CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

CLOUD COMPUTING

Cloud Computing is a recent advancement wherein IT infrastructure and applications are provided as 'services' to end-users under a usage-based payment model. It can leverage virtualized services even on the fly based on requirements (workload patterns and QoS) varying with time. The application services hosted under Cloud computing model have complex provisioning, composition, configuration, and deployment requirements. Evaluating the performance of Cloud provisioning policies, application workload models, and resources performance models in a repeatable manner under varying system and user configurations and requirements is difficult to achieve.

APACHE HADOOP

- Apache Hadoop is a set of algorithms for distributed storage and distributed processing of very large data sets (Big Data) on computer clusters built from commodity hardware.
- Apache Hadoop consists of a storage part (Hadoop Distributed File System HDFS) and a processing part (Map Reduce). Hadoop splits files into large blocks and distributes the blocks amongst the nodes in the cluster. To process the data, Hadoop Map/Reduce transfers code to nodes that have required the data, which the nodes then process in parallel.

APACHE GIRAPH

- Apache Giraph is an iterative graph processing system built for high scalability. It is currently used at Facebook to analyse the social graph formed by the users and their connections. It originated as the open source counterpart of Pregel (graph processing architecture developed at Google).
- It is inspired by the Bulk Synchronous Parallel model of distributed computation. It includes several features beyond the basic Pregel model, including master computation, sharded aggregators, edge oriented input, out-of core computation, and more. It can be used to deploy structured datasets at a massive scale.

CHAPTER 2

PROBLEM DEFINITION

2.1 Problem Definition

Apache Hadoop is a project that has already underwent extensive development and improvisation. The concept of Apache Giraph deployed on Apache Hadoop cluster is a relatively new concept. The project aims at extending the capabilities and functionalities that Apache Hadoop possesses by deploying Apache Giraph on its single node and multi node cluster. We therefore aim to deploy graphs on Apache Giraph on single node and multi node Hadoop cluster and analyze the output for the various graphs.

CHAPTER 3

LITERATURE SURVEY

3.1 Literature Survey

The paper [1] discusses the GoFFish design and architecture, discuss the synergistic subgraph centric storage and composition model that allow scalable analytics over space and time, and illustrate them using both graph algorithms (connected-components) and Big Data Applications (Internet traces, social networks) as exemplars.

The paper [2] presents a computational model suitable for the task of processing trillions of edges and billions of vertices which are prominent of Web Graphs and social networks. Programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. This vertex centric approach is flexible enough to express a broad set of algorithms. This model has been designed for efficient, scalable and fault-tolerant implementation on clusters of thousands of commodity computers, and its implied synchronicity makes reasoning about programs easier. Distribution related details are hidden behind an abstract API. The result is a framework for processing large graphs that is expressive and easy to program.

CHAPTER 4

PROJECT REQUIREMENT DEFINITION

4.1 Functional Requirements:

Code model for:

- Running Map Reduce Jobs on Single Node Hadoop Cluster.
- Deploying Apache Giraph on Single Node Hadoop Cluster.
- Running Giraph Jobs on Apache Giraph (Page Rank and Simple Source Shortest Paths Compute).
- Algorithm for conversion of Graph Input Format to suitable format.
- Running Large Graphs (Big Data Sets) on Apache Giraph on single node cluster.
- Deployed Apache Giraph on Multi Node Hadoop Cluster.
- Ran Giraph Jobs on Virtual Machines (Master Node and Slave Node) to replicate Server behavior.
- Ran Giraph Jobs on Server (Master and Slave) and track the tasks and performance of Master and Slave Nodes.

CHAPTER 5

SYSTEM REQUIREMENTS SPECIFICATION

5.1 Software System Requirement:

Eclipse:

Eclipse is an integrated development environment. It contains a base workspace and an extensible plug-in system for customizing the environment. It can also be used to develop packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

Unix Platform (UBUNTU):

Ubuntu 14.04 LTS is a Debian-based Linux operating system, with Unity as its default desktop environment (GNOME was the previous desktop environment).

Oracle Virtual Machine:

Oracle VM is the server virtualization offering from Oracle Corporation. Oracle VM Server for x86 incorporates the free and open source Xen hypervisor technology, supports Windows, Linux, and Oracle Solaris, guests and includes an integrated Web based management Console. Oracle VM features fully tested and certified Oracle Applications stack in an enterprise virtualization environment.

Java:

Java is a set of several computer software and specifications developed by Sun Microsystems, later acquired by Oracle Corporation that provides a system for developing application software and developing it in a cross-platform computing environment. Java is used in a wide variety of computing platforms from embedded devices and mobile phones to enterprise servers and supercomputers. While less common, Java applets run in secure, sandboxed environments to provide many features of native applications and can be embedded in HTML pages.

5.2 Hardware System Requirements

- CPU – 2.67 GHz Quad Core
- RAM – 8 GB (Running multiple instances of Virtual Machine).
- Hard Disk Space - <1 GB

CHAPTER 6

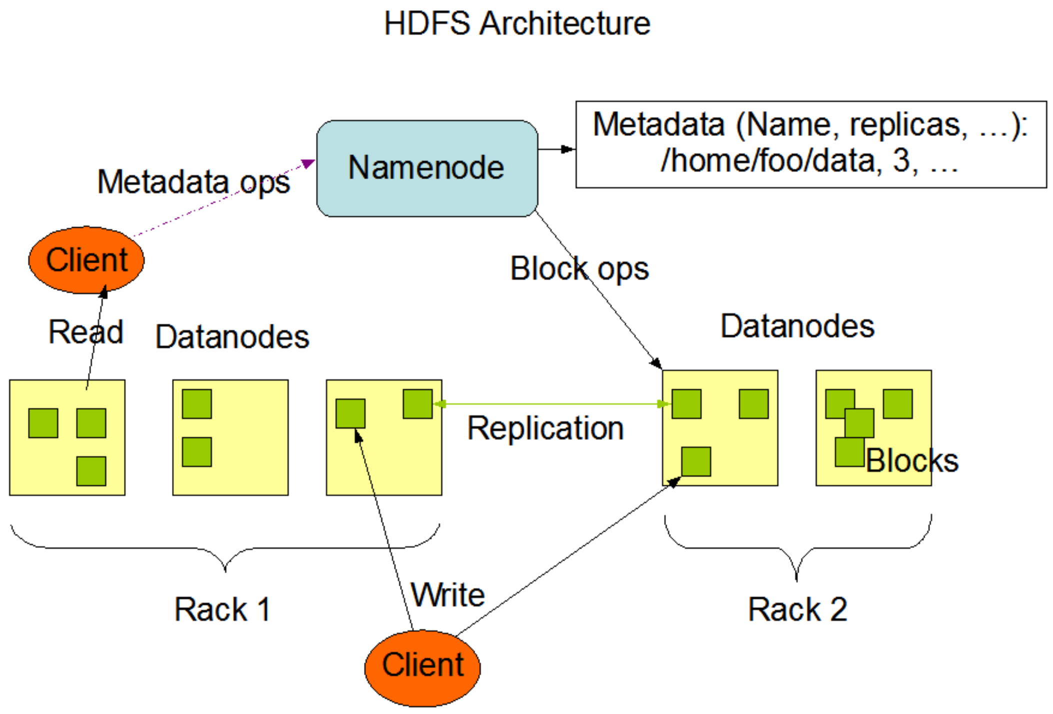
SYSTEM DESIGN

6.1 Apache Hadoop Single Node Cluster

Apache Hadoop has two components:

1) Storage Part - Hadoop Distributed File System (HDFS)
   The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to deploy on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. It is responsible for managing the namenode, secondary namenode and datanode.

2) Processing Part – MapReduce
   Hadoop MapReduce transfers packaged code for nodes to process in parallel, based on the data each node needs to process. It is a programming model for large scale data processing. It is responsible for managing the tasktracker and jobtracker.



6.2 JPS – Java Virtual Machine Process Status Tools

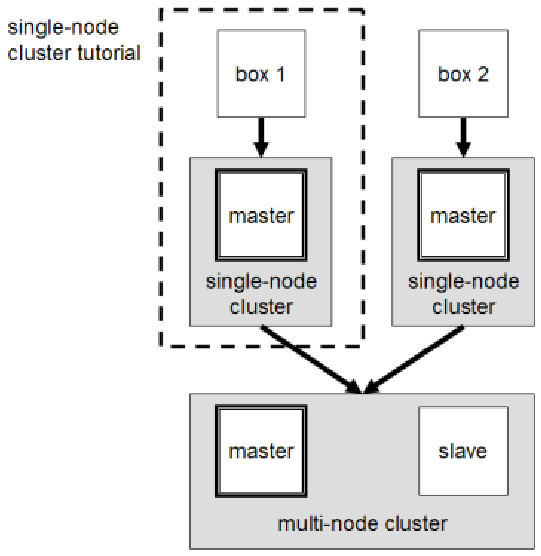
The JPS tools lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system. The tool is limited to reporting information on JVMs for which it has the access permissions. It reports the local VM identifier, for each instrumented JVM found on the target system. The JPS command uses the java launcher to find the class name and arguments passed to the main method.

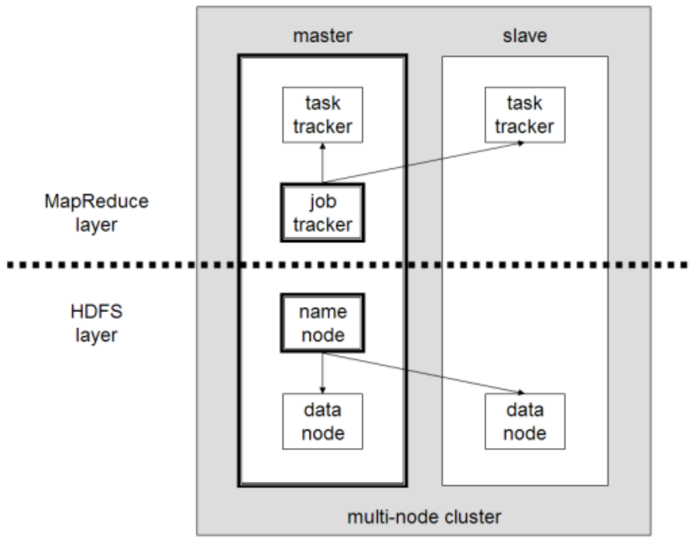
6.3 Apache Giraph deployed on Hadoop Cluster

Apache Giraph is an iterative graph processing system built for high scalability. It is inspired by the Bulk Synchronous Parallel model of distributed computation. Giraph libraries are deployed on Hadoop Cluster to run Graph Processing Algorithms such as Single Source Shortest Paths Compute and Page Rank Algorithms.

6.4 Apache Hadoop Multi Node Cluster

Data storage and processing is spread to multiple machines. The master node will run the "master" daemons for each layer: NameNode for the HDFS storage layer, and JobTracker for the MapReduce processing layer. The slave node will run the "slave" daemons: DataNode for the HDFS layer, and TaskTracker for MapReduce processing layer. Basically, the "master" daemons are responsible for coordination and management of the "slave" daemons while the latter will do the actual data storage and data processing work.





CHAPTER 7

PSEUDO CODE

7.1 Single Source Shortest Paths Compute Algorithm

```
function Dijkstra (Graph, source):

    dist[source] ← 0                    // Distance from source to source
    prev[source] ← undefined            // Previous node in optimal path initialization

    for each vertex v in Graph:  // Initialization
        if v ≠ source           // Where v has not yet been removed from Q (unvisited nodes)
            dist[v] ← infinity          // Unknown distance function from source to v
            prev[v] ← undefined         // Previous node in optimal path from source
        end if
        add v to Q                      // All nodes initially in Q (unvisited nodes)
    end for

    while Q is not empty:
        u ← vertex in Q with min dist[u]  // Source node in first case
        remove u from Q

        for each neighbor v of u:         // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:             // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u
            end if
        end for
    end while

    return dist[], prev[]

end function
```

7.2 Our Algorithm for converting Stanford Large Network Graphs in Edge List Input Format to acceptable JSON Long Double Float Double Vertex Input Format

Step 1: Read the input in the form of Simple Edge List Format.
Step 2: Create arrays of source ids and destination ids from the given input.
Step 3: For undirected graphs convert the given input format to a format where an edge is represented between every pair of vertices. For directed graphs jump to Step 4.
Step 4: Write a regular expression to convert the input in the given format to the JSON Long Double Float Double Vertex Input Format.
Step 5: Store the output in a file and provide as input to Single Source Shortest Paths Algorithm.

7.3 Our Algorithm for counting the number of Vertices and Edges in a given Large Graph

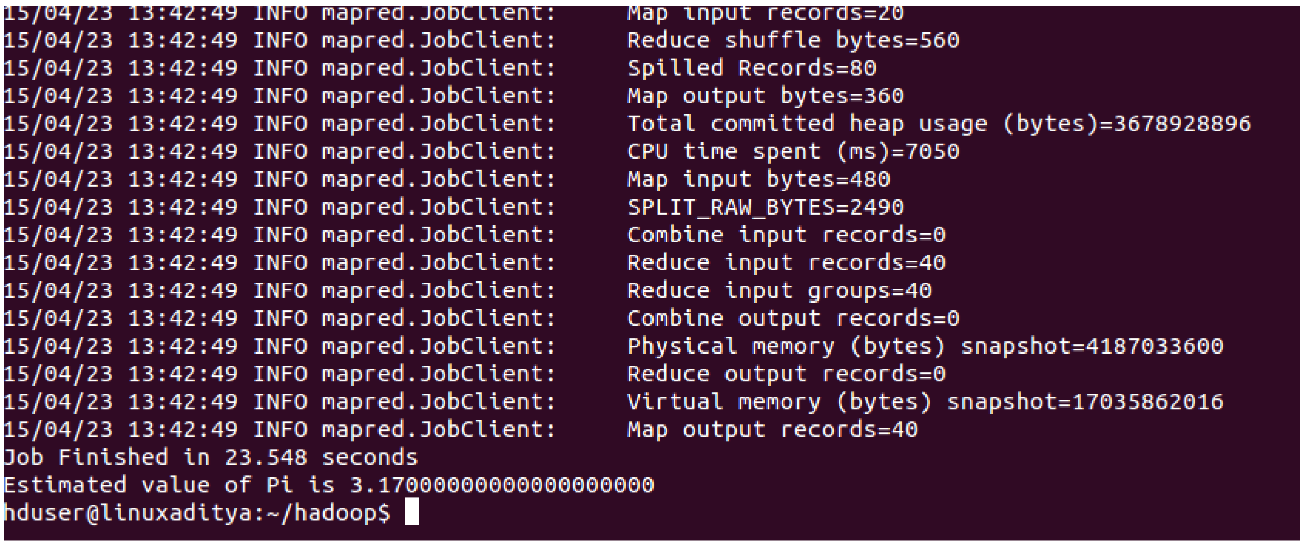
Step 1: Read the input in the form of Simple Edge List Format.
Step 2: Create arrays of source ids and destination ids from the given input.
Step 3: For undirected graphs convert the given input format to a format where an edge is represented between every pair of vertices. For directed graphs jump to Step 4.
Step 4: Count the number of lines in the given format and maintain a count to get the number of edges in the graph.
Step 5: In the regular expression conversion algorithm for each edge, count the number of distinct vertices in the graph by maintaining the count variable. This gives the total number of vertices in the graph.

CHAPTER 8

RESULTS DISCUSSION

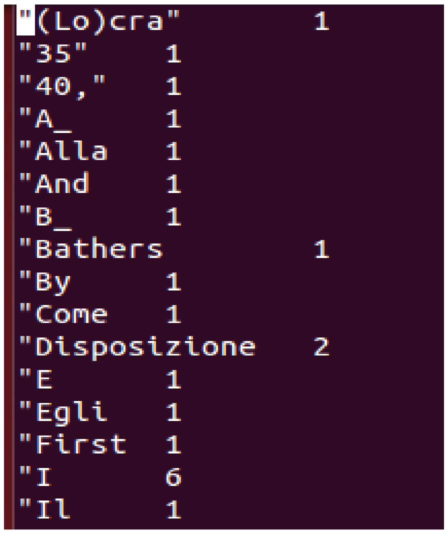
The project enabled us to get a clear understanding of Apache Hadoop and Apache Giraph and its implementation on a Single Node and Multi Node Hadoop Cluster.
We utilized this understanding to deploy Apache Giraph on Single Node and Multi Node Hadoop Cluster and run Giraph Jobs on the same.
These functionalities were further enhanced by deploying large graphs (Big Data sets) on real time servers and also simulating them on local machines using virtual box.

8.1 Running MapReduce Jobs on Apache Hadoop (PI Estimator)



```
15/04/23 13:42:49 INFO mapred.JobClient:     Map input records=20
15/04/23 13:42:49 INFO mapred.JobClient:     Reduce shuffle bytes=560
15/04/23 13:42:49 INFO mapred.JobClient:     Spilled Records=80
15/04/23 13:42:49 INFO mapred.JobClient:     Map output bytes=360
15/04/23 13:42:49 INFO mapred.JobClient:     Total committed heap usage (bytes)=3678928896
15/04/23 13:42:49 INFO mapred.JobClient:     CPU time spent (ms)=7050
15/04/23 13:42:49 INFO mapred.JobClient:     Map input bytes=480
15/04/23 13:42:49 INFO mapred.JobClient:     SPLIT_RAW_BYTES=2490
15/04/23 13:42:49 INFO mapred.JobClient:     Combine input records=0
15/04/23 13:42:49 INFO mapred.JobClient:     Reduce input records=40
15/04/23 13:42:49 INFO mapred.JobClient:     Reduce input groups=40
15/04/23 13:42:49 INFO mapred.JobClient:     Combine output records=0
15/04/23 13:42:49 INFO mapred.JobClient:     Physical memory (bytes) snapshot=4187033600
15/04/23 13:42:49 INFO mapred.JobClient:     Reduce output records=0
15/04/23 13:42:49 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=17035862016
15/04/23 13:42:49 INFO mapred.JobClient:     Map output records=40
Job Finished in 23.548 seconds
Estimated value of Pi is 3.17000000000000000000
hduser@linuxaditya:~/hadoop$
```

8.2 Running MapReduce Jobs on Apache Hadoop (Word Count Program)



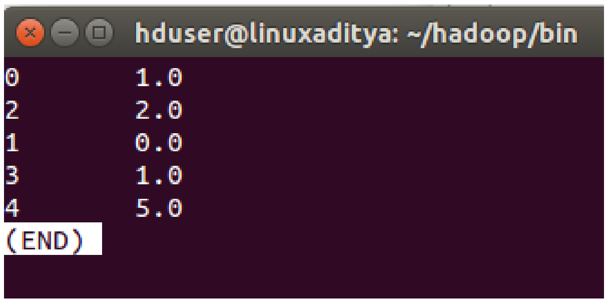
```
"(Lo)cra"        1
"35"     1
"40,"    1
"A_      1
"Alla    1
"And     1
"B_      1
"Bathers         1
"By      1
"Come    1
"Disposizione    2
"E       1
"Egli    1
"First   1
"I       6
"Il      1
```

8.3 Running Apache Giraph Job on Single Node Hadoop Cluster (Single Source Shortest Paths Compute)

Input File

```
[0,0,[[1,1],[3,3]]]
[1,0,[[0,1],[2,2],[3,1]]]
[2,0,[[1,2],[4,4]]]
[3,0,[[0,3],[1,1],[4,4]]]
[4,0,[[3,4],[2,4]]]
```

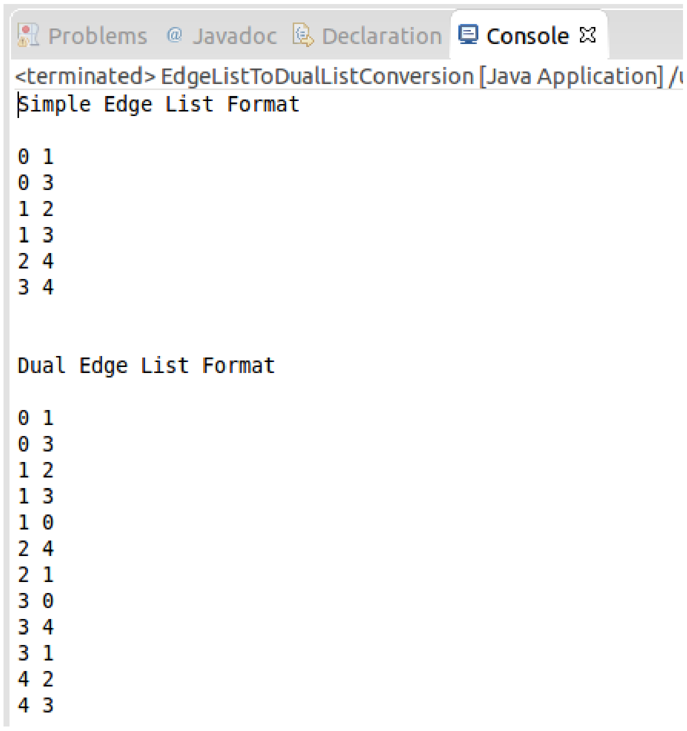
Output File



```
0       1.0
2       2.0
1       0.0
3       1.0
4       5.0
(END)
```

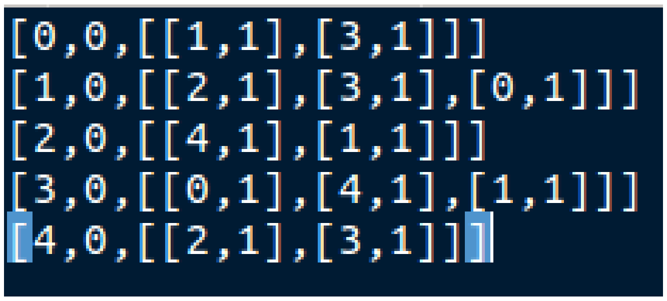
8.4 Running Stanford Large Network Graphs (SNAP – Big Data Set) on Apache Giraph deployed on Single Node Hadoop Cluster (Single Source Shortest Paths Compute)

Algorithm for converting Simple Edge List format to a format with edges between all the vertices



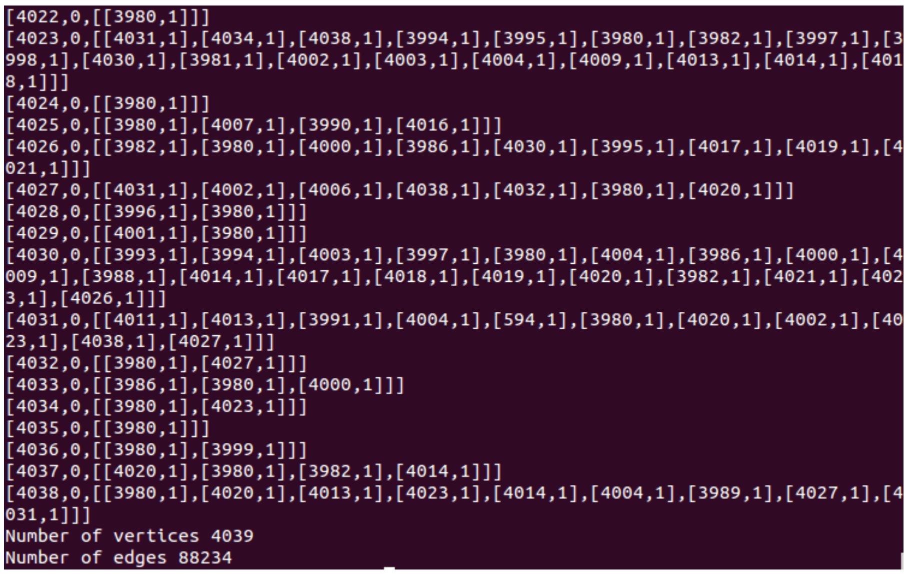
```
<terminated> EdgeListToDualListConversion [Java Application] /
Simple Edge List Format

0 1
0 3
1 2
1 3
2 4
3 4


Dual Edge List Format

0 1
0 3
1 2
1 3
1 0
2 4
2 1
3 0
3 4
3 1
4 2
4 3
```

Regular Expression Conversion Algorithm for converting the given Input Format to acceptable JSON Format

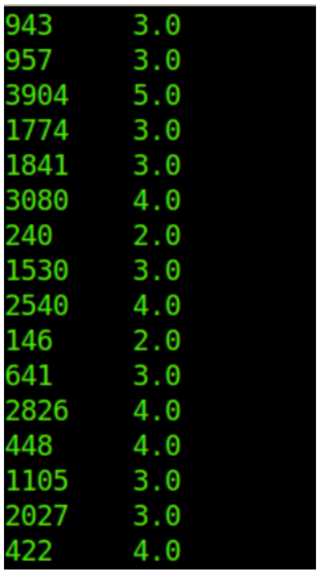


```
[0,0,[[1,1],[3,1]]]
[1,0,[[2,1],[3,1],[0,1]]]
[2,0,[[4,1],[1,1]]]
[3,0,[[0,1],[4,1],[1,1]]]
[4,0,[[2,1],[3,1]]]
```

Converted JSON Format and Output (Single Source Shortest Path) for facebook Stanford Large Network Graph (SNAP)

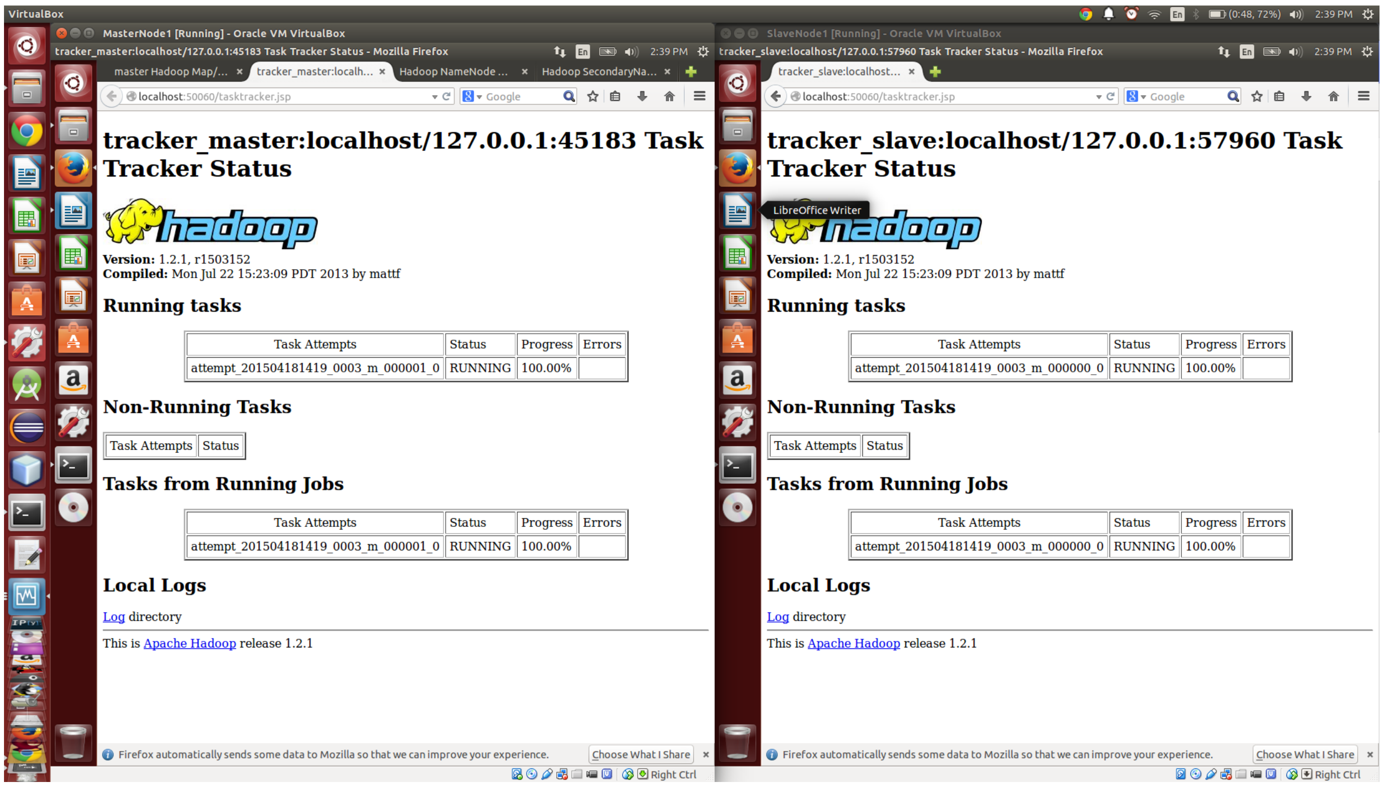


```
[4022,0,[[3980,1]]]
[4023,0,[[4031,1],[4034,1],[4038,1],[3994,1],[3995,1],[3980,1],[3982,1],[3997,1],[3
998,1],[4030,1],[3981,1],[4002,1],[4003,1],[4004,1],[4009,1],[4013,1],[4014,1],[401
8,1]]]
[4024,0,[[3980,1]]]
[4025,0,[[3980,1],[4007,1],[3990,1],[4016,1]]]
[4026,0,[[3982,1],[3980,1],[4000,1],[3986,1],[4030,1],[3995,1],[4017,1],[4019,1],[4
021,1]]]
[4027,0,[[4031,1],[4002,1],[4006,1],[4038,1],[4032,1],[3980,1],[4020,1]]]
[4028,0,[[3996,1],[3980,1]]]
[4029,0,[[4001,1],[3980,1]]]
[4030,0,[[3993,1],[3994,1],[4003,1],[3997,1],[3980,1],[4004,1],[3986,1],[4000,1],[4
009,1],[3988,1],[4014,1],[4017,1],[4018,1],[4019,1],[4020,1],[3982,1],[4021,1],[402
3,1],[4026,1]]]
[4031,0,[[4011,1],[4013,1],[3991,1],[4004,1],[594,1],[3980,1],[4020,1],[4002,1],[40
23,1],[4038,1],[4027,1]]]
[4032,0,[[3980,1],[4027,1]]]
[4033,0,[[3986,1],[3980,1],[4000,1]]]
[4034,0,[[3980,1],[4023,1]]]
[4035,0,[[3980,1]]]
[4036,0,[[3980,1],[3999,1]]]
[4037,0,[[4020,1],[3980,1],[3982,1],[4014,1]]]
[4038,0,[[3980,1],[4020,1],[4013,1],[4023,1],[4014,1],[4004,1],[3989,1],[4027,1],[4
031,1]]]
Number of vertices 4039
Number of edges 88234
```



```
943     3.0
957     3.0
3904    5.0
1774    3.0
1841    3.0
3080    4.0
240     2.0
1530    3.0
2540    4.0
146     2.0
641     3.0
2826    4.0
448     4.0
1105    3.0
2027    3.0
422     4.0
```

8.5 Running Giraph Jobs on Multi Node Hadoop Cluster (Analysis)



VirtualBox
MasterNode1 [Running] - Oracle VM VirtualBox
tracker_master:localhost/127.0.0.1:45183 Task Tracker Status - Mozilla Firefox
master Hadoop Map/...
tracker_master:localh...
Hadoop NameNode ...
Hadoop SecondaryNa...
localhost:50060/tasktracker.jsp
SlaveNode1 [Running] - Oracle VM VirtualBox
tracker_slave:localhost/127.0.0.1:57960 Task Tracker Status - Mozilla Firefox
tracker_slave:localhost...
LibreOffice Writer
2:39 PM
Right Ctrl

tracker_master:localhost/127.0.0.1:45183 Task Tracker Status

Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf

Running tasks

Task Attempts	Status	Progress	Errors
attempt_201504181419_0003_m_000001_0	RUNNING	100.00%	

Non-Running Tasks

Task Attempts	Status

Tasks from Running Jobs

Task Attempts	Status	Progress	Errors
attempt_201504181419_0003_m_000001_0	RUNNING	100.00%	

Local Logs

Log directory

This is Apache Hadoop release 1.2.1

Firefox automatically sends some data to Mozilla so that we can improve your experience. Choose What I Share

tracker_slave:localhost/127.0.0.1:57960 Task Tracker Status

Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf

Running tasks

Task Attempts	Status	Progress	Errors
attempt_201504181419_0003_m_000000_0	RUNNING	100.00%	

Non-Running Tasks

Task Attempts	Status

Tasks from Running Jobs

Task Attempts	Status	Progress	Errors
attempt_201504181419_0003_m_000000_0	RUNNING	100.00%	

Local Logs

Log directory

This is Apache Hadoop release 1.2.1

Firefox automatically sends some data to Mozilla so that we can improve your experience. Choose What I Share

CHAPTER 9

CONCLUSION

With various organizations shifting their operations to the Cloud, it becomes important to perform graph processing on big data. Giraph utilizes Apache Hadoop's MapReduce implementation to process graphs. Facebook used Giraph with some performance improvements to analyse one trillion edges using 200 machines in 4 minutes.
Since running these jobs on real time servers involves a lot of expenditure, Apache Giraph deployed on Hadoop clusters on local machines avoids this.
Giraph is based on a paper published by Google about its own graph processing system called Pregel. It can be compared to other Big Graph processing libraries such as Cassovary.

CHAPTER 10

FURTHER ENHANCEMENTS

10.1 Running Time Series Graphs

Deploying Time Series Graphs which is a sequence of data points, typically consisting of successive measurements over a time interval such as ocean tides, counts of sunspots and the real time road networks on Apache Giraph.

10.2 Working with GoFFish

GoFFish is a Sub-Graph Centric Framework for Large-Scale Graph Analytics. Large scale graph processing is a major research area for Big Data exploration.

CHAPTER 11

REFERENCES

[1] GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics
Yogesh Simmhan, Alok Kumbhare, Charith Wickramaarachchi, Soonil Nagarkar, Santosh Ravi, Cauligi Raghavendra, Viktor Prasanna, Indian Institute of Science, University of Southern California

[2] Pregel: A System for Large-Scale Graph Processing
Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski Google, Inc.

[3] Apache Giraph deployed on Single Node Hadoop Cluster: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[4] Apache Giraph deployed on Multi Node Hadoop Cluster: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

[5] Quick Start Apache Giraph: http://giraph.apache.org/quick_start.html

[6] Apache Giraph Foundation: http://giraph.apache.org/

[7] Wikipedia Apache Giraph: http://en.wikipedia.org/wiki/Apache_Giraph

[8] Apache Hadoop Foundation: https://hadoop.apache.org/

[9] Wikipedia Apache Hadoop: http://en.wikipedia.org/wiki/Apache_Hadoop

[10] Hadoop and Big Data: http://www.cloudera.com/content/cloudera/en/about/hadoop-and-big-data.html