|  |
| --- |
|  |
|  |
|  |

**1.Introduction to Distributed systems**

Tanenbaum defines a distributed system as a “*collection of independent computers that*

*appear to the users of the system as a single computer*.” There are two essential points in this

definition. The first is the use of the word *independent*. This means that, architecturally, the

machines are capable of operating independently. The second point is that the software

enables this set of connected machines to appear as a *single computer* to the users of the

system. This is known as the **single system image** and is a major goal in designing

distributed systems that are easy to maintain and operate.

A distributed system may have a common goal, such as solving a large computational problem. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users. Other typical properties of distributed systems include the following:

* The system has to tolerate failures in individual computers.
* The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program.

**Why build them?**

Just because it is easy and inexpensive to connect multiple computers together does not

necessarily mean that it is a good idea to do so. There are genuine benefits in building

distributed systems:

Price/performance ratio: You don't get twice the performance for twice the price in

buying computers. Processors are only so fast and the price/performance curve

becomes nonlinear and steep very quickly.

Distributing machines may make sense. It makes sense to put the CPUs for ATM cash

machines at the source, each networked with the bank. Each bank can have one or more computers networked with each other and with other banks.

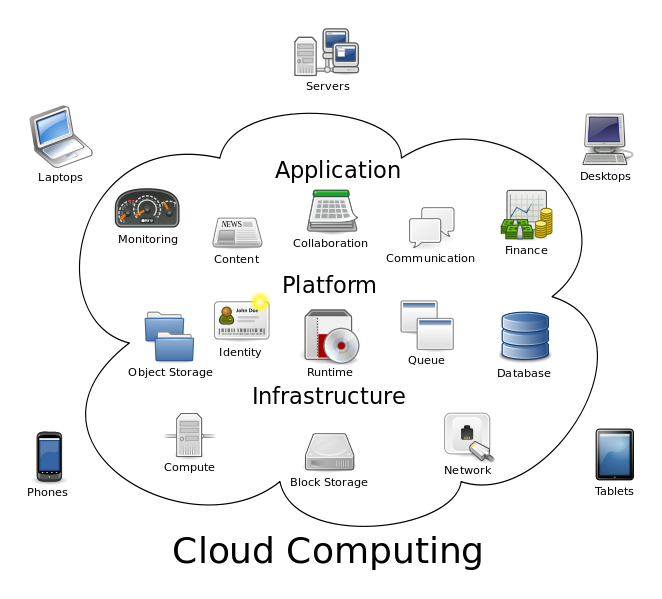
Computer supported cooperative networking. Users that are geographically separated

can now work and play together. Examples of this are electronic whiteboards,

distributed document systems, audio/video teleconferencing.

Increased reliability. If a small percentage of machines break, the rest of the system

remains intact and can do useful work.



**1.1 Aim**

With the emergence of the Net as a computing platform, distributed applications are being widely deployed by organizations. Understanding the principles/theory and the technologies underlying distributed computing and systems design is increasingly important. The Internet has greatly expanded the scope and importance of distributed systems to include Web 2.0 sites, Information retrieval (search), Utility (cloud) computing, P2P systems and the Internet of things. Further science is facing an unprecedented data deluge and the emergence of data oriented analysis as a fourth paradigm of scientific methodology after theory, experiment and simulation.

This course was outlined as follows:

* Algorithms – PageRank
* Synchronization – MPI
* Processes – Virtualization
* Communication – Messaging Middleware
* Synchronization – Lamport, NTP and SNTP
* Performance Analysis – Scaling

**1.2Scope**

In this course we have developed four projects, i.e. the sequential PageRank algorithm, the Parallel PageRank algorithm (MPI PageRank), Resource Monitoring System and a Provisioning

system that allows users to run and monitor parallel MPI program on Baremetal and Virtual machines which calculates the memory, CPU usage of the system as well as of the MPI program on the system.

In the Performance analysis on academic cloud, MPI PageRank computation was executed on two different environments Baremetal and Eucalyptus using the services provided by Futuregrid. Speed up on both the environments was calculated along with the analysis of the graph charts.

Dynamic provisioning provides the ability and possibility to use shared resources in a shared cloud environment. Client sends a request specifying their needs to resource manager to obtain computing resources. The requested resources are generated or instantiated dynamically.

1. **Projects:  
   2.1** **The Page Rank Algorithm**

Page Rank is a result of a mathematical algorithm based on web graph model where web pages are the nodes of the graph and hyperlinks are the edges in the graph. The rank value of a page indicates the importance of that page. Each hyperlink to a page counts as a vote of support for that page. The page rank of a page is recursively calculated and it depends on the number and page rank metric of all the pages that link to it. If a page is linked to by many pages that have a high page rank the rank of the page itself will be high too. However, if there are no incoming links to a page then there is no support for that page.

The Page Rank of a webpage is calculated as follows:

PR (A) = (1-d) + d [PR (P1)/N (P1) + ... + PR(Pn)/N(Pn)]

Where,

PR (A) is the Page Rank of a page A.

PR(P1) is the Page Rank of a page P1.

N (P1) is the number of outgoing links from the page P1.

d is a damping factor the range of which is 0 < d < 1 and is set to 0.85.

In the general case, the Page Rank value for any page p can be calculated as :

PR(p)=(1-d)/N+d\* ∑\_(q∈Set)▒(PR(q))/(L(q))

Where,

L (q) is the out degree of each web page in the vertices set

PR (q) is the Page rank of q

d is the damping factor= 0.85 generally

N is the total number of the unique URLs.

**Architecture and Design –Pagerank Algorithm**



**2.2 MPI PageRank**

Developing parallel PageRank is an active research area for both in industry and academia and numerous algorithms have been proposed. The key idea in developing parallel PageRank is to partition PageRank problem into N sub problems so that N processes solve each sub-problem concurrently. One of simple approaches in partitioning is a vertex-centric approach. The graph of PageRank can be divided into groups of vertices and each group will be processed by a process. We take this approach for our MPI PageRank implementation.

MPIRank

static public void main(String[] args)

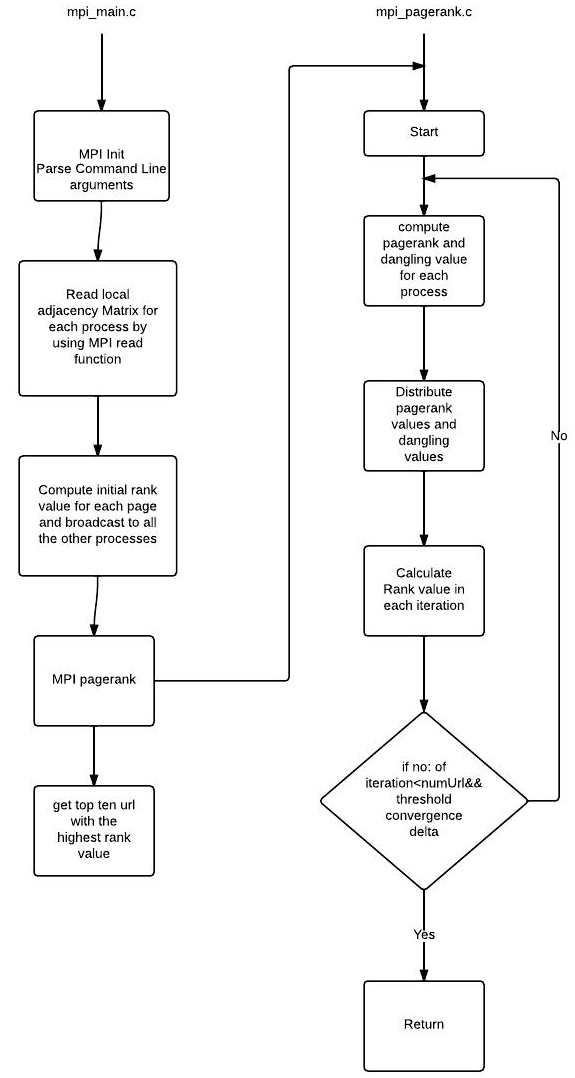
the main function which calls individual functions from the MPIFuntion class, and then sends over the results to a sort function in the same class.

MPI.Init( ) : This function initializes the MPI environment

MPI.Finalize ( ): This should be the last function called and as the name suggests this function finalizes the MPI environment.

public static long sort(double[] store\_rank, String output)

Sorts the results of the PageRank, and writes the top 10 rank values into an output file name.



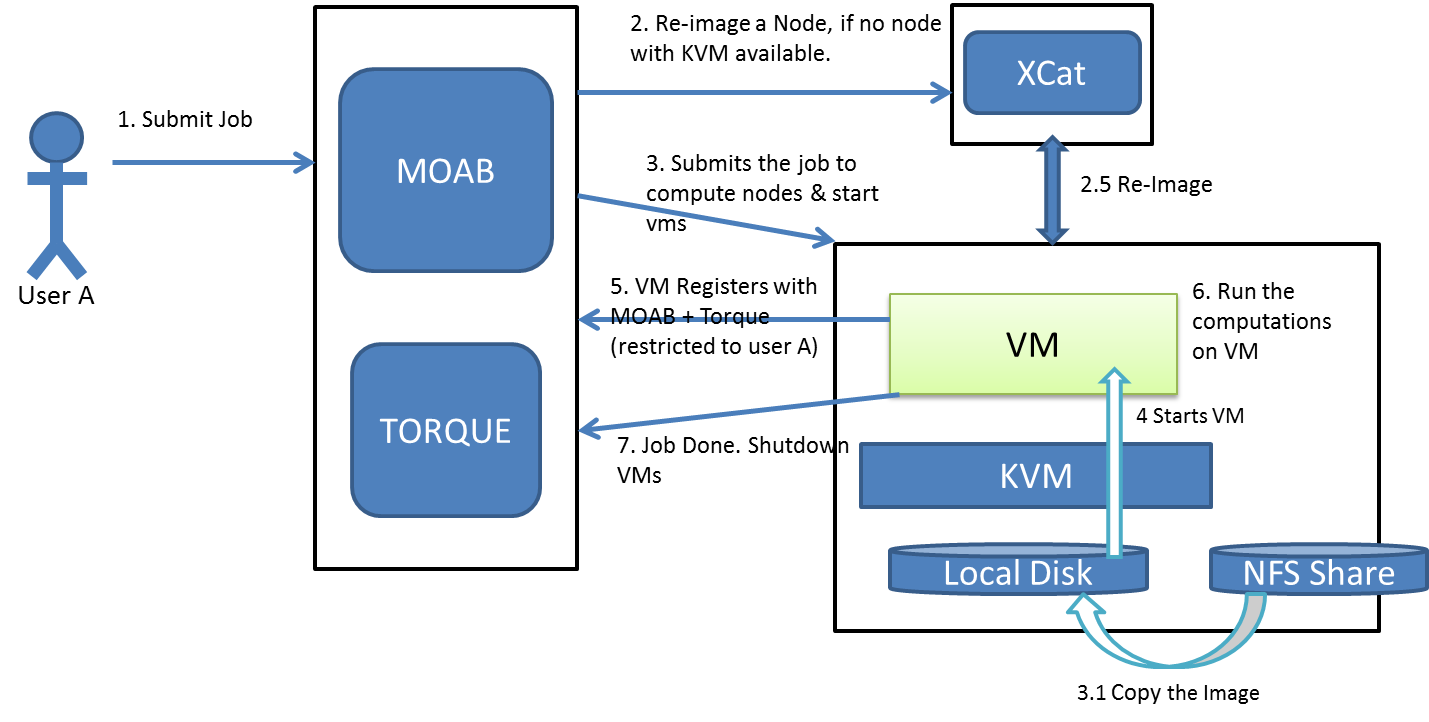
**2.3The Resource Monitoring Project**

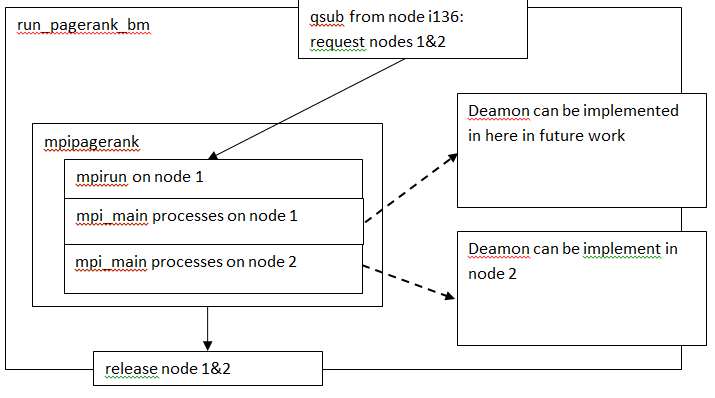
Resource monitoring in distributed systems is essential in order to manage the resources efficiently and to detect failures as well as to optimize the distributed application performance. We implement a system that monitors the CPU and memory utilization for a distributed set of nodes. We achieve this by having a producer that sends messages to the Broker which in turn routes these messages to the consumer and plot the CPU utilization and memory utilization percentages on that graph.

Message brokering is defined as an architectural pattern for message validation, message transformation and message routing. It acts as an intermediary thereby minimizing the awareness among the producer and consumer thereby enabling efficient message exchange between the two. One of the key things in understanding broker-based messaging is that the production, or sending of a message, is disconnected from the consumption of that message. The broker acts as an intermediary, serving to make the method by which a method is consumed as well as the route that the message has travelled orthogonal to its production. When a standard message is sent to a queue from a producer, it is sent to the broker, where the message remains in the message queue.

The architecture essentially consists of three modules- The Message Broker, Monitoring Daemon (Publisher) and a Monitoring UI (Consumer). In our project we use Narada Brokering or Apache ActiveMQ Messaging System as the broker.

**Fig .** User interactions with Dynamic provisioning system (borrowed from project instruction document)





**Fig.** shows overview of scripts “run\_pagerank\_bm” and “run\_pagerank\_vm”, performing dynamic provisioning on bare metal nodes and virtual machines, respectively. These two scripts are integrated into one single script

**2.4 Performance Analysis on the Academic Cloud**

The main aim of this project was to analyze the performance of the MPI-PageRank algorithm on the Academic Cloud. For analysis, we run the program on two different modes; the Bare Metal and the Eucalyptus VMs on the FutureGrid. Having observed, marked and analyzed the time reading on these, we represent them quantitatively by plotting charts for them. The charts denote differences in these readings and help us understand the gradient. Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with a user's data, software and computation. Cloud computing systems provide access to large amounts of data and computational resources through a variety of interfaces. On a cloud resources can be acquired and released on-demand and that the user interface is kept fairly simple. In addition, resources provided by cloud computing systems hide a great deal of information from the user through virtualization.

The sequential and parallel PageRank programs were tested on two environments on FutureGrid namely on bare metal and eucalyptus VM’s and performance analysis was done on these environments by calculating the speed up and also various experiments were performed by changing the parameters like number of processes, number of nodes and number of cores.

**Capturing Process Cpu and Memory Usage**

We used getCpuPrecList() function from Sigar getting the number of CPU first, and added up all usedCpuPercent on every CPUs (multicore) together then divide by the core number to get the average overall CPU usage. Sigar directly provide a getMem().getUsedPercent() function to get the overall memory usage.

In this project, in order to capture the process CPU and Memory usage, we have to consider the scenario of multiply processes running on multi cores. Sigar actually provide a function called getMultiProcCpu and getMultiProcMem to get the total CPU and Mem usage of processes with same process name. However, getMultiProcCpu function provided by sigar 1.6.4 behaves in Solaris mode not Irix mode, which means %CPU cannot be more than 100%, if default mode is irix, when there is only one CPU, result 20% means 20% of one CPU, if there are four CPUs the total %CPU can be 400%, but sigar hardcodes if (procCpu>1.0) procCpu =0.99 and using Solaris mode as default which may cause incorrect result on multicores.

So we changed to useProcCpu.getPercent() function get single process cpu usage via the pid list we got from Process finder function in Sigar and added all together. Fortunately, ProcCpu.getPercent() lib changed to use Irix mode since 1.6.2, we don’t have to worry about the multicore %CPU issues. However, this function is getting CPU usage via the time difference which makes us have to call this function twice to get data, otherwise, a single call returns zero usage, and also, the interval between function calls should be more 1000ms (In fact, on bare metal nodes and virtual machine, the interval needs to be 2000ms process average CPU usage that is not acceptable.

run\_pagerank\_bm

ProcCpuMem on node 1:

Sends message periodically Stops when there is no mpi\_main processes on node 1

qsub from node i136:

request nodes 1&2

ProcCpuMem on node 2:

Sends message periodically Stops when there is no mpi\_main processes on node 2

mpipagerank

mpirun on node 1

mpi\_main processes on node 1

mpi\_main processes on node 2

release node 1&2

run\_pagerank\_vm

qsub from node i136:

request nodes 1&2

start\_vms (v1 & v2)

wait\_for\_vms (v1 & v2)

shutdown\_vms

mpipagerank

mpirun on v1

mpi\_main processes on v1

mpi\_main processes on v2

ProcCpuMem on v1:

Sends message periodically Stops when there is no mpi\_main processes on v1

ProcCpuMem on v2:

Sends message periodically Stops when there is no mpi\_main processes on v2

release node 1&2

Fig 2. Overview of scripts. The dashed arrows indicate that the starting of one process precedes the starting of other process. The solid arrow indicate that the start of one process depend on the completion of the previous process. Note: the failures of some processes may cause crash or going to other states, and these cases are not included in this figure for the sake of simplicity; this figure represents for arbitrary number of nodes.

CONCLUSION:

We came across a lot of hurdles while implementing these projects but they have enhanced our knowledge about a distributed system in many ways. It has been a good learning experience implementing each of these projects namely- Sequential page rank, parallel page rank, resource monitoring and dynamic switch/provision clusters on Academic Cloud which bare great importance in the real world.

**ACKNOWLEDGEMENTS:**

We would like to express our gratitude and are thankful to Professor Judy Qiu for her valuable assistance which helped us complete this project successfully. We would also like to thank Stephen Wu for his guidance and support throughout this project. We feel it is the combined efforts of all that helped making this project a success. We would also like to that Team Futuregrid for letting us use their resources.

6. REFERENCES:

[1]. Eucalyptus tutorial on Future Grid

<https://portal.futuregrid.org/tutorials/eucalyptus>

[2]. Wiki Link for Message Broker:

<http://en.wikipedia.org/wiki/Message_broker>

[3]. Sigar API Souce:

<http://www.hyperic.com/products/sigar>

<http://www.hyperic.com/support/docs/sigar/org/hyperic/sigar/SigarProxy.html>

[4]. JfreeChart Source:

<http://www.java2s.com/Code/Java/Chart/JFreeChartDynamicDataDemo.htm>

<http://stackoverflow.com/questions/5048852/using-jfreechart-to-display-recent-changes-in-a-time-series>

[5]. Publisher-Subscriber Source:

<http://stackoverflow.com/questions/3032827/lightweight-publish-subscribe-framework-in-java>

[6]. Process Id Reference:

[http://www.coderanch.com/t/109334/Linux-UNIX/UNIX-process-ID-java-program](http://www.coderanch.com/t/109334/Linux-UNIX/UNIX-process-ID-java-program" \t "_blank)

[7]. MPJ Linux Guide:

[http://mpj-express.org/docs/guides/linuxguide.pdf](http://mpj-express.org/docs/guides/linuxguide.pdf" \t "_blank)

[8] Page Rank Wiki Page:

<http://en.wikipedia.org/wiki/PageRank>

[9] Google Page Rank:

<http://www.smashingmagazine.com/2007/06/05/google-pagerank-what-do-we-really-know-about-it/>

[10] Efficient parallel page rank algorithm:

<http://www.cs.ucsb.edu/~kmeeker/ppagerank_report.pdf>

[11] MPI Wiki Page

<http://en.wikipedia.org/wiki/Message_Passing_Interface>

[12] MPI Allreduce

<http://www.mpi-forum.org/docs/mpi-11-html/node82.html>

[13] MPI\_Bcast

<http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Bcast.html>