HPC, Grids, Clouds: A Distributed System from Top to Bottom

Summarizes and integrates semester long work while putting together overall findings and conclusion.

Indiana University, Bloomington

Fall-2012

A project report submitted to Indiana University

By

Shubhada Karavinkoppa and Jayesh Kawli

Under supervision of Prof. Judy Qiu

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Table of Contents

1. Introduction

This is the final project report for implementation and performance analysis of Page Rank algorithm. We summarize our semester long work in this report. This includes initial and advanced page rank algorithm implementation, analysis and monitoring of resources it consumes on non-distributed as well as distributed system.

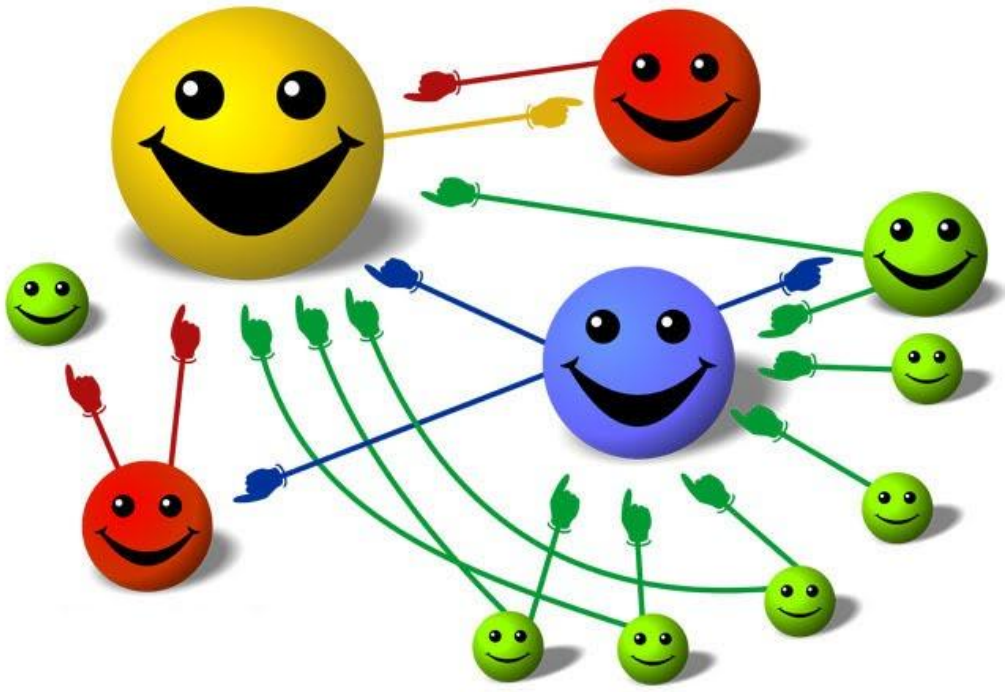
This work can be regarded as a walkthrough various project phases. In the first phase we implemented sequential and parallel version of page rank algorithm and tested it on very large data set. However, we did not monitor how it consumes resources and impact of parallel implementation on system resource consumption.

As a part of second phase, we analysed and monitored how page rank algorithm uses resources by running it on local departmental machine. This analysis did not only involve numbers but also its visual representation for quick analysis.

In the third phase, we ran same implementation in distributed/cloud environment. This was essential as program was itself crafted to take advantage of distributed environment. We tested our program on both Bare Metal and Virtual Machine environment and observed how the performance differs in both environments. This was followed by analysis and conclusion on difference between its behaviour in non-distributed and distributed environment.

Fourth and final phase involved dynamically switching clusters on Academic Cloud (FutureGrid) which is a multicore supercomputer environment provided by Indiana University, Bloomington. Purpose of this phase was to automate process of acquiring resources, running PageRank, resource monitoring tool and releasing resources on respective nodes after finishing job.

2. Phase 1-Part 1: Sequential Page Rank algorithm



Source: http://www.webseoanalytics.com/blog/is-google-pagerank-still-important-in-seach-engine-optimization/

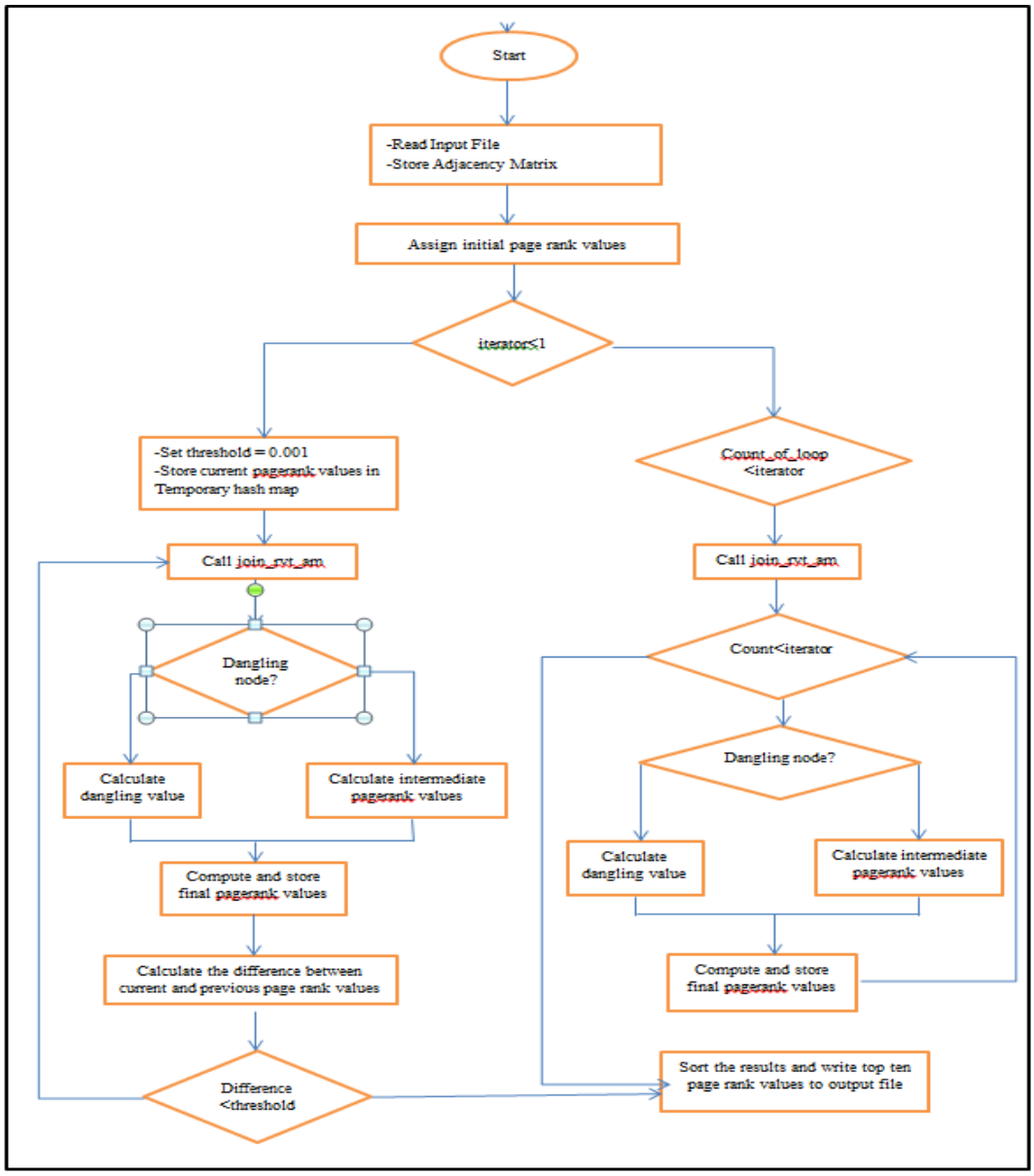
2.1. Introduction

In this project we implemented Sequential PageRank algorithm. The algorithm calculates the PageRank value of each page depending on the inbound links to that page. Each inbound link is like a vote. These votes are used to determine which pages are more important.

In regular implementation calculations are done iteratively until the values converge. But in out implementation we have two options - Until the values converge or the specified number iterations are executed.

2.2. Architecture

The below diagram shows the high level architecture of Algorithm



2.3. Optimizations and Efficiency boosting

Instead of going over for loop for number of iterations given from command line, we try to limit the iterations until convergence happens. Each time page rank values are calculated, we compare them against earlier page rank computations. If this difference goes below specific threshold (Set in the program) we break the loop and return results of iteration in output file.

2.4. Results

List of Top 10 URLs was accumulated in output file along with their page rank value as follows

Top 10 URLs with Highest Page Rank values

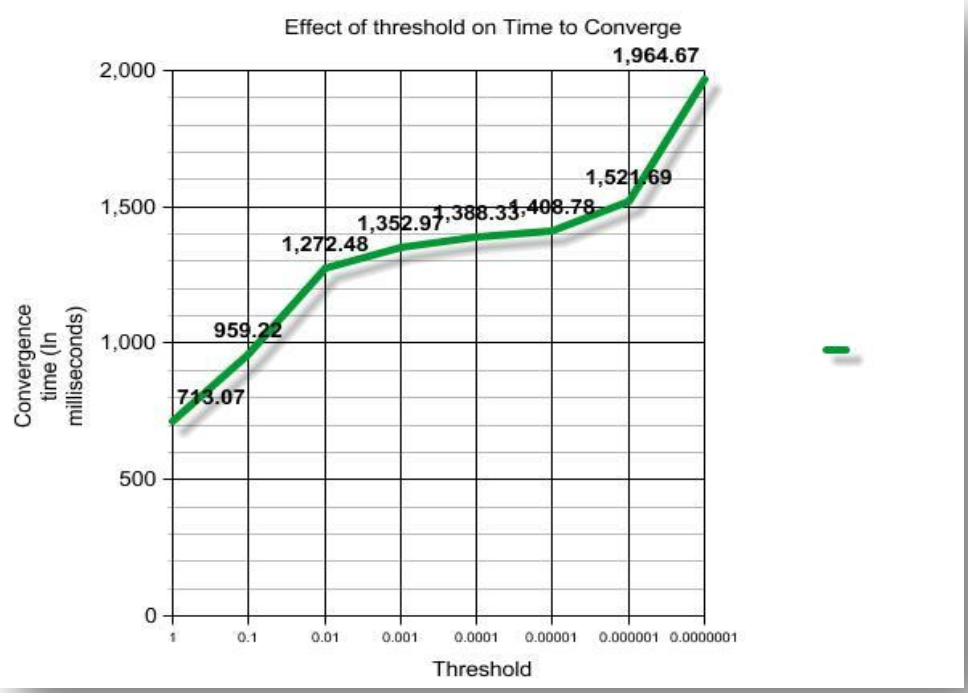
URL	Page Rank
4	0.13821304217473024
34	0.12302491704773691
0	0.11257935294330157
20	0.07736590523118934
146	0.05713176348278271
2	0.04792631126705502
12	0.02006643690709921
14	0.01790592635583653
16	0.01302811362009985
6	0.01295544157190792

Sum of individual Page rank values = 0.9999999999999800

2.5. Performance Analysis

Program converged for value well below given number of input iterations. Page rank values were calculated for only 14 numbers of times until convergence occurred for threshold value of 0.001

2.5.1. Following graph shows time taken to converge (ms) as a function of threshold value



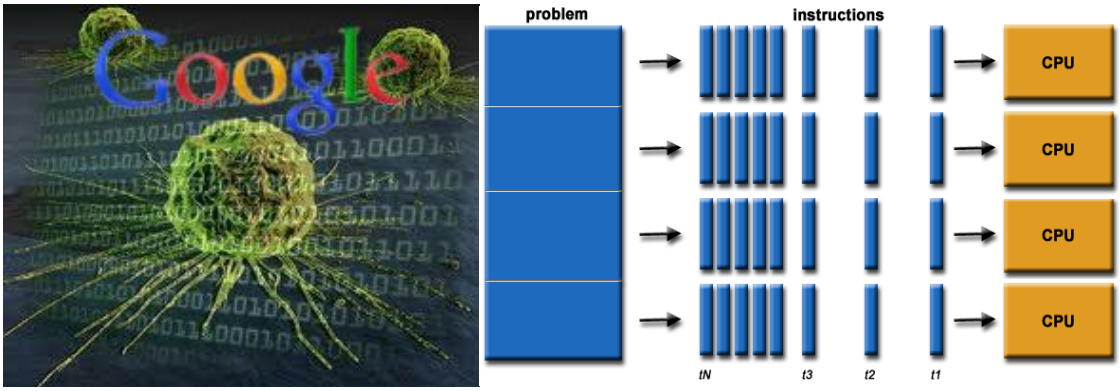
2.6. Findings

- When page rank values of dangling nodes are ignored, we were still able to get expected list of top 10 URLs. However sum of observed page rank values thus obtained was less
- When we considered dangling nodes into our calculation, sum of 0.9999999999999800 was obtained which is very close to ideal value
- We get good approximation of page rank algorithm after algorithm is ran for fixed number of times when convergence happens
- Minimum value of page rank could be (d-1)/N according to given formula when it has no inbound links
- Maximum value of page rank can be ((d-1)/N)+(d*(N-1)) when all remaining (N-1) nodes point only to current node
- If the page has more number of inbound links, then it results in increasing page rank of that page which is evident from given page rank calculation formula

2.7. Conclusion

We implemented page rank in sequential manner. However, performance and efficiency of this implementation decreases as we increase the number of input URLs in given graph. As a part of future work, we can implement this algorithm in parallel fashion which is much more time efficient than its sequential counterpart.

3. Phase 1-Part 2: Page Rank Algorithm using MPI



Source: http://blog.drchrono.com/2012/05/24/dresden-university-using-googles-search-engine-algorithms-to-fight-cancer/
https://computing.llnl.gov/tutorials/parallel_comp/

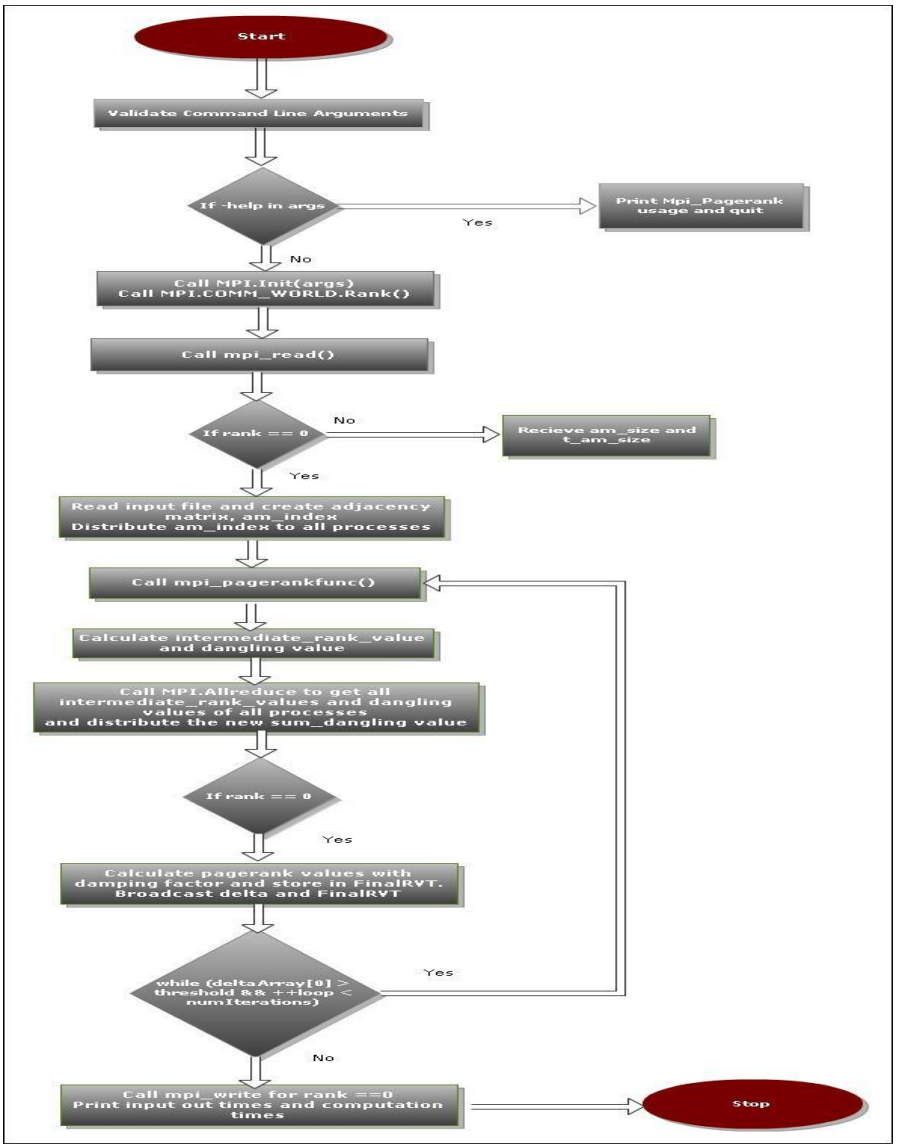
3.1. Introduction

In this phase we implemented MPI PageRank algorithm. The algorithm calculates the PageRank value of each page depending on the inbound links to that page. Each inbound link is like a vote. These votes are used to determine which pages are more important.

We implemented this algorithm using MPJ open source library. This algorithm is revision of first part in which page rank was implemented in a sequential manner. We used java for parallel page rank implementation because java is considered as excellent language for running high performance parallel applications on grid and clusters.

3.2. Architecture

The below diagram shows the high level architecture of MPI algorithm workflow



3.3. Optimization Techniques

Instead of going over for loop for number of iterations given from command line, we try to limit the iterations until convergence happens. Each time page rank values are calculated, we compare them against earlier page rank computations. If this difference goes below specific threshold we break the loop and return results of iteration in output file.

This algorithm is highly optimized compared to first phase sequential implementation. We divide input file data into number of subdivisions which is equal to number of processes to run in parallel. We then assign fixed number of input URLs to each subdivision. If the total number of URLs is not a multiple of number of input processes, we adjust remaining processes into last subdivision. So last subdivision might contain more number of URLs than rest ones. All the processes run concurrently.

We use MPJ library function broadcast to broadcast messages from root process to all the other n processes created which are responsible for individual page rank evaluation. Reduce is used to combine values from all processes and distributing result back to each individual process.

3.4. Results

List of Top 10 URLs was accumulated in output file along with their page rank value as shown in following page,

Top 10 URLs with Highest Page Rank values

URL	Page Rank
4	0.13809182637339654
34	0.12293720129122925
0	0.11259168048995331
20	0.07746633682627928
146	0.05723789230252064
2	0.04795732052423655
12	0.02010149462956356
14	0.01791527501372222
16	0.01303327335882033
6	0.01294396513791297

Cumulative Sum of Page Rank values

0.9999999999999813

3.5. Findings

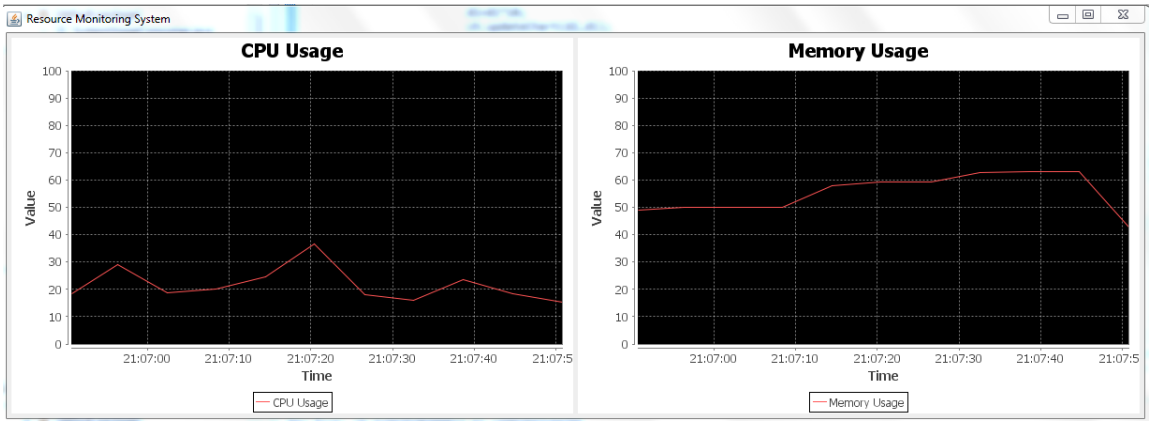
We successfully implemented MPI implementation of page rank algorithm which gives the top 10 URLs in terms of the page rank from given list of N input URLs. Following put forward our findings as follows

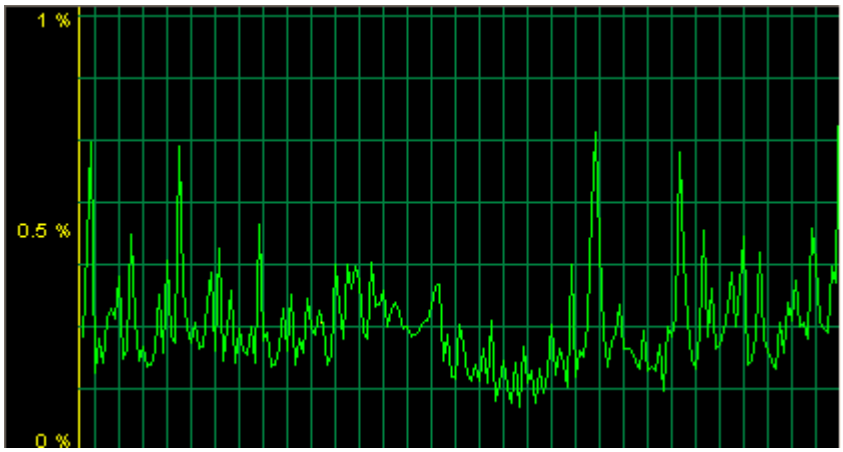
- When we considered dangling nodes into our calculation, sum of 0.9999999999999813 was obtained which is very close to ideal value (Ideal value was set to 1)
- It was observed that, it is not at all necessary to run page rank algorithm for any number of times because in some cases if node configuration is ideal, convergence occurs even before specified number of iterations
- For fixed values of threshold and iteration count of 50, convergence occurred only at 14th iteration
- We get good approximation of page rank algorithm after algorithm is ran for fixed number of times when convergence happens

3.6. Conclusion and Future Work

These algorithms when tested on file containing N=1000 URLs, gave excellent results in terms of final list of top 10 webpages due to its parallel nature (using MPI). Next part of this project is to run parallel page rank version on Multi core Supercomputer Future grid provided by Indiana University.

4. Phase 2 - Resource Monitoring System





Source: http://technet.microsoft.com/en-us/library/bb456996.aspx

4.1. Introduction

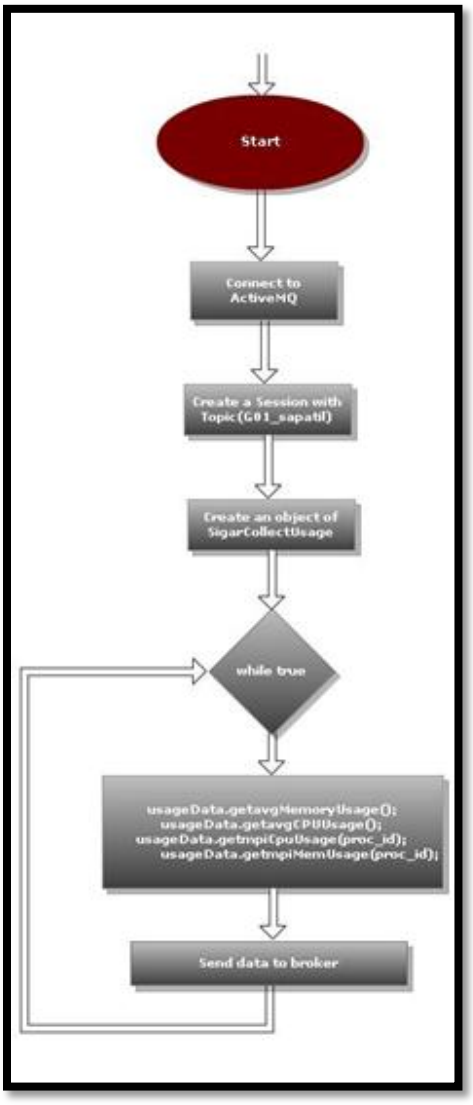
Aim of this project phase is to monitor resource consumption on distributed system. However, we make use of external APIs to perform message communication and resource utilization calculation.

In the first project, we implemented parallel version of page rank algorithm. We also performed performance analysis on earlier version of Page Rank algorithm.

However, no analysis was done on how it burdens operating system in terms of memory and CPU utilization. In this project we analyse how it affects resources in the underlying system.

4.2. Modules

4.2.1. Producer/Publisher

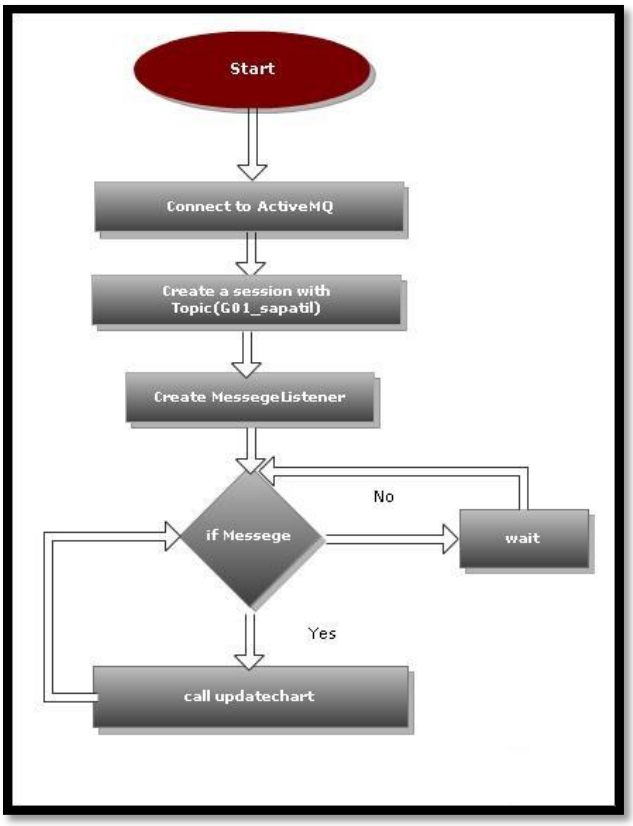


Description:

Producer runs on a remote machine which is responsible for monitoring how MPI page rank algorithm utilizes system resources in terms of memory and CPU utilization. Producer keeps on sending message to queue which is given unique id with respect given pair of publisher and subscriber.

Producer gathers system resource utilization using SIGAR APIs for both system and MPI page rank execution appends output as a string and sends to receiver using ActiveMQ API send().

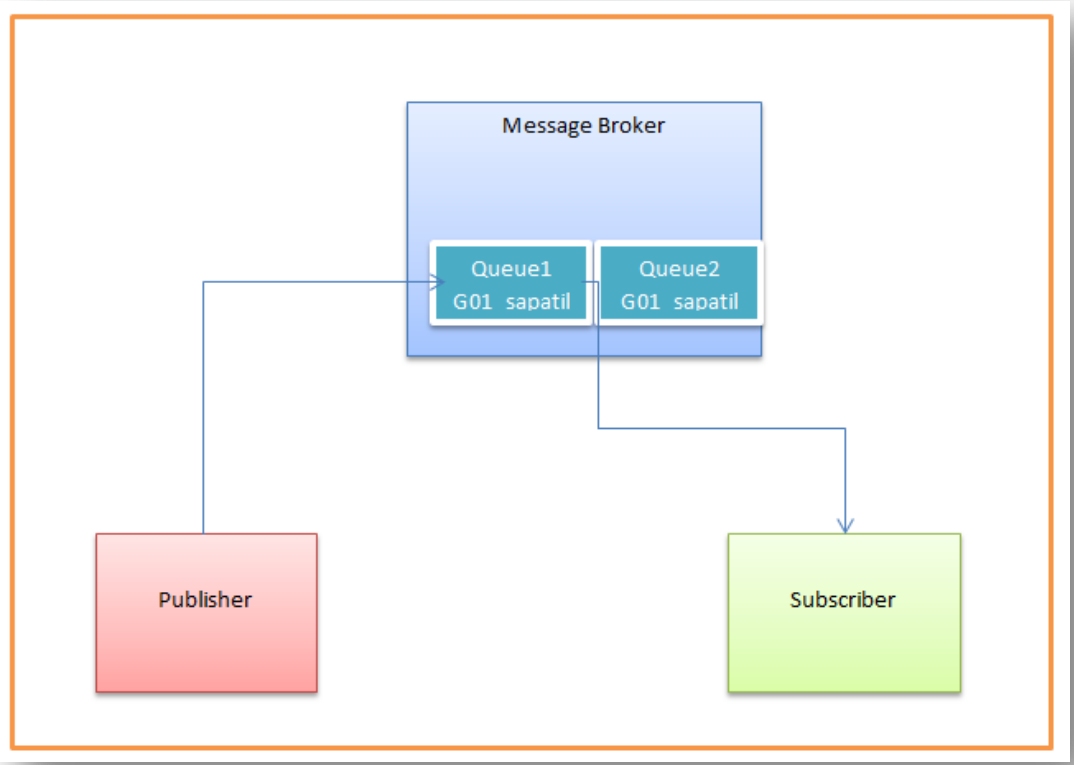
4.2.2. Consumer/Subscriber



Description

Consumer lies on a local machine. It is also called as subscriber because it has already been subscribed to specific queue where it accepts message from broker which in turn receives message from publisher.

4.2.3. Broker Architecture



Description:

The message broker is a middleware which acts as an interpreter between subscriber and publisher. It is responsible for rule conversion. Whenever two heterogeneous systems want to talk with each other using message queuing model, they can experience problem in interpreting syntax and semantics of the messages thus sent.

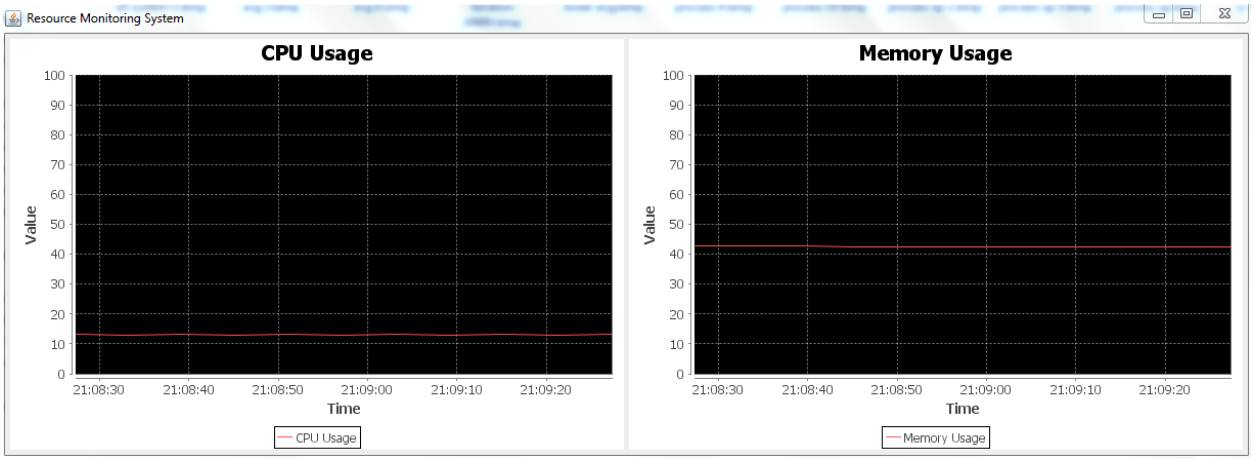
Broker is a collection of conversion rules and program which makes appropriate conversion before storing and retrieving messages into and from respective queues.

4.3. Results:

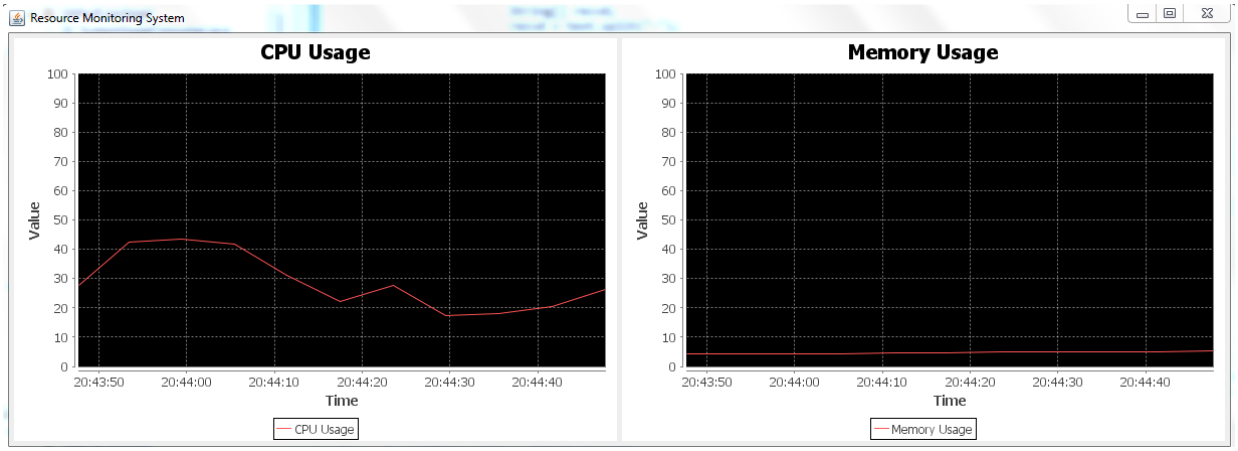
We observed system CPU and memory utilization in the scenario when MPI page rank algorithm is running and other one where it is under normal situation when MPI page rank process is not running.

We observed substantial difference when MPI page rank algorithm was running on a machine. Following section describes behaviour of system as it goes through different phases.

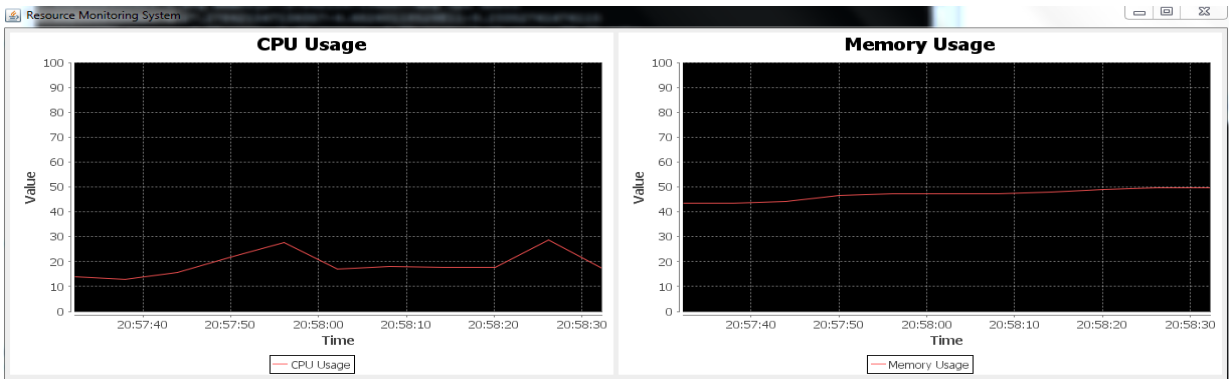
Case 1: When system is Idle – No MPI page rank is running



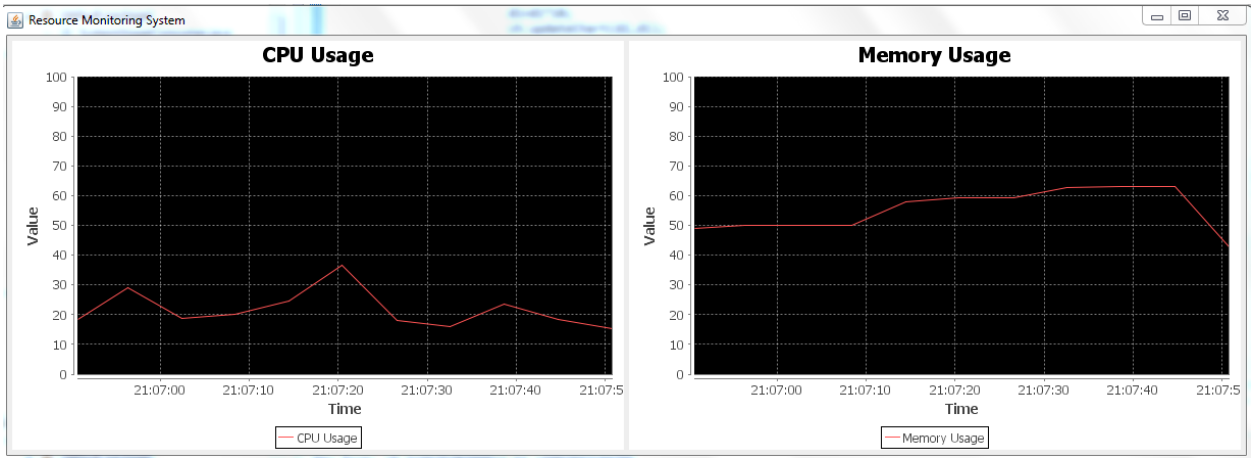
Case 2: When MPI- Page rank algorithm is running



Case 3: Average system utilization – Page rank algorithm



Case 4: Average system CPU and memory utilization when page rank algorithm is running

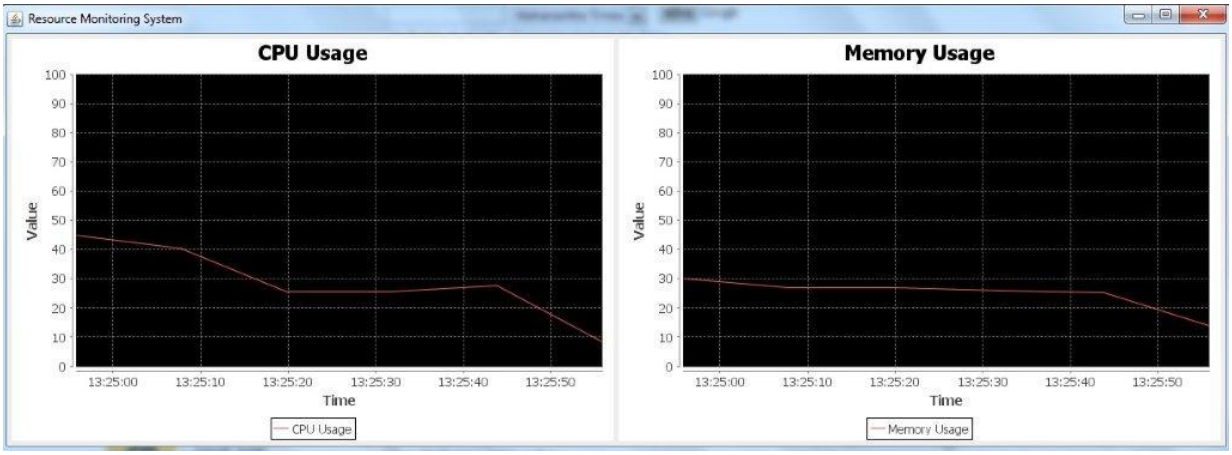


4.4. **Performance analysis with synchronization using machine names**

(Taking average of CPU and memory utilization from two nodes/machines)

As seen in earlier graphs, we can see spikes in final result. This is because we have directly plotted resource utilization values received from two different nodes on which page rank algorithm is running.

To compensate with spikes, we ran our code on nodes molerat.cs.indiana.edu and wolverine.cs.indiana.edu. We stored values of CPU and memory utilization in different variables and as soon as we received values from both the nodes (which is indicated by two flag variables for each process), average is taken and it is plotted in a graph.



4.5. Conclusion and Future Work

This resource monitoring system was tested on distributed set of nodes on which PageRank algorithm is running. The system gives the visualization of overall CPU and memory utilization of nodes and also of a process running PageRank algorithm.

Next part of this project is to run resource monitoring system on Multi core Supercomputer and analyse its CPU and memory utilization where it actually utilizes multiprocessor power for which it was designed.

5. Phase 3 - PageRank Performance Analysis on Academic Cloud

Source: http://www.contrib.andrew.cmu.edu/~aishah/CC.html

5.1. Introduction

In this project we are running Page rank algorithm on FutureGrid academic cloud. FutureGrid is a countrywide NSF (National Science Foundation) sponsored high performance distributed project to run parallel, grid and cloud based applications.

FutureGrid can be considered as a test-bed that allows many distributed system researchers to run their experiments with the help of complex workflow system. Main goal of this project is to run Page rank program thus developed on FutureGrid in two different modes. First one is FutureGrid bare metal and other is FutureGrid Eucalyptus which is virtual infrastructure management tool to access and use virtual system resources.

5.2. Project setup

5.2.1. We performed the experiments using the datasets of 1K, 10K, 50K, 100K, 150K, 200K, 250K, 300K, 350K, 400K, 450K, 500K, 1 million and 2 million URLs on Bare Metal

Parameters	Bare Metal
Worker Nodes	2 Nodes (8 processes on each node)
Datasets	1K, 10K, 50K, 100K, 150K, 200K, 250K, 300K, 350K, 400K, 450K,500K, 1 million and 2 million URLs
Number of Processes	2,4,6,8,10,12,14,16
Threshold	0.000001
Iterations	50,100

5.2.2. We also performed these experiments using the datasets of 10K, 50K, 150K, 200K, 250K, 300K, 350K, 400K, 450K, 500K, 1 million URLs on Eucalyptus VM

Parameters	Eucalyptus VM
Worker Nodes	2 VMs (2 cores/processes on each VM)
Datasets	10K, 50K, 150K, 200K, 250K, 300K, 350K, 400K, 450K,500K, 1 million URLs
Number of Processes	2,3,4,5,6,7,8,9,10,11,12,13,14
Threshold	0.001, 0.000001, Varying
Iterations	50,100, Until convergence

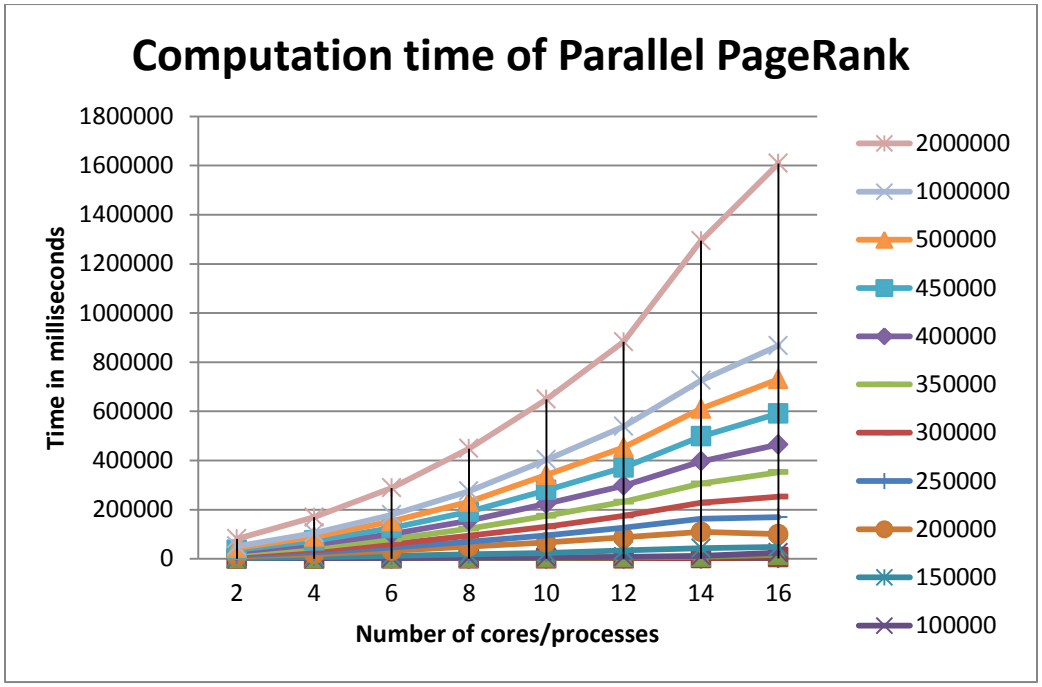
5.3. Results

5.3.1. *Results on Bare Metal*

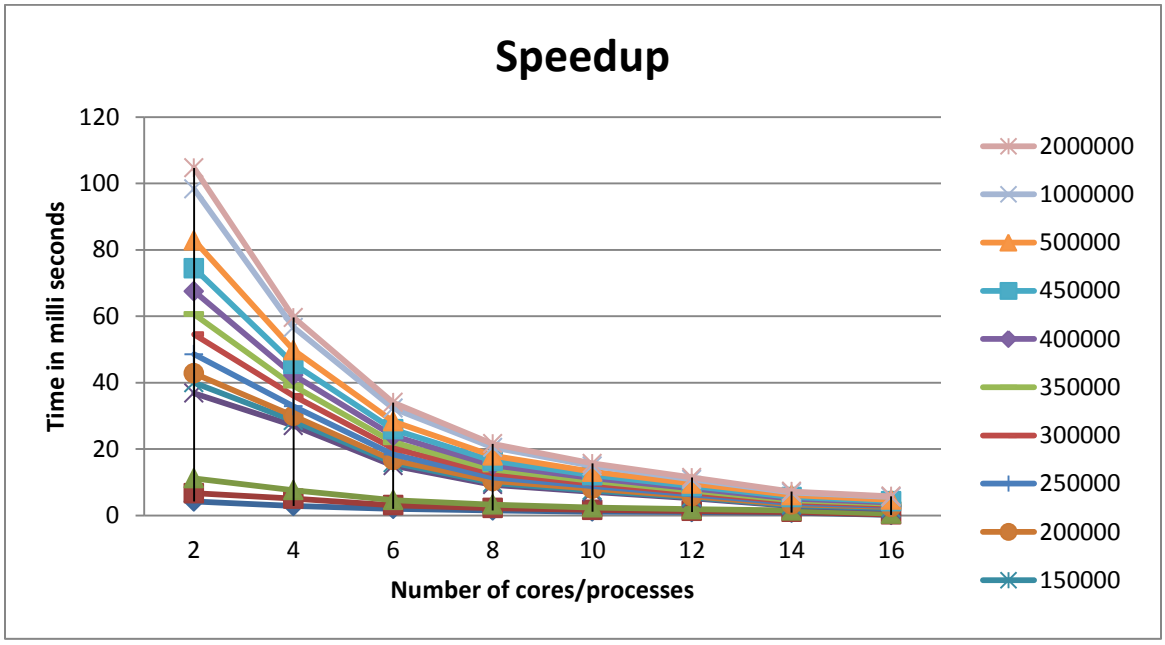
	Number of Processes							
File Size	2	4	6	8	10	12	14	16
1000	64.33333333	94.33333333	138	178.666667	230.6666667	269	334.3333	1711.333
10000	181.6666667	204.3333333	407	632.666667	889.6666667	1279	1727.667	3979.333
50000	556.6666667	977	1593	2415.33333	3452.666667	4671.333	6079.667	9788.667
100000	204.3333333	270.3333333	506.6666667	873	1117.333333	1599.333	3897.232	9552.667
150000	2499	5190.333333	9102.666667	13871.6667	18154.33333	25945.33	31170.33	23002
200000	5993	11971	20804	31561.6667	42405.33333	53380.67	66207	52506
250000	3903.666667	7707.666667	13153.33333	20123.6667	29048.33333	39587.33	53051.33	68477.33
300000	4736.333333	9101.333333	15102.66667	24130.6667	35799.66667	48086.33	65363.67	83286.33
350000	5499	10983.33333	18680	28457	43196.33333	56883.67	78373.67	99611
400000	6010.333333	12813	21432.33333	32478.3333	49510.33333	65821.33	90556.33	112768.7
450000	6896.666667	14213	24196.33333	36615	55634	73445	101346.7	126136
500000	7684.666667	15528.66667	27050.33333	40661	61914.33333	82001.67	112752.7	139936
1000000	6919.333333	15631	27092.33333	43199.3333	61806.66667	85256.33	114999.3	136282.3
2000000	30369.66667	64434	109371.3333	173459	246102.6667	344282.7	568083.7	741261.3

Table 5.3.1

5.3.1.1. *Graphical Representation of Computation time for Parallel PageRank Algorithm on FutureGrid Bare metal*

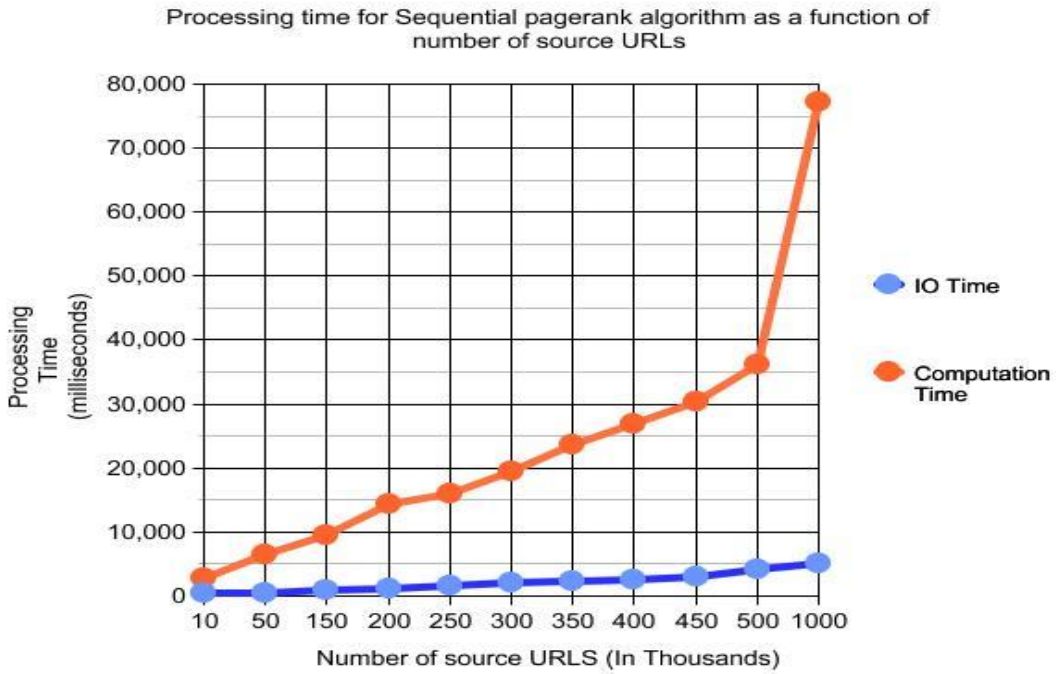


5.3.1.2. *Graphical Representation of Speed up on FutureGrid Bare metal*

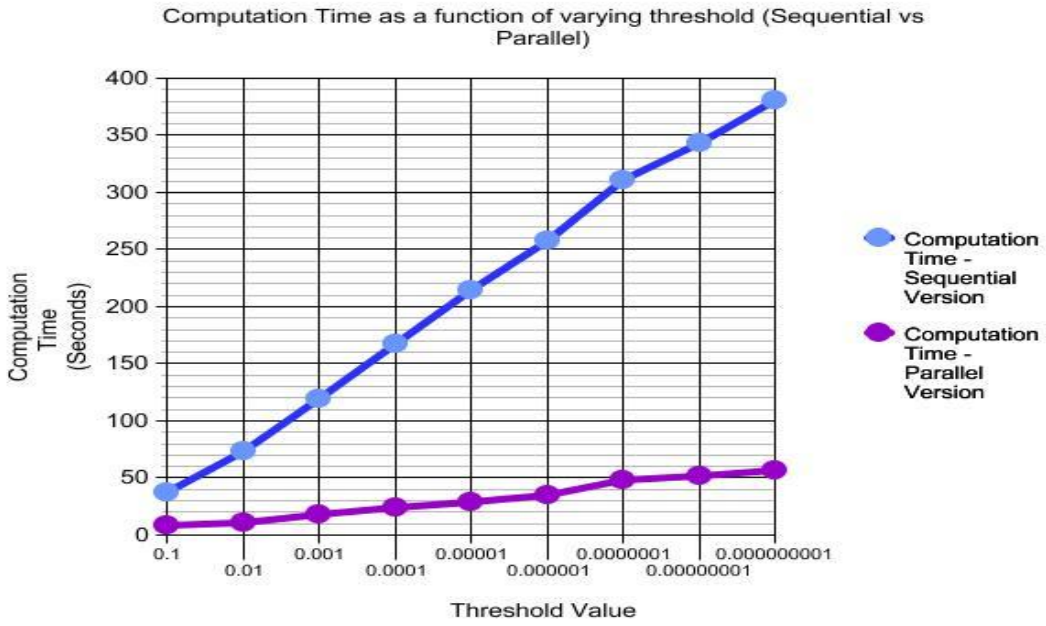


5.3.2. Graphical Representation of Speed up on FutureGrid Eucalyptus virtual machine environment

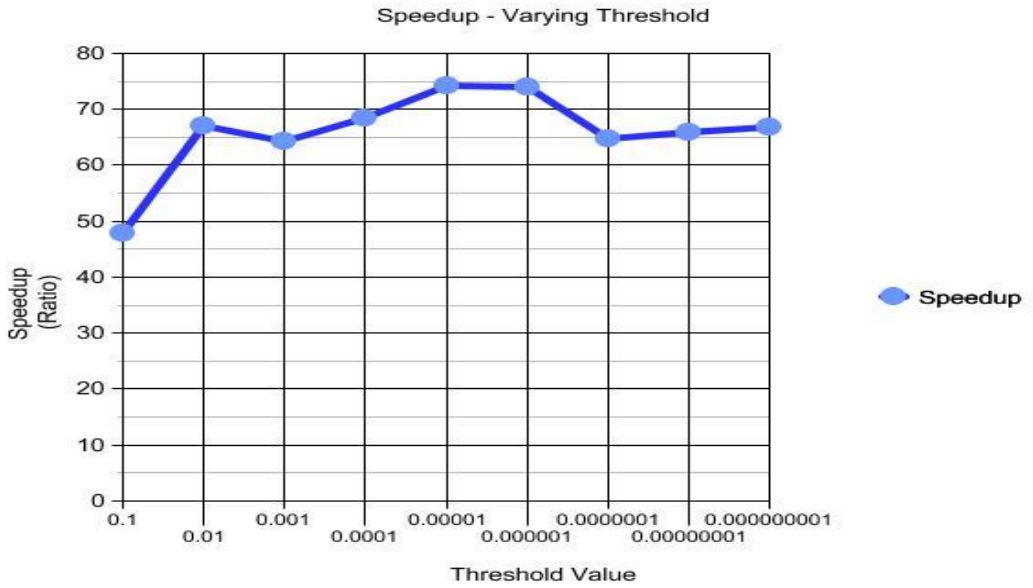
5.3.2.1. Computation time comparison – Parallel vs. Sequential Page rank implementation



5.3.2.2. Effect of varying threshold on computation time – Sequential and MPI implementation (Fixed input file)

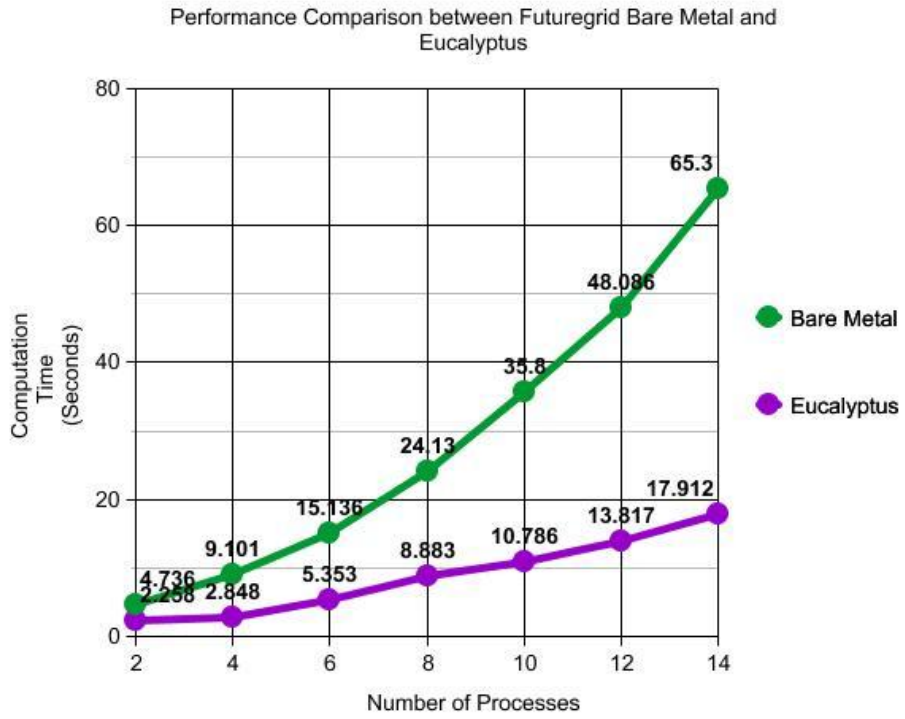


5.3.2.3. Speedup observed in MPI page rank algorithm – varying threshold (Fixed input file)



5.3.3. Performance Comparison between FutureGrid bare metal and Eucalyptus

Following graph shows execution time for parallel PageRank algorithm on both bare metal and Eucalyptus VMs. (Varying number of processes with input file of size 300000 URLs and threshold value of 0.001)



5.4. Findings and Observation

- We observed that, as we increase the number of processes, speed up is decreasing. Even when we varied the parameters, such as the number of worker nodes, threshold, iterations etc. we found the decrease in speed up.
  This drop in speed up as the number of processes increase could be due to the overhead involved in inter-process communication when number of processes increase.

- We also observed that, the difference in computation time was not significant when we varied the number of iterations from 50 to 100 as values converged within 40-45 iterations. Also, we observed a very slight difference in computation time when we varied the threshold.

- We observed a significant difference in computation time for the same input file with same parameters which are run different days or may be at different times of the day.

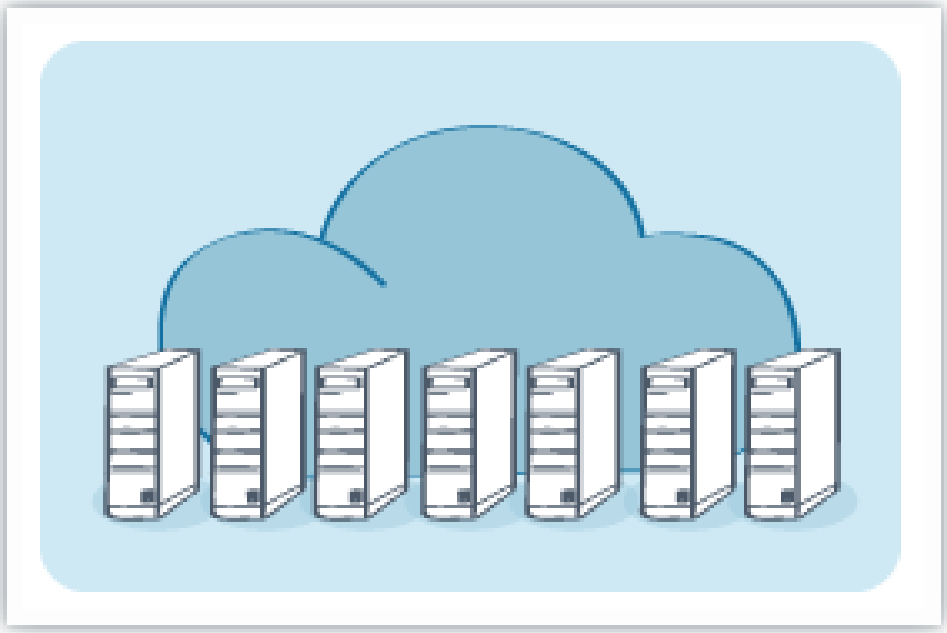
- When tested against input files of different sizes varying from 10000 to 1 million, significant improvement was observed in page rank computation time. For file of maximum size, sequential algorithm took almost 80 seconds, while MPI version finished just under 3 seconds.

- Speedup was calculated using same range of input files mentioned in Observation 4. Minimum speedup observed was 11.7 while maximum value of 26 for file size of 1 million URLs. This observation suggests that, MPI page rank algorithm performs well as size of input file increases.

5.5. Future work

Thus, we successfully analysed PageRank performance on academic cloud provided by Indiana University. Next part of this project is to dynamically switch clusters on academic cloud. Which includes running resource monitoring system on FutureGrid machine in both Bare Metal and Virtual Machine environment and analyse resource consumption in terms of CPU and Memory usage.

6. Final Phase - Dynamically switch/provision clusters on Academic Cloud



Source: http://www.zend.com/en/products/server/multi-server-support

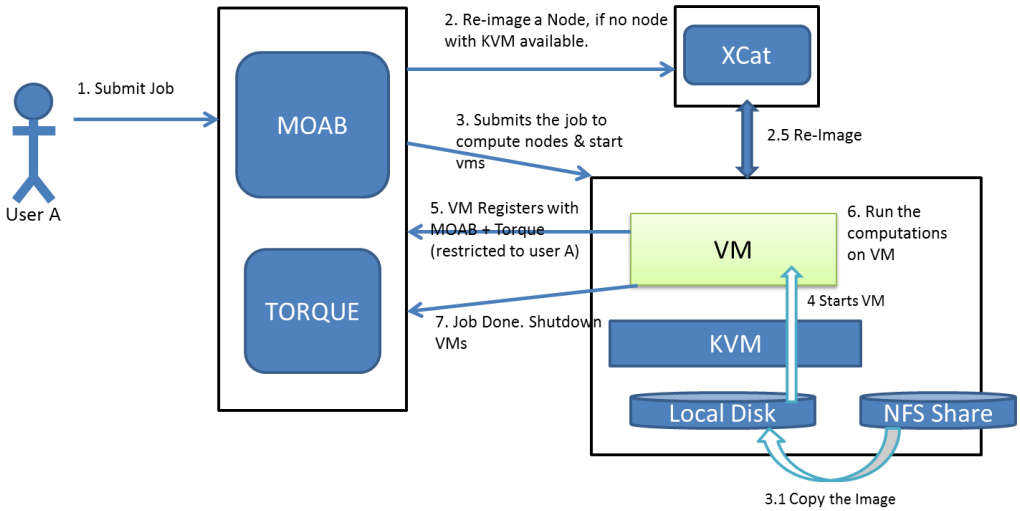
6.1. Introduction

Aim of this final phase is to monitor resource consumption on Dynamic provisioned cluster which is running with 2 physical and 2 virtual nodes. However, we make use of external APIs to perform message communication and resource utilization calculation.

In the first project, we implemented parallel version of page rank algorithm. However, no analysis was done on how it burdens operating system in terms of memory and CPU utilization. As a part of third project, we analysed system performance on non-cloud department systems.

However as a part of this project we will be monitoring memory and CPU utilization on academic cloud which is essentially a FutureGrid Dynamic provisioning infrastructure which provides Bare Metal and Virtual machine environments to run out program in.

6.2. System Architecture



Source: Project description document

Description:

Diagram shows interaction between different system components. We are assigned a special queue as b534. This queue holds all the nodes assigned for this infrastructure. Also as per the schedule, list of nodes (i70, i71 and i72) were assigned to our group.

As shown in figure, submits his job script to acquire nodes start respective VMs. Then we run publisher and page rank algorithm on assigned nodes. To perform actual implementation, virtual machine loads the system image from shared NFS area. Through the use of job script, VM is restricted to the user who is running that script. After finishing running jobs, script automatically shuts down respective VMs.

6.3. Scripts

6.3.1. Script description - Bare Metal Environment

1. Request assigned nodes on Bare Metal
2. Specify parameters such as total number of processors per node and clock time for which nodes are requested.
3. Specify the file to redirect the output on screen in
4. Submit the job script and wait until resources become available
5. Run resource monitoring application on each of the two assigned nodes
6. Start MPJ daemon
7. Run MPJ PageRank algorithm on both assigned nodes.
8. After finishing running program, release the nodes and free other resources thus held

6.3.2. Script description - Virtual Machine Environment

1. Request assigned nodes on Virtual Machine
2. Specify parameters such as total number of processors per node and clock time for which nodes are requested.
3. Stage and start Virtual Machine on all the assigned nodes (i70 and i71)
4. Wait for Virtual Machine resources to become available
5. Run resource monitoring application on each of the two assigned nodes
6. Start MPJ daemon
7. Run MPJ PageRank algorithm on both assigned nodes.
8. After finishing running program, shut down acquired VM's

6.4. Improvements introduced in Final phase

There are two major changes we are doing in this project which makes it different from the project we implemented as a part of second project.

6.4.1. Running PageRank algorithm in cloud environment

We are running our parallel PageRank algorithm on academic FutureGrid dynamic provisioning infrastructure to provision between Bare Metal and Virtual Machine environments.

As a part of second project, we ran it on departmental machine where no provision was provided for cloud environment. We ran earlier project on two departmental machines named molerat.cs.indiana.edu and wolverine.cs.indiana.edu

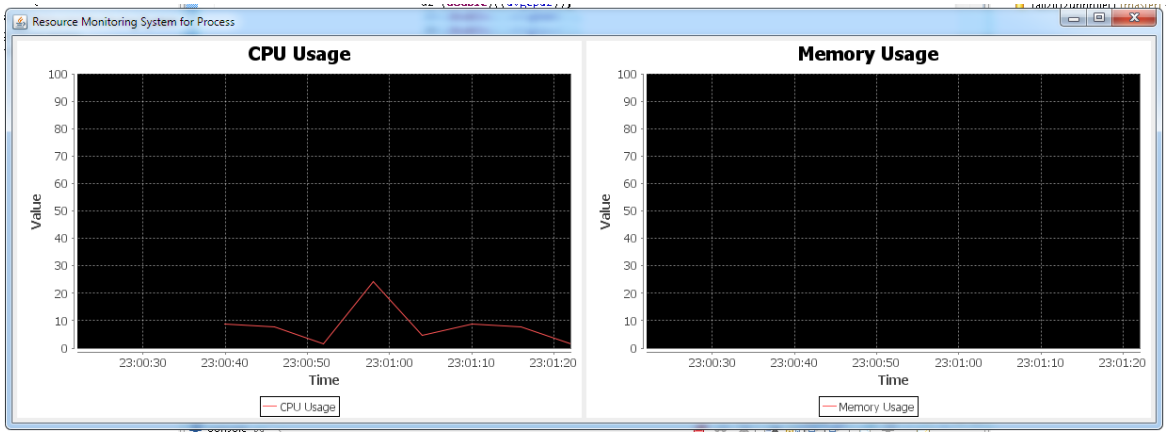
6.4.2. *Job automation on Bare Metal and Virtual Machine using shell scripts*

We ran earlier project by manually entering system commands. As a part of this project, we created job scripts and executed parallel PageRank algorithm and producer on academic cloud without manual intervention.

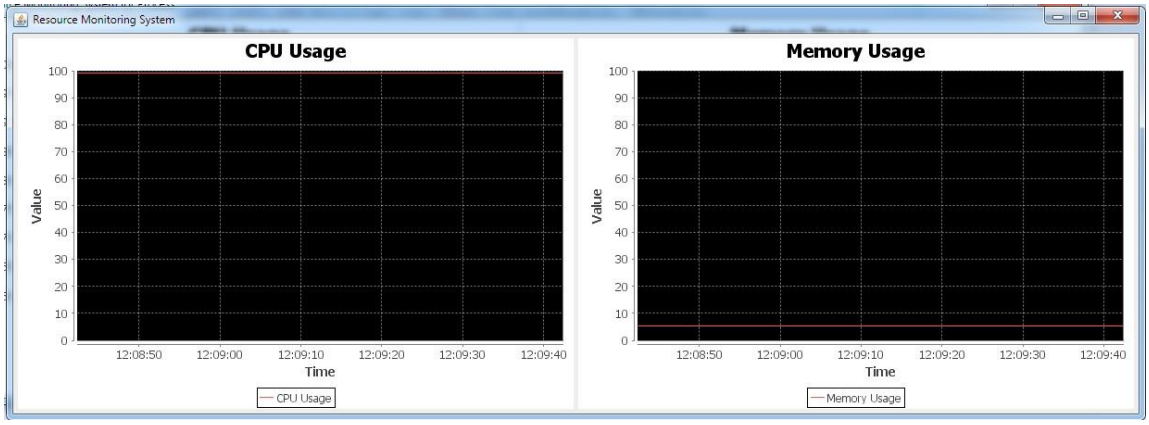
Using automated script made our work simple by, obtaining assigned nodes, running publisher and page rank algorithm on those nodes and releasing acquired resources once job is done.

6.5. Results

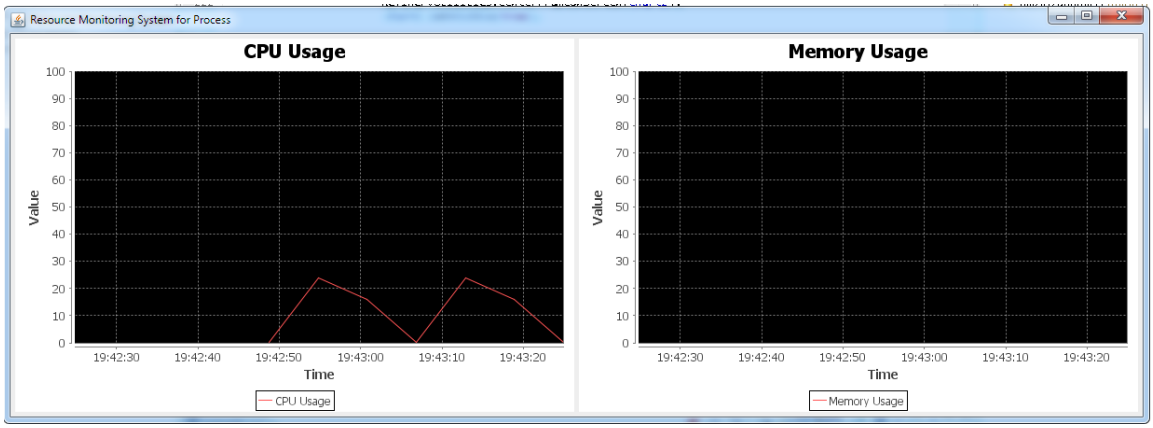
Case 1 – Bare Metal Environment



Case 2 – Virtual Machine environment



Case 3 – On Bare Metal addition of synchronization with Timestamp



6.5.1. Synchronization

Instead of directly plotting CPU and memory resource utilization values received from two different nodes on which page rank algorithm is running we perform synchronization on the messages received from both nodes. To perform synchronization, we are utilizing machine names and timestamp of message sent from publisher to subscriber.

We stored values of CPU and memory utilization in different variables and as soon as we received values from both the nodes (which is indicated by two flag variables for each process), average is taken and it is plotted in a graph.

6.6. Findings

Thus we have written scripts to automate the job of running resource consumption tool as well as page rank algorithm on distributed nodes.

In summary, we can say purpose of the automated scripts is as follows

- Requesting assigned nodes on Bare Metal
- Specifying parameters such as total number of processors per node and clock time for which nodes are requested.
- Submitting the job script and waiting until resources become available
- Running resource monitoring application on each of the two assigned nodes
- Starting MPJ daemon
- Running MPJ PageRank algorithm on both assigned nodes
- After finishing program, releasing the nodes and free other resources held in the past

6.7. Difference between two Bare Metal and Virtual Machine environments

In case of Bare Metal environment, we are acquiring actual nodes and specific number of processors on it. This is like acting directly on hardware.

In case of virtual machine environment, we are not dealing with real world system infrastructure. This system is based on the use of system image. Once we place the request to get system image from NFS shared area, it will load the image on local disk, load virtual machine environment from the image. This image is assigned to specific user. After successful load, we run resource monitoring tool and PageRank algorithm on it and perform required computation. Once we are done with job, we shutdown acquired VMs and unload the image.

7. Conclusion

Thus we implemented page rank in sequential manner. In the next phase, we ran parallel page rank version on Multi core Supercomputer Future grid provided by Indiana University and analyse its performance where it actually utilizes multiprocessor power for which it was designed.

Also in the next phase of this semester long project, the resource monitoring system was tested on distributed set of nodes on which PageRank algorithm is running. The system gave the visualization of overall CPU and memory utilization of nodes and also of a process running PageRank algorithm.

In the next part, we ran resource monitoring system on Multi core Supercomputer Future grid provided by Indiana University and analyse its CPU and memory utilization where it actually utilizes multiprocessor power for which it was designed.

In the final part of project, we made use of FutureGrid dynamic provisioning infrastructure to switch between Bare Metal environments. Used message passing based monitoring system to monitor CPU/ Memory usage and overall performance on the dynamic provisioned cluster.

We also automated the process of acquiring resources, running PageRank algorithm, resource monitoring tool and releasing resources on respective nodes after job is done.

8. Acknowledgements

We would like to thank Prof. Judy Qiu, Assistant Professor of Computer Science and Informatics at School of Informatics and Computing at Indiana University for helping us to develop an idea of this wonderful system. Presentation slides of lecture notes and lab material provided on course web page helped us to get quick start on this project as we got deep insight about how distributed and cloud systems actually work.

We also appreciate the help from Ila Jogaikar, for her patience and co-operation throughout this project. We specially thank AI Stephen Wu for his cooperation through the semester and especially help to resolve errors in FutureGrid environment while deploying and running code.

We would also like to thank all those anonymous people all around the internet, who shared their knowledge and experience on various blogs and webpages. This helped us to know more about various APIs that we used in our program implementation.

Last but not least, thanks to all those technicians at FutureGrid infrastructure who provided us with advanced cloud system infrastructure to run and analyse PageRank algorithm.

9. References

- http://en.wikipedia.org/wiki/PageRank
- http://www.webworkshop.net/pagerank.html
- http://www.sirgroane.net/google-page-rank/
- http://pr.efactory.de/e-pagerank-algorithm.shtml
- http://infolab.stanford.edu/~backrub/google.html
- http://mpj-express.org/
- http://en.wikipedia.org/wiki/PageRank
- http://www.webworkshop.net/pagerank.html
- http://www.sirgroane.net/google-page-rank/
- http://pr.efactory.de/e-pagerank-algorithm.shtml
- http://infolab.stanford.edu/~backrub/google.html
- ActiveMQ, http://activemq.apache.org/
- Sigar Resource monitoring API,http://www.hyperic.com/products/sigar
- http://sourceforge.net/projects/sigar/
- JFreeChart: http://www.jfree.org/jfreechart/
- MPJ Linux user guide, http://mpj-express.org/docs/guides/linuxguide.pdf
- http://sourceforge.net/projects/sigar/
- Publish/Subscribe Wikipedia: http://en.wikipedia.org/wiki/Publish/subscribe
- JMS Wikipedia: http://en.wikipedia.org/wiki/Java_Message_Service
- ActiveMQ Wikipedia: http://en.wikipedia.org/wiki/Apache_ActiveMQ
- ActiveMQ HelloWorld Example: http://activemq.apache.org/hello-world.html
- https://portal.futuregrid.org/hardware/india
- http://www.webworkshop.net/pagerank.html
- https://portal.futuregrid.org/tutorials/eucalyptus
- Presentation on Eucalyptus on FutureGrid by Archit Kulshrestha, Andrew J. Younge, Gregor von Laszewski, Geoffrey C. Fox
- Eucalyptus White Paper: Eucalyptus Open-Source Cloud Computing Infrastructure http://www.eucalyptus.com/pdf/whitepapers/Eucalyptus_Overview.pdf
- ActiveMQ, http://activemq.apache.org/
- Sigar Resource monitoring API, http://www.hyperic.com/products/sigar
- http://sourceforge.net/projects/sigar/
- JFreeChart: http://www.jfree.org/jfreechart/
- MPJ Linux user guide, http://mpj-express.org/docs/guides/linuxguide.pdf
- http://sourceforge.net/projects/sigar/
- Publish/Subscribe Wikipedia: http://en.wikipedia.org/wiki/Publish/subscribe
- JMS Wikipedia: http://en.wikipedia.org/wiki/Java_Message_Service
- ActiveMQ Wikipedia: http://en.wikipedia.org/wiki/Apache_ActiveMQ
- ActiveMQ HelloWorld Example: http://activemq.apache.org/hello-world.html
- http://www.open-mpi.org/
- Project description document