bin: No such file or directory
datadotz@bigdata:~$ cd hive-0.12.0
datadotz@bigdata:~/hive-0.12.0$ bin/hive

Logging initialized using configuration in jar:file:/home/datadotz/hive-0.12.0/l
ib/hive-common-0.12.0.jar!/hive-log4j.properties
hive> create external table youtube01(videoname string, videorating double) row
format delimited fields terminated by '\t' stored as textfile location '/output'
;
OK
Time taken: 5.726 seconds
hive> 
```

Obtaing list of top 10 videos



```
Ended Job = job_201704050759_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201704050759_0003, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201704050759_0003
Kill Command = /home/datadotz/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_201704050759_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2017-04-05 08:34:26,835 Stage-2 map = 0%,  reduce = 0%
2017-04-05 08:34:27,842 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:28,848 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:29,865 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:30,882 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:31,890 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:32,895 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:33,909 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:34,922 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.66 sec
2017-04-05 08:34:35,927 Stage-2 map = 100%,  reduce = 33%, Cumulative CPU 0.66 sec
2017-04-05 08:34:36,934 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.7 sec
2017-04-05 08:34:37,948 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.7 sec
2017-04-05 08:34:38,955 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.7 sec
MapReduce Total cumulative CPU time: 3 seconds 700 msec
Ended Job = job_201704050759_0003
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 3.6 sec   HDFS Read: 68690 HDFS Write: 466 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 3.7 sec   HDFS Read: 926 HDFS Write: 170 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 300 msec
OK
3v1oRJYJl6A     4.99
j1uCA4RRtXE     4.99
h_8gsd8IT7Y     4.99
KQweSiiviVQ     4.99
r3Q-2Q3Vijc     4.99
aoDBacpCX34     4.99
cYbVkXai6Ec     4.99
U4yJB1ynN-Y     4.98
4p1DUBc8TwE     4.98
sWI0yZnnChk     4.98
Time taken: 33.929 seconds, Fetched: 10 row(s)
hive>
```

ANALYSIS

So finally after execution of code, we get a list of top 10 frquently viewed videos.

VIDEO NAME	RATING
3v1ORJYJLl6A	4.99
j1uCA4RRtXE	4.99
h_8gsd8IT7Y	4.99
KQweSiiviVQ	4.99
r3Q-2Q3Vijc	4.99
aoDBacpCX34	4.99
cybvkXai6EC	4.99
u4yJB1ynN-Y	4.99
4p1DUBc8TwE	4.99
SWI0yZnnchk	4.99

5. FUTURE ENHANCEMENT

Java coding is not an easy task. In some cases for modifying the code for further execution either using java code it is an easy way to choose hive or pig, with this we give a single line query to get result. For instance, 50 lines of java code can be written in 4-5 lines by hive. Using hive time and complexity is reduced. Major disadvantage of MapReduce is high latency with that it is unusable for real time applications. There can be complicates to implement everything as MapReduce program. MR is not suitable for a large number of short on-line transactions, when we have OLTP needs. Implementation of interactive jobs and modals is impossible because MapReduce is only suitable for the batch processing jobs. Due to more space consumption for each job, the implementation of the MapReduce jobs becomes expensive. The MapReduce do not support the interaction between the intermediate processes, that means the job is isolated.

6. CONCLUSION

Big Data even though being in early stages, it has got a huge effect on the technological companies and the way the companies do the business. A new category of data storage and analysis systems with different architectures may be required for the exploitation as suggested by the datasets. In Big Data community Hadoop-MapReduce programming paradigm have a substantial base due to the presence of cost-effectiveness on commodity Linux clusters. The involvement of many data analysis algorithms made the map reduce method effective and ease of use in parallelization. HDFS, the Distributed File System which is designed to hold huge amount of data and provide high-throughput to access to this information. Hadoop with HDFS provides a very good way of handling faults tolerance, despite of cons of failure and breakdown of name node.

REFERENCE

1. Bittencourt, L.F. and Madeira, E.R.M. “A Performance-Oriented Adaptive Scheduler for Dependent Tasks on Grids,” Concurrency and Computation: Practice and Experience.
2. Caron, E. Chis, A. Desprez, F. And Su, A. “Design of Plug-in Schedulers for a GRIDRPC Environment,” Future Generation Computer Systems, vol. 24, no. 1, pp. 46-57.
3. Dinda, P.A. And O’Hallaron, D.R. “Host Load Prediction Using Linear Models,” Cluster Computing, vol. 3, no. 4, pp. 265-280.
4. Dinda, P.A. “Design, Implementation, and Performance of an Extensible Toolkit for Resource Prediction in Distributed Systems,” IEEE Trans. Parallel and Distributed Systems, vol. 17, no. 2, b pp. 160-173.
5. Eddy Caron, Andreea Chis, Frederic Desprez, Alan Su (November 2011) **“**Plug-in Scheduler Design for a Distributed Grid Environment”.
6. Liang Hu, Xi-Long Che, (2012)”Online System for Grid Resource Monitoring and Machine Learning-Based Prediction” IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23.
7. Massie, M.L. Chun, B.N. And Culler, D.E. “The Ganglia Distributed Monitoring System: Design, Implementation, and Experience,” Parallel Computing, vol. 30, no. 7, pp. 817-840.
8. Peter Dinda, A. and David R. O’Halloran (July 2012) “AN Extensible Toolkit for Resource Prediction in Distributed Systems” School of Computer Science Carnegie Mellon University Pittsburgh, PA, 15213.
9. Sam Verboven, Peter Hellinckx, Frans Arickx and Jan Broeckhove (2011) “Runtime Prediction based Grid

Scheduling of Parameter Sweep Jobs" University of Antwerp Antwerp, Belgium.

10. Sodan, A.C. Gupta, G. Han, L. Liu, L. and Lafreniere, B. "Time and Space Adaptation for Computational Grids with the ATOPGrid Middleware," Future Generation Computer Systems, vol. 24, no. 6, pp. 561-581.

11. Swany, D.M. and Wolski, R. "Multivariate Resource Performance Forecasting in the Network Weather Service," Proc. ACM/IEEE Conf. Supercomputing, pp. 1-10.

12. Vetter, J.S. and Reed, D.A. "Real-Time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids," Int'l J. High Performance Computing Applications, vol. 14. no. 4, pp. 357-366.

13. Waheed et al., "An Infrastructure for Monitoring and Management in Computational Grids," Proc. Fifth Int'l Workshop Languages, Compilers and Run-Time Systems for Scalable Computers, vol. 1915, pp. 235-245.