DECEMBER 8, 2016

CS 3650: VISUAL LANGUAGES AND VISUAL PROGRAMMING

PEGAGUS IMPLEMENTATION ON BERKELEY - STANFORD WEBPAGES

INSTRUCTOR: PROF. S. K. CHANG
STUDENT: ANURADHA KULKARNI (ANK163)
UNIVERSITY OF PITTSBURGH
Department of Computer Science

1. ABSTRACT:

Graphs or networks are everywhere, ranging from the Internet Web graph, social networks (Facebook, Twitter), biological networks, and many more. The large volume of available data, the low cost of storage and the stunning success of online social networks and web2.0 applications all lead to graphs of unprecedented size. There are mainly two goals while analyzing these graphs or networks. One is to find patterns in these graphs and detect anomalies such as outliers, etc. Second is to scale up analyses to grab billions of nodes and edges. Graph Mining is an area of data mining to find patterns, rules, and anomalies of graphs. Typical graph mining algorithms silently assume that the graph fits in the memory of a typical workstation or at least on a single disk; the above graphs violate these assumptions, spanning multiple Giga-bytes, and heading to Tera- and Peta- bytes of data. Here comes PEGASUS, based on Hadoop is a graph mining package for handling graphs with millions of nodes and edges. My Project experiments run PageRank algorithm using PEGASUS on Berkeley-Stanford and Stanford Data sets collected from 2002. The findings of these graphs have been reported.

2. INTRODUCTION:

Graphs are ubiquitous: computer networks, social networks, mobile call networks, the World Wide Web, protein regulation networks to name a few. PEGASUS, an open source Peta-Scale Graph Mining System is a promising tool which provides parallelism and use Map-Reduce programming framework. PEGASUS has various contributions:

a. Unification of seemingly different graph mining tasks, via a generalization of matrix-vector multiplication (GIM-V).
b. The careful implementation of GIM-V, with several optimizations, and several graph mining operations (PageRank, Random Walk with Restart(RWR), diameter estimation, and connected components). Moreover, the method is linear on the number of edges, and scales up well with the number of available machines.
c. Successful combination of optimizations, which lead to up to 5 times better speed than naive implementation.
d. Analysis of large and real graphs.

The two well-known universities: University of Berkeley and Stanford University based in California are one of the world's leading research and teaching institutions. The domains Berkeley.edu and standford.edu may be accessed numerous number of times. It would be interesting to find out the most accessed web pages from these two domains in other words perform PageRank on this data set giving us an idea which page is actually accessed more. An inference of the result can help us understand whether most accessed web page belongs to Berkeley or Stanford. The project aims to analyze web graph (Berkeley- Stanford and Stanford Web graphs) with help of the well-known software, PEGASUS by computing the degree, PageRank and generate plots. It includes managing the graphs, running algorithms and generating plots.

3. BACKGROUND:

This section gives a background information about few topics that are required in understanding PEGASUS.

3.1. Large-Scale Graph Mining:

There are a huge number of graph mining algorithms, computing communities (eg. DENGRAPH, METIS), subgraph discovery (e.g., GraphSig, gPrune, gApprox, gSpan, Subdue, HSIGRAM/VSIGRAM, ADI, CSV), finding important nodes (e.g., PageRank and HITS), computing the number of triangles, computing the diameter, topic detection, attack detection, with too-many-to-list alternatives for each of the above tasks. Most of the previous algorithms do not scale, at least directly, to several millions and billions of nodes and edges. For connected components, there are several algorithms, using Breadth-First Search, Depth-First-Search, "propagation", or "contraction". These works rely on a shared memory model which limits their ability to handle large, disk-resident graphs.

3.2. MapReduce and Hadoop:

MAPREDUCE is a programming framework for processing huge amounts of unstructured data in a massively parallel way. MAPREDUCE has two major advantages:

a. The programmer is oblivious of the details of the data distribution, replication, load balancing etc.
b. The programming concept is familiar, i.e., the concept of functional programming.

Briefly, the programmer needs to provide only two functions, a map and a reduce. The typical framework is as follows:

a. The map stage sequentially passes over the input file and outputs (key, value) pair
b. The shuffling stage groups of all values by key
c. The reduce stage processes the values with the same key and outputs the final result.

HADOOP is the open source implementation of MAPREDUCE. HADOOP provides the Distributed File System (HDFS) and PIG is a high level language for data analysis. Due to its power, simplicity and the fact that building a small cluster is relatively cheap, HADOOP is a very promising tool for large scale graph mining applications.

3.3. PageRank:

PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google: "PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely

to receive more links from other websites." It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known. The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called "iterations", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

A probability is expressed as a numeric value between 0 and 1. A 0.5 probability is commonly expressed as a "50% chance" of something happening. Hence, a PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank.

4. PEGASUS:

The main idea in PEGASUS is to use 'Generalized Iterative Matrix-Vector multiplication' i.e. GIM-V that is provided by PEGASUS and then customizing the GIM-V to handle graph operations like PageRank. This section gives a brief information of the algorithm and its mechanism on using Hadoop.

4.1. Generalized Iterative Matrix-Vector multiplication (GIM-V):

It is a generalization of normal matrix-vector multiplication. Suppose we have a n by n matrix M and a vector v of size n. Let $m_{i,j}$ denote the $(i,j)^{th}$element of M. Then the usual matrix-vector multiplication is

$$M * v = v' \text{ where } v'_i = \sum_{j=1}^{n} m_{i,j}\, v_j$$

There are three operations, if customized separately, can help in performing graph mining algorithms: combine2(multiply the operands), combineAll (sum n multiplication results for node i) and assign (overwrite previous value v_i with new result to make v'_i). In GIM-V, the operator $\times_G$, defines these three operations arbitrarily:

$$v' = M *\times_G * v$$

Where

$$v'_i = assign(v_i,\ combineAll_i(\{x_j | j = 1 \dots \dots n \text{ and } x_j = combine2(m_{i,j}, v_j)\}))$$

The $\times_G$ operation is called iteratively until an algorithm specific convergence criterion is met and customize to handle Page Rank Algorithm.

4.2. GIM-V and PageRank:

The Page Rank algorithm would be a direct implementation of GIM-V. The page rank vector p satisfies the Eigenvector equation.

$$p^{next} = (c\ E^T + (1-c)U)p^{cur}$$

where c is a damping factor (usually set to 0.85), E is the row-normalized adjacency matrix (source, destination), and U is a matrix with all elements set to 1/n. The current PageRank vector p^{cur} is initialized to 1/n. The p^{next} is calculated until p converges. In this view, first a matrix M is constructed by column-normalize E^T such that every column of M sum to 1. Then the next PageRank is calculated by

$$p^{next} = M \times_G p^{cur}$$

where the three operations are defined as follows: \

a. $combine2(m_{i,j}, v_j) = c \times m_{i,j} \times v_j$
b. $combineAll_i(x_1, \ldots., x_n) = (1-c)/n + \sum_{j=1}^{n} x_j$
c. $assign(v_i, v_{new}) = v_{new}$

4.3. Fast Algorithm for GIM-V:

This describes the Hadoop algorithms for GIM-V. The GIM-V BASE is a two-stage algorithm whose pseudo code is in Algorithm 1 and 2. The inputs are an edge file and a vector file. Each line of the edge file contains one $(id_{src}, id_{dst}, mval)$ which corresponds to a non-zero cell in the adjacency matrix M. Similarly, each line of the vector file contains one $(id, vval)$ which corresponds to an element in the vector V . Stage1 performs combine2 operation by combining columns of matrix(id_{dst} of M) with rows of vector(id of V). The output of Stage1 are (key, value) pairs where key is the source node id of the matrix(id_{src} of M) and the value is the partially combined result($combine2(mval, vval)$). This output of Stage1 becomes the input of Stage2. Stage2 combines all partial results from Stage1 and assigns the new vector to the old vector. The $combineAll_i()$ and $assign()$ operations are done in line 16 of Stage2, where the “self” and “others” tags in line 16 and line 21 of Stage1 are used to make v_i and v_{new} of GIM-V, respectively. This two-stage algorithm is run iteratively until application-specific convergence criterion is met. In Algorithm 1 and 2, Output(k, v) means to output data with the key k and the value v.

Algorithm 1: GIM-V BASE Stage 1.

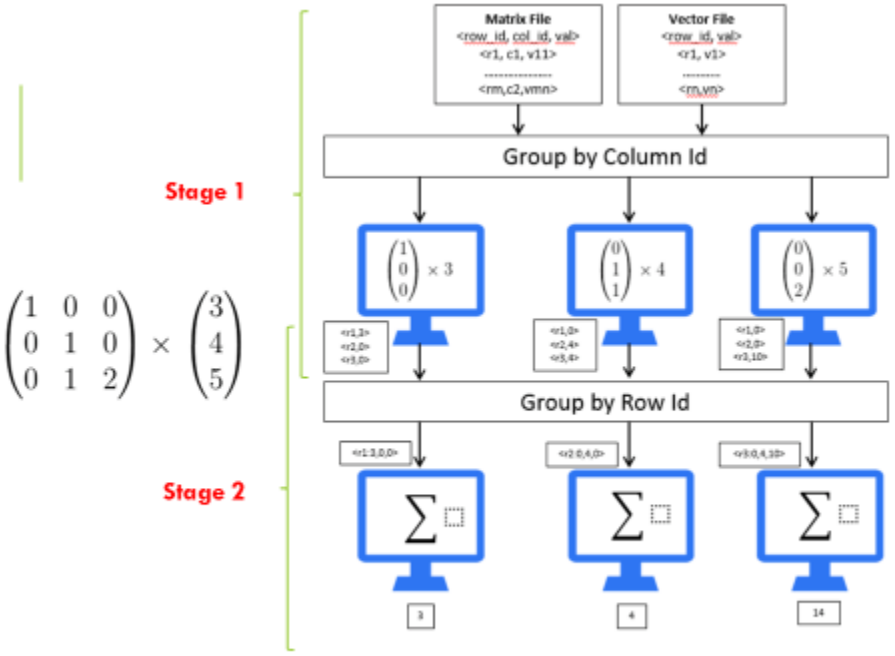
Input : Matrix $M = \{(id_{src}, (id_{dst}, mval))\}$,
Vector $V = \{(id, vval)\}$
Output: Partial vector
$V' = \{(id_{src}, \texttt{combine2}(mval, vval)\}$

```
1  Stage1-Map(Key k, Value v) ;
2  begin
3      if (k, v) is of type V then
4          Output(k, v);                          // (k: id, v: vval)
5      else if (k, v) is of type M then
6          (id_dst, mval) ← v;
7          Output(id_dst, (k, mval));             // (k: id_src)
8  end
9  Stage1-Reduce(Key k, Value v[1..m]) ;
10 begin
11     saved_kv ←[ ];
12     saved_v ←[ ];
13     foreach v ∈ v[1..m] do
14         if (k, v) is of type V then
15             saved_v ← v;
16             Output(k, ("self", saved_v));
17         else if (k, v) is of type M then
18             Add v to saved_kv      // (v: (id_src, mval))
19     end
20     foreach (id'_src, mval') ∈ saved_kv do
21         Output(id'_src, ("others",combine2(mval', saved_v)
22     end
23 end
```

Algorithm 2: GIM-V BASE Stage 2.

Input : Partial vector $V' = \{(id_{src}, vval')\}$
Output: Result Vector $V = \{(id_{src}, vval)\}$

```
1  Stage2-Map(Key k, Value v) ;
2  begin
3      Output(k, v);
4  end
5  Stage2-Reduce(Key k, Value v[1..m]) ;
6  begin
7      others_v ←[ ];
8      self_v ←[ ];
9      foreach v ∈ v[1..m] do
10         (tag, v') ← v;
11         if tag == "same" then
12             self_v ← v';
13         else if tag == "others" then
14             Add v' to others_v;
15     end
16     Output(k,assign(self_v,combineAll_k(others_v)));
17 end
```



$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}$$

Fig 1: Represents the Distribution of work among machines during GIM-V execution

5. PEGASUS AS ICONIC SYSTEM:

The input would be graphs with TAB-separated plain text format. Each line contains the source and destination node id of an edge. To bring out the logical meaning of these graphs, the input file will be computed over algorithms such as degree, PageRank, etc. After obtaining the logical meaning of graphs, they can be visualized by plotting the results into graph plots to find interesting patterns and anomalies. The dual representation of an object can be written as (X_m, X_i):

X_m: Logical part i.e. The logical meaning obtained after running the graph's nodes and edges across algorithms.

X_i: Physical part i.e. Generating the plot of these graph on visual graph plots so that the anomalies, outliers can be easily detected.

The Operator used is GIM-V operator.

6. INSTALLATION GUIDE:

This sections briefs about the installation needed to run PEGASUS. The installation commands are specific to Linux machines (i.e. Ubuntu Operating System).

6.1. Environment:

PEGASUS can be run in any machine that supports Hadoop, but the shell scripts and code packaging scripts works easily in Linux or Unix machines. PEGASUS needs the following software's to be installed in the system: Hadoop 0.20.1 or greater, Apache Ant 1.7.0 or greater, Java 1.6.x or greater (preferably from Sun), Python 2.4.x or greater, Gnuplot 4.2.x or greater.

6.2. Installing Java:

a. sudo apt-get update
b. sudo apt-get install default-jdk
c. java –version
d. update-alternatives --config java

6.3. Installing Python:

Python 3.4 is installed on the stable release of Ubuntu 14.04. Use python3 to use python.

6.4. Installing Apache Ant:

a. sudo apt-get install ant

6.5. Installing Gnuplot:

a. sudo apt-get install gnuplot

6.6. Installing Hadoop:

There are a number of steps in installing and setting up the Hadoop on a single machine.

6.6.1. Setting up a dedicated Hadoop user group:

a. sudo addgroup hadoop
b. sudo adduser --ingroup hadoop hadoop #Prompts you for some additional details like first name, last name. Just hit enter for the defaults.

6.6.2. Downloading Hadoop:

a. cd /
b. cd usr/local
c. wget http://apache.claz.org/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz
d. sudo tar xzf hadoop-2.7.1.tar.gz
e. sudo mv hadoop-2.7.1 hadoop
f. sudo chown -R hadoop:hadoop hadoop #Provide rights for the created hadoop group and hadoop user

6.6.3. Setting up single node cluster:

6.6.3.1. Setting up configuration file (bashrc file):

a. 3D.1.1 su - hadoop #Change to the previously created hadoop user using the command
b. 3D.1.2 vi ~/.bashrc

--Add the following lines into the bashrc file and save it --

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

c. source ~/.bashrc #To reflect the made changes

6.6.3.2. Pointing to JAVA_HOME and disabling IPv6:

a. su - [Prompts for system password, just enter it.]
b. cd /
c. cd usr/local/hadoop/etc/hadoop
d. vi hadoop-env.sh

--Add the following lines to hadoop-env.sh--

```
export JAVA_HOME =/usr/lib/jvm/java-7-openjdk-amd64
```

export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true #Disabling IPv6 for Hadoop

6.6.3.3. Modifying core-site.xml:

a. vi core-site.xml

--Add the following lines to core-site.xml between the configuration tags--

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>Temporary Directory.</description>
</property>

<property>
  <name>fs.defaultFS</name>
  <value>hdfs://master:54310</value>    #Make sure to change the machine name here.
  <description>Use HDFS as file storage engine</description>
</property>
```

#Make sure to save the made changes

6.6.3.4. Modifying yarn-site.xml:

a. vi yarn-site.xml

--Add the following lines to yarn-site.xml between the configuration tags--

```
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>

<property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

#Make sure to save the made changes

6.6.3.5. Creating and editing mapred-site.xml:

a. cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml
b. vi mapred-site.xml

--Add the following lines to mapred-site.xml between the configuration tags--

```
<property>
```

<name>mapreduce.jobtracker.address</name>
<value>master:54311</value> #Make sure to change the machine name here.
<description>The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map
and reduce task.
</description>
</property>

#Make sure to save the made changes

6.6.3.6. Modifying hdfs-site.xml:

a. vi hdfs-site.xml

--Add the following lines to hdfs-site.xml between the configuration tags--

<property>
<name>dfs.replication</name>
<value>1</value>
</property>
#Make sure to save the made changes

6.6.3.7. Creating a temporary folder for intermediate map-reduce results:

a. sudo mkdir -p /app/hadoop/tmp
b. sudo chown hadoop:hadoop /app/hadoop/tmp #Assign rights to previously created hadoop user group

6.6.3.8. Formatting namenode and starting services:

a. su - hadoop #Shift to hadoop user
b. hdfs namenode -format #Donot format namenode when hdfs is running
c. start-dfs.sh
d. start-yarn.sh
e. jps

#You should see the following services running -
Jps
NodeManager
SecondaryNameNode
NameNode
ResourceManager
DataNode

6.6.3.9. Shutting down the file system and YARN:

a. stop-dfs.sh
b. stop-yarn.sh
c. jps

6.7. Installing PEGASUS:

a. Download the installation file 'PEGASUSH-2.0.tar.gz' from http://www.cs.cmu.edu/~pegasus
b. Extract the file, then the directory 'PEGASUS' will be created.
c. Cd to the PEGASUS directory, then you are done.

7. PROJECT EXPERIMENT:

This section gives the details about how to use the PEGASUS for my project scenario i.e. applying the PEGASUS of the two data sets that Berkeley-Stanford and Stanford Data set.

7.1. Input:

PEGASUS works on graphs with TAB-separated plain text format. Each line contains the source and destination node id of an edge. The node id starts from 0. For example, here is an example graph. It has 16 nodes.

0	1
1	2
1	3
3	4
3	6
5	6
6	7
6	8
6	9
10	11
10	12
10	13
10	14
10	15

Fig2: Example of the input File

For my experiment, I took input data file from snap.stanford.edu. I took the Berkeley – Stanford data set and Stanford data set for my project. Both the data sets were collected in 2002.

7.1.1. Berkeley-Stanford Data set:

Nodes represent pages from berkely.edu and stanford.edu domains and directed edges represent hyperlinks between them. The data was collected in 2002.

Dataset statistics	
Nodes	685230
Edges	7600595
Nodes in largest WCC	654782 (0.956)
Edges in largest WCC	7499425 (0.987)
Nodes in largest SCC	334857 (0.489)
Edges in largest SCC	4523232 (0.595)
Average clustering coefficient	0.5967
Number of triangles	64690980
Fraction of closed triangles	0.002746
Diameter (longest shortest path)	514
90-percentile effective diameter	9.9

7.1.2. Stanford Data set:

Nodes represent pages from Stanford University (stanford.edu) and directed edges represent hyperlinks between them. The data was collected in 2002.

Dataset statistics	
Nodes	281903
Edges	2312497
Nodes in largest WCC	255265 (0.906)
Edges in largest WCC	2234572 (0.966)
Nodes in largest SCC	150532 (0.534)
Edges in largest SCC	1576314 (0.682)
Average clustering coefficient	0.5976
Number of triangles	11329473
Fraction of closed triangles	0.002889
Diameter (longest shortest path)	674
90-percentile effective diameter	9.7

7.2. Run the Hadoop Cluster:

In order to run the Hadoop cluster, first login to the hadoop user that was created during installation and then start the Hadoop Cluster.

a. su – hadoop
b. start-dfs.sh
c. start-yarn.sh

In order to check if all the nodes have started please run jps on the command prompt and you should be able to see all the 6 nodes that are shown in the installation guide. This shows that all the nodes have started.

7.3. Run PEGASUS:

Once the Hadoop is started, we can run the PEGASUS using the shell. To start the PEGASUS shell, type `pegasus.sh` in the command line. Here is the list of the possible commands in the shell.

Command	Description
add [file or directory] [graph_name]	upload a local graph file or directory to HDFS
del [graph_name]	delete a graph
list	list graphs
compute ['deg' or 'pagerank' or 'rwr' or 'radius' or 'cc'] [graph_name]	run an algorithm on a graph
plot ['deg' or 'pagerank' or 'rwr' or 'radius' or 'cc' or 'corr'] [graph_name]	generate plots
help	show this screen
demo	show demo
exit	exit PEGASUS

Fig 3: All the available command in PEGASUS

If you use the `compute` command to run algorithms, the result is saved under the HDFS directory pegasus/graphs/[GRAPH_NAME]/results/[ALGORITHM_NAME].

7.3.1. Interactive Shell:

To access the shell, type pegasus.sh in the PEGASUS installation directory. Then, the PEGASUS shell will appear. For available commands in the shell, type help and it will show the commands displayed in Fig 3.

```
[pegasus@heineken PEGASUS]$ ./pegasus.sh

        PEGASUS: Peta-Scale Graph Mining System
        Version 2.0
        Last modified September 5th 2010

        Authors: U Kang, Duen Horng Chau, and Christos Faloutsos
                 School of Computer Science, Carnegie Mellon University
        Distributed under APL 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

        Type `help` for available commands.
        The PEGASUS user manual is available at http://www.cs.cmu.edu/~pegasus
        Send comments and help requests to <ukang@cs.cmu.edu>.


PEGASUS> help
```

Fig 4: Starting the PEGASUS interactive shell

7.3.2. Managing graphs:

To use PEGASUS, the graphs to be analyzed should be uploaded to the Hadoop File System (HDFS). In the shell, the add command is used for uploading a graph to HDFS. To add a local edge file 'www_edges.tab' to HDFS and name it to 'www', issue the following command:

```
PEGASUS> add www_edges.tab www
Creating pegasus/graphs/www in HDFS
Creating pegasus/graphs/www/edge in HDFS
Graph www added.
PEGASUS> list
=== GRAPH LIST ===

www

PEGASUS>
```

Fig 5: Adding the graphs in HDFS

7.3.3. Running PageRank Algorithm:

To compute the PageRank, use the compute pagerank [graph_name] command. On entering the command, it will ask additional parameters: the number of nodes in the graph, the number of reducers, and whether to symmetrize the graph. In this example, I use 325729 for the number of nodes, and 10 for the number of reducers, and 'nosym' which means not to symmetrize the graph. After entering the parameters, the PageRank is computed on Hadoop.

```
PEGASUS> compute pagerank www
Enter parameters: [#_of_nodes] [#_of_reducers] [makesym or nosym]: 325729 10 nos
ym
rmr: cannot remove pr_tempmv: No such file or directory.
rmr: cannot remove pr_output: No such file or directory.
rmr: cannot remove pr_minmax: No such file or directory.
rmr: cannot remove pr_distr: No such file or directory.

---------[PEGASUS: A Peta-Scale Graph Mining System]--------

[PEGASUS] Computing PageRank. Max iteration = 1024, threshold = 3.07003674833987
76E-7, cur_iteration=1

Creating initial pagerank vectors.............
```

Fig 6: Computing the Graph mining Algorithm

When the computation is finished, you will see the following messages.

```
[PEGASUS] PageRank computed.
[PEGASUS] The final PageRanks are in the HDFS pr_vector.
[PEGASUS] The minium and maximum PageRanks are in the HDFS pr_minmax.
[PEGASUS] The histogram of PageRanks in 1000 bins between min_PageRank and max_P
ageRank are in the HDFS pr_distr.

Creating pegasus/graphs/www/results/pagerank in HDFS
PEGASUS>
```

Fig 7: Commands displayed on computation of the Algorithm

7.3.4. Plotting the Graph:

The PageRank distribution is plotted by the plot pagerank [graph_name] command. The output file [graph_name]_pagerank.eps is generated in the current directory. Here is an example of the PageRank distribution plotted.

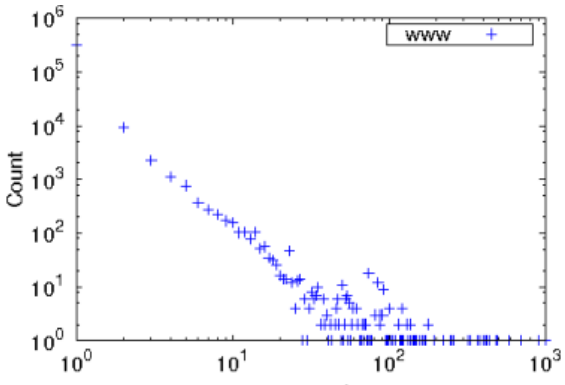


Fig 8: Example of the Page Rank Plot

7.4. Output:

Apart from the plot, there are three other things that can be taken as an output from the HDFS directory for analysis:

a. **pr_vector:**
   Each line contains the PageRank of each node in the format of (nodeid TAB "v"PageRank_of_the_node). - For example, the line "1 v0.10231778333763829" means that the PageRank of node 1 is 0.10231778333763829
b. **pr_minmax:**
   The minimum and the maximum PageRank. - The minimum PageRank is the second column of the line that starts with "0". - The maximum PageRank is the second column of the line that starts with "1".
c. **pr_distr:**
   The histogram of PageRank. It divides the range of (min_PageRank, max_PageRank) into 1000 bins and shows the number of nodes which have PageRanks that belong to such bins.

7.4.1. Berkeley -Stanford Output:

The minimum and maximum PageRank for this data set was found to be: 2.1890456658077156E-7 (minimum) and 0.005618834606433688 (maximum). The distributions as mentioned above is into 1000 bins. The first bin consists has around 676217 nodes and the second bin has 5744. The Page Rank plot for this data set:

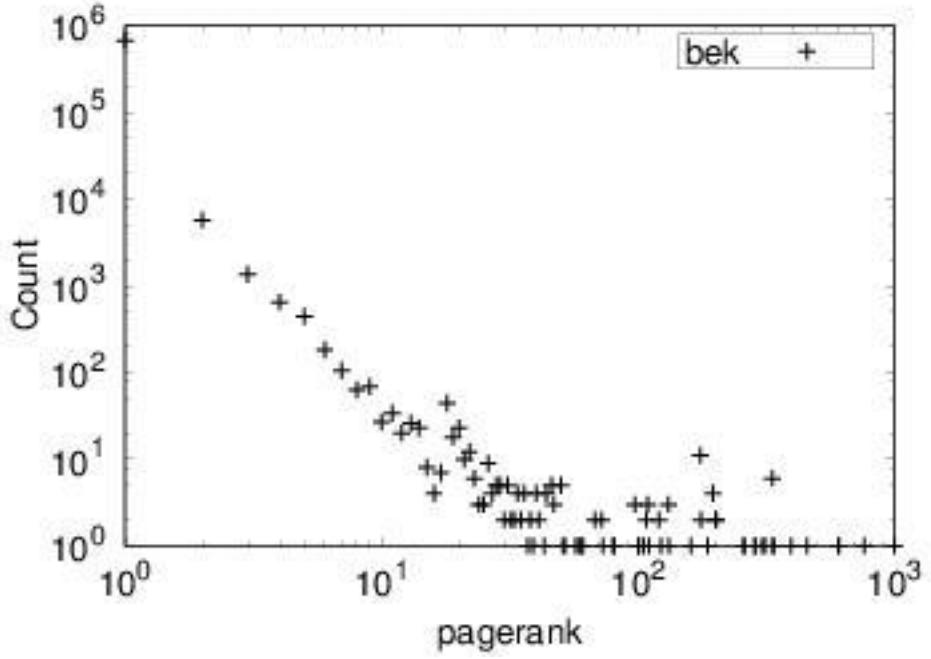


Fig 9: PageRank plot for Berkeley-Stanford data set

7.4.2. Stanford Output:

The minimum and maximum PageRank for this data set was found to be: 1.1890456658077156E-7 (minimum) and 0. 005618834606433688 (maximum). The distributions as mentioned above is into 1000 bins. The first bin consists has around 280003 nodes and the second bin has 1024. The Page Rank plot for this data set:

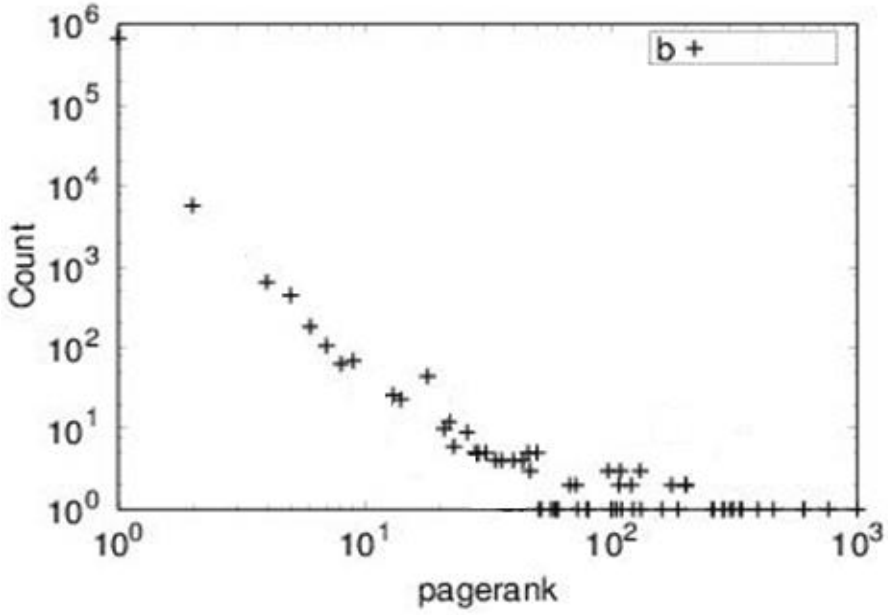


Fig 10: PageRank plot for Stanford data set

8. ANALYSIS AND RESULTS:

For the Analysis purpose, the pr_vector and the pr_distr should be copied onto your local system. The pr_minmax can be noted down. Order the PageRank value i.e. v0.02257263 in descending order to get the top 10 values. Before doing that, I mapped the nodes to a list of Berkeley-Stanford pages from the Berkeley.edu and Stanford.edu domain. I applied the same technique for Stanford data set instead here I mapped the nodes with pages from Stanford.edu domain. The list was obtained from snap.edu website. On careful observation on the two PageRank graphs, I could clearly point out the anomalies and similarity of the two graphs.

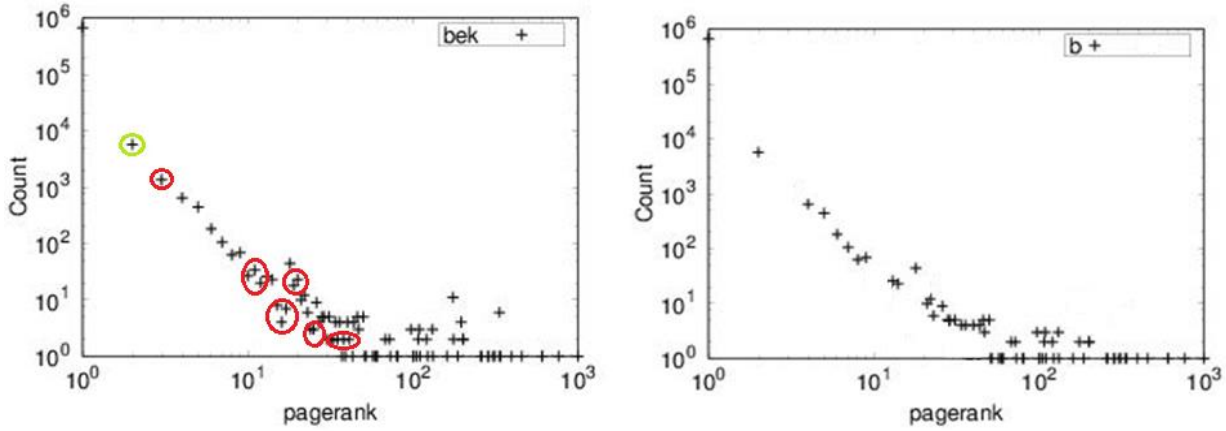


Fig 11: PageRank plots showing similarity and difference between the two data sets

Here green circle show the similarity and red circle shows the difference i.e. these points are only seen in Berkeley-Stanford PageRank plot whereas are not seen in Stanford PageRank plot. There are few points that are common between them as well. From this graph we can infer:

a. The highest bin belongs to Stanford as it correlates to the one on plot that contains only Stanford data set.
b. The second bin belongs to Berkeley as this bin point is missing in the Stanford PageRank plot.
c. The points that match shows that this bin share pages from both the Stanford and Berkeley domains.

The top 5 pages ranked the most in Stanford domain are:

1. online.stanford.edu/
2. www.stanford.edu/research/
3. searchworks.stanford.edu/
4. admission.stanford.edu/application/
5. doresearch.stanford.edu/research-administration

The top 5 pages ranked the most in Berkeley domain are:

1. guides.lib.berkeley.edu/ebooks
2. graddashboard.berkeley.edu/
3. graduategiving.berkeley.edu/
4. grad.berkeley.edu/program/computer-science/
5. career.berkeley.edu/

9. CONCLUSION:

Based on this project results, two main things can be concluded:

1. The most highly ranked pages belong to Stanford.edu domain, the top three domain being online.stanford.edu, stanford.edu/research and searchworks.stanford.edu.

2. The other conclusion can be made on the highlights of PEGASUS i.e. it's linear run time on the number of edges. Berkeley-Stanford data set has 7600595 edges and the time taken to process it was 6 hours whereas the Stanford data set has 2312497 and it took around 2.5 hours to process.

There are many research directions to add to PEGASUS. There is various other research work from HEIGEN, a proposed eigensolver for the spectral analysis of very largescale graphs to Apolo, a system that uses a mixed-initiative approach— combining visualization, rich user interaction and machine learning—to guide the user to incrementally and interactively explore large network data and make sense of it. The main contribution is to provide effectiveness, careful design (in order to include parallelism) and scalability.

10. REFERENCE:

[1] U Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations. IEEE International Conference On Data Mining 2009, Miami, Florida, USA.

[2] U Kang. "Mining Tera-Scale Graphs: Theory, Engineering and Discoveries." Diss. Carnegie Mellon U, 2012. Print.

[3] https://snap.stanford.edu/data/index.html#web

[4] http://social.technet.microsoft.com/wiki/contents/articles/13845.the-hadoop-on-azure-pegasus-page-rank-sample.aspx

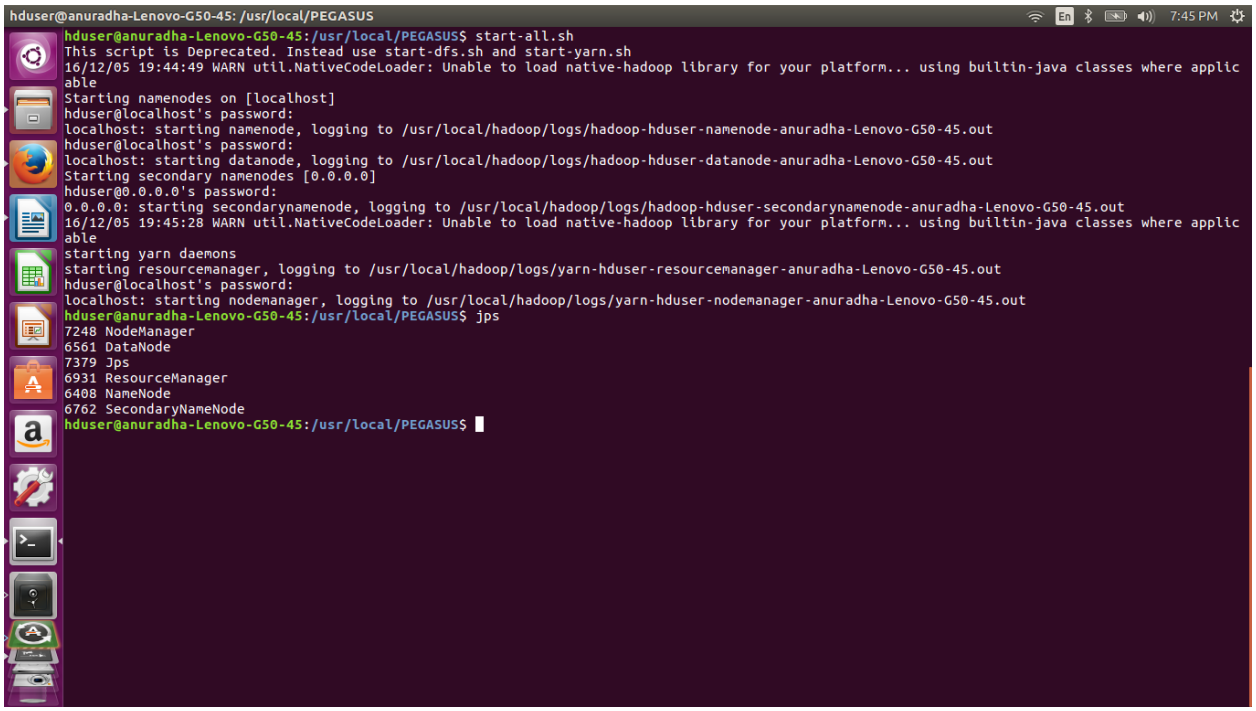
[5] https://en.wikipedia.org/wiki/PageRank

[6] Http://home.ie.cuhk.edu.hk/~wkshum/papers/pagerank.pdf

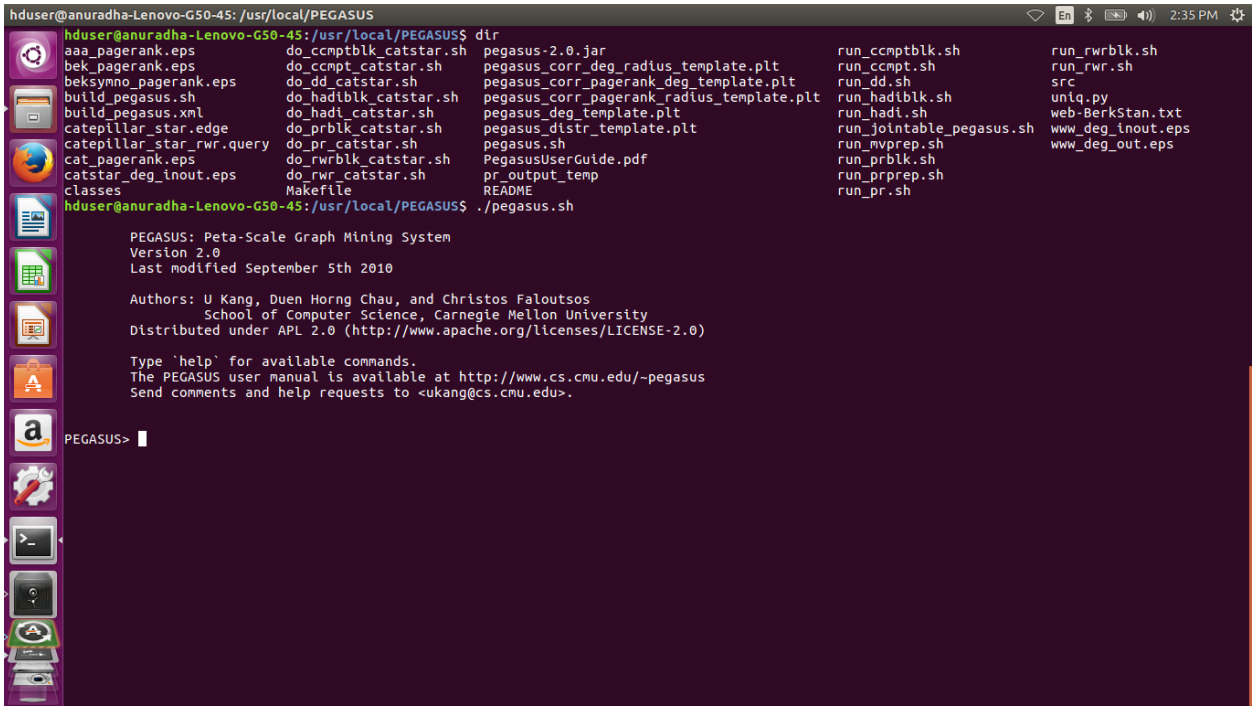
11.APPENDIX:

This section has the screen shots.

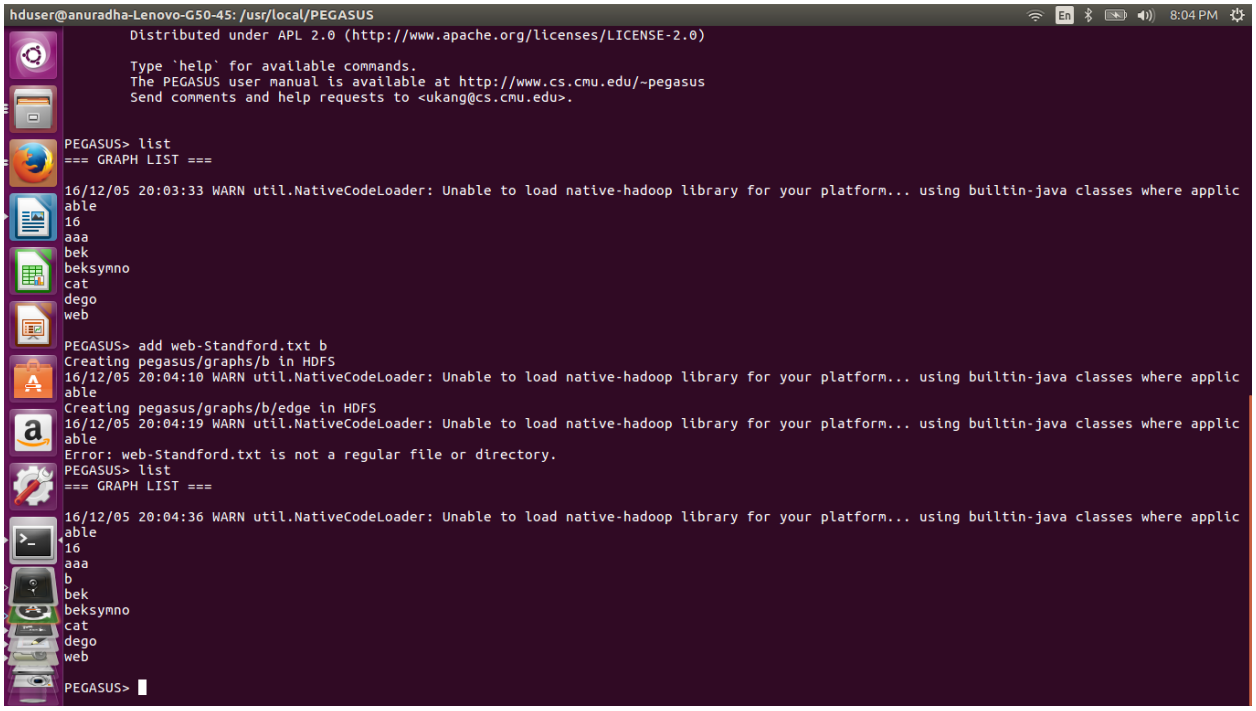
11.1. Starting the Hadoop cluster on single node:



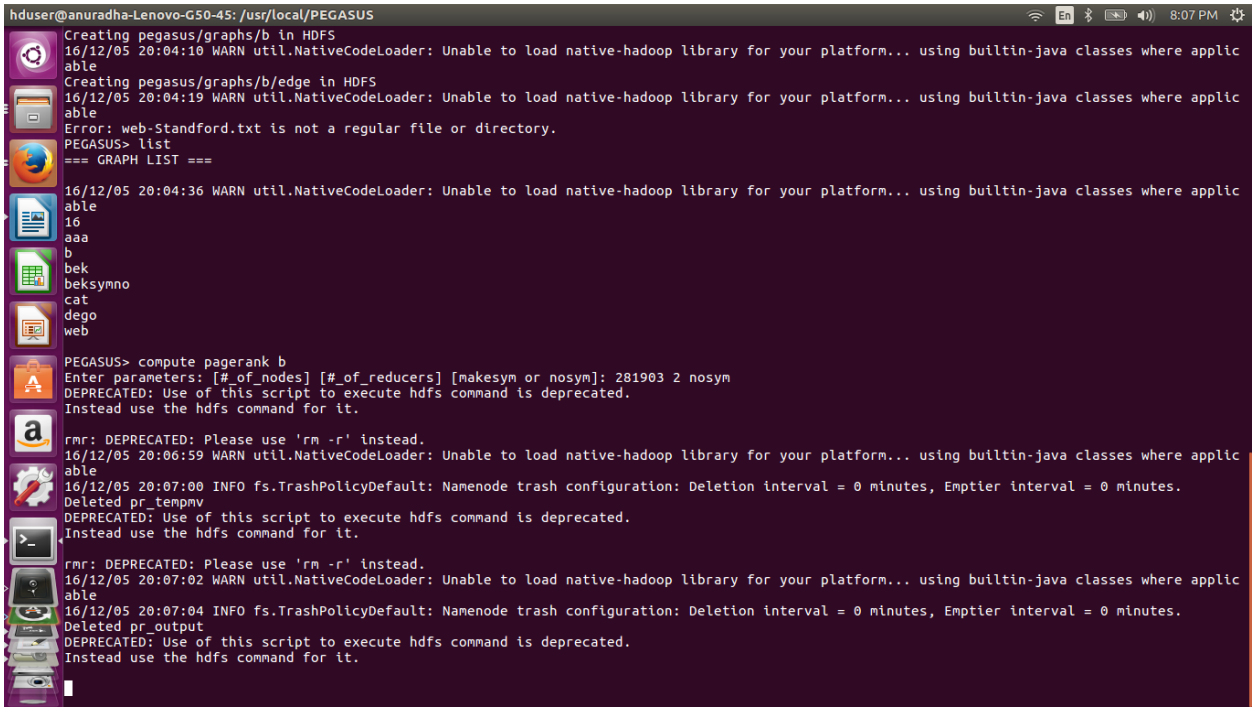
11.2. Starting the PEGASUS shell:



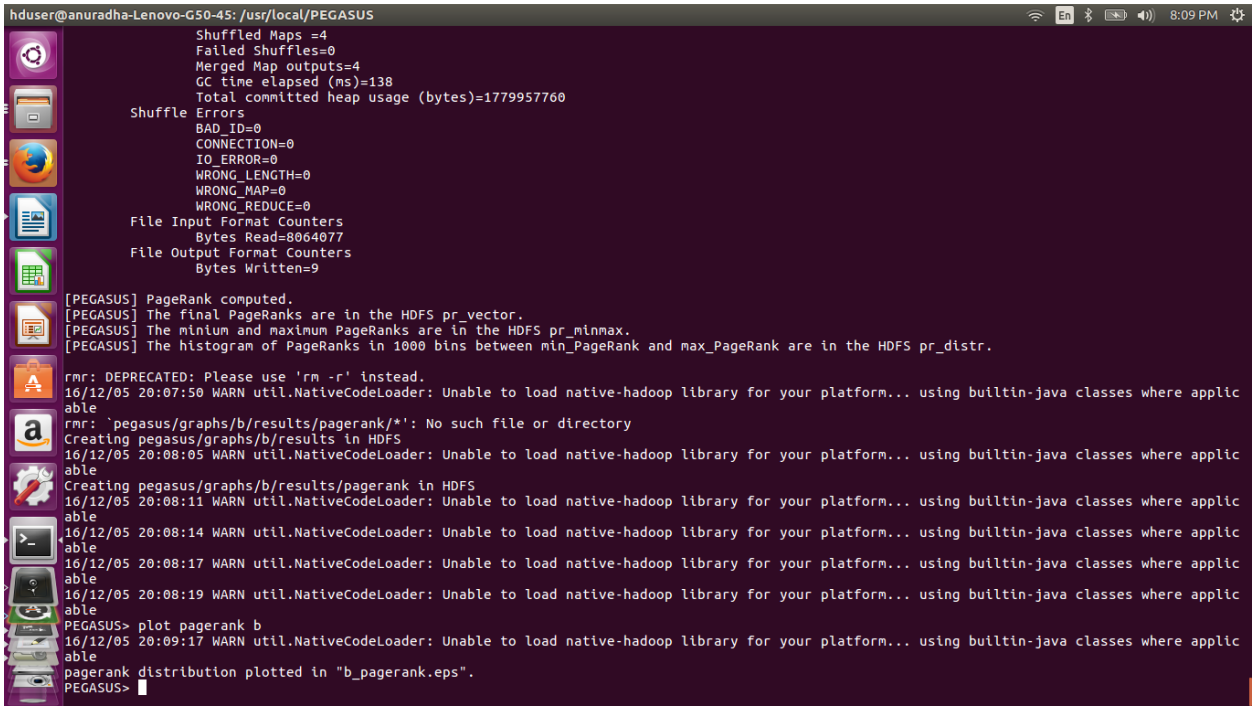
11.3. Adding the input file in HDFS:



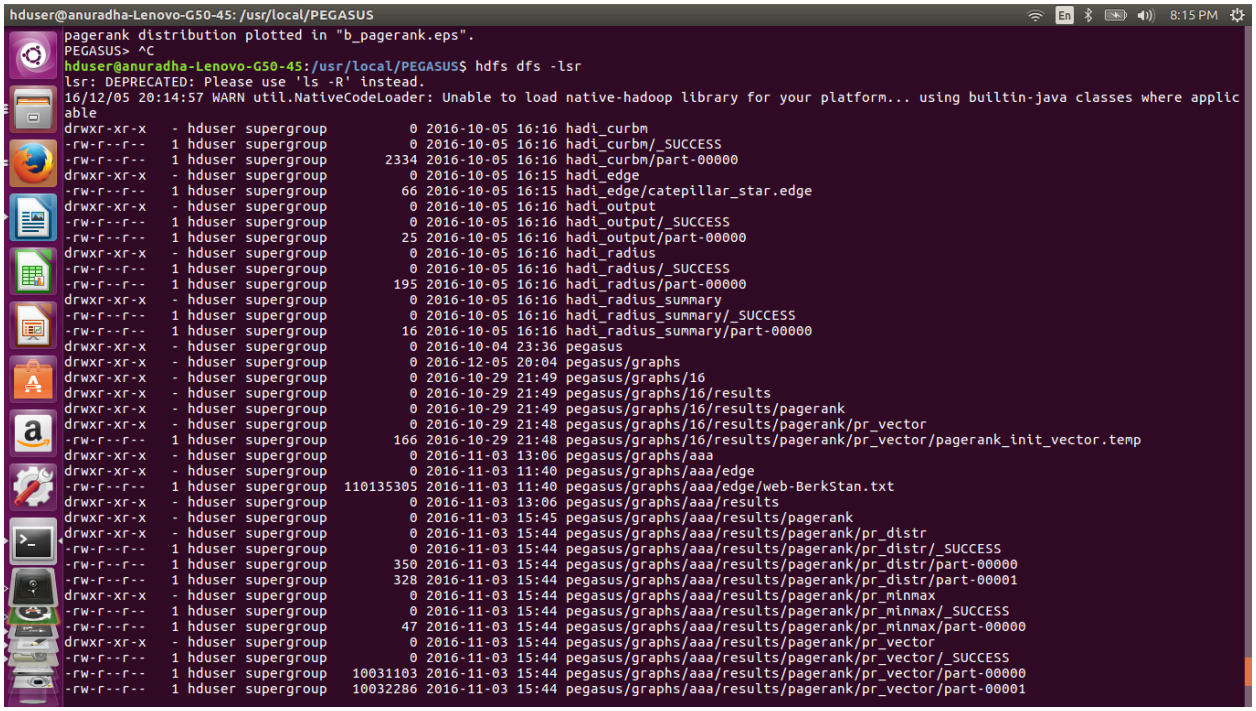
11.4. Compute the page Rank:



11.5. Plot the PageRank:



11.6. View the file directories output:



11.7 Stop Hadoop Cluster on Single Node: