Performance Analysis of PageRank Algorithm on HPC architecture: Case study with Academic supercomputing platform and academic clouds

Swapnil Joshi, Vaibhav Nachankar and Harshad Joshi

Indiana University, Computer Science
Introduction

Distributed systems

Historically, computers were used only for complex Scientific and engineering problems. These computers utilized large computer clusters that engage several compute nodes to solve a particular problem. Issues of performance and benchmarking of these clusters were thus mainly limited to the select set of scientists and engineers.

The picture has changes with the birth of internet. With that, distributed systems are becoming ubiquitous and dominating our special as well as daily tasks. These include using mobile phones to booking of travel tickets to office works. Internet and internet-based computing can be found everywhere. Cloud computing is becoming another measure of success and has sparked many academic and commercial institutions to implement this platform for their work. It is thus important to understand the features and differences between the distributed systems and study their components. In this project we attempt to decompose and study in details two systems: academic cloud and academic bare-metal supercomputing platform.

Two popular systems: Bare-metal and cloud computing

Bare-metal platform is one of the oldest supercomputing platform which is formed by joining compute nodes via a interconnect communication switch. The speed of the computation thus not only depends on the computing capabilities of each node but also depends on the communication speed and is limited by the interconnect switch capability. There are many types of these switches including commonly used Gigabit Ethernet, Myrinet and Infiniband; infiniband being the fastest among them. Bare-metal clusters can take advantage of the fastest switches and so are very popular in places where complex problems need to be solved.

Cloud computing is a model for delivering Internet-based information and technology services in real time. It allows users to see the services while the infrastructure that delivers these services remains transparent (or in the "cloud"). More importantly, cloud computing can gather power of large number of computers. However, mainly based on the virtualization concepts a Cloud computing thus can at most take the speed of gigabit interconnect network.

In the present study we focus on the academic bare-metal cluster and cloud platform to study above mentioned features and differences.

Hypothesis for the study in question

A hypothesis for this study is that for larger and more complex problems where the performance of the computation on a distributed system relies on the communication will show stark differences in the results obtained from the above two platforms. The cloud platform will show
lower performance in this case since the infini0band interconnect in the bare-metal will be much faster in achieving better communication between compute nodes.

We will show these differences by implementing a parallel program on PageRank on both systems.

**Overview of PageRank algorithm:**

In a web2.0 era it is becoming increasingly important to search/find the most relevant data specific to query when we have data corresponding to billions of webpages on the internet. Moreover everyday thousands or more webpages get added to the pool of this data, so the filtering of this search criteria has to be updated constantly or at least periodically enough to get the data properly indexed. Thus there is always a need to sort/index the webpages with some scoring index. PageRank algorithm introduced by Google-search engine tries to address this need. The original algorithm gives the ranking to each webpage (not to the whole website) based on how many times it can be accessed (based on the links it has to other pages and links from other pages to this page).

**The algorithm**

![Figure1. Illustration of PageRank Calculation (taken from Prof Judy Qiu’s notes)](image-url)
The original PageRank algorithm was described by Lawrence Page and Sergey Brin in several publications [1]. It is given by

\[ PR(A) = (1-d) + d \left( \frac{PR(T1)}{L(T1)} + \ldots + \frac{PR(Tn)}{L(Tn)} \right) \]  

(1)

Where,

- \( PR(A) \) is the PageRank of page A,
- \( PR(Ti) \) is the PageRank of pages Ti which link to page A,
- \( L(Ti) \) is the number of outbound links on page Ti and
- \( d \) is a damping factor which can be set between 0 and 1.

Thus, PageRank algorithm does not rank web sites as a whole, but is determined for each page individually. Further, the PageRank of page A is recursively defined by the PageRanks of those pages which link to page A.

The PageRank of pages Ti which link to page A does not influence the PageRank of page A uniformly. Within the PageRank algorithm, the PageRank of a page T is always weighted by the number of outbound links L(T) on page T. This means that the more outbound links a page T has, the less will page A benefit from a link to it on page T. The weighted PageRank of pages Ti is then added up. The outcome of this is that an additional inbound link for page A will always increase page A's PageRank.

Finally, the sum of the weighted Page Ranks of all pages Ti is multiplied with a damping factor \( d \) which can be set between 0 and 1. Thereby, extension of PageRank benefit for a page by another page linking to it is reduced.

The larger the dataset of the webpages, the larger is the computation and therefore time increases if it is computed serially. Thus, with billions of webpages out on the internet, we need to have fast and parallel algorithm that will be run periodically but efficiently. This project aims to develop the PageRank algorithm using parallel computing interface (MPI). When optimized, PageRanks can be computed fast enough (within the time taken by server to reply to a query).

**Monitoring system:**

To ensure sustained productivity in any server based or distributed system, it is critical to make sure that there are downtimes of the system as few times as possible. This in turn requires constant monitoring and log-keeping systems. In this project we implement the monitoring system with pub/sub.

Real-time monitoring of the computing system can allow one to check:

- Disk Usage: How much space is used and what is the read/write rate.
- Memory usage: How much system memory is used by the program
- CPU usage: CPU usage per core or total usage for the running program
These monitoring enables whether a particular program is appropriate for the used system. For example, if the program takes 100% memory, then it needs memory upgrade on the system.

Monitoring also allows preventing faulty incidents. For example, constant log-keeping and monitoring can tell when the disk is about to be full and may warn the system-administrators, who in turn can request users to reduce their quotas, before the system becomes full. This prevents crashing of on-going programs in time.

There are many monitoring systems such as Ganglia [1], Inca [2] or Nagios [3]. In a good monitoring system a good frequency is chosen for variety of monitoring instances such as: CPU usage, Memory, Disk Usage, System logs, Specific processes (databases, webservers, and application servers), System Backup logs, Application logs, Network usage, and Network connections. For each instance one can define actions that can be taken when threshold of each instance reaches (for example, mail the users when the disk is 80% full).

Fig. 2 Pub/Sub based monitoring system

Thus, in the present study, we attempt to provide such monitoring system for our PageRank implementation.

**Experiments:**

In the following sections we describe our implementation of the parallel PageRank algorithm, the experimental set up and the details of the program as well as input datasets.

The parallel implementation of the PageRank algorithm follows following flowchart:
Fig. 3: Flowchart of the parallel PageRank algorithm

The Program/Work Flow

The parallel implementation as shown in above Fig. has following features:

- **init_rank_value_table()** – this function initializes the ranks in rank_values_table which contains the page ranks of individual pages to value: (1/No of URL’s)
- **join_rvt_am()** – This function actually calculates the page rank.
- **adjacencyMatrix** – This matrix is formed by reading the input file. Each row in this matrix represents the source url and list of destination url’s
- **rank_values_table**: this table actually holds the url’s and their corresponding rank values.
- **intermediate_rvt**
- **dangling_value**- if any node has no outbound links then we increase the dangling value by the page rank of that node.
• dangling_value_per_page- after calculating all the page ranks we add dangling value per page(dangling value/ no. of URLs ) to the all page ranks.

**Input Data:**
Input data files representing the link structures, are generated for various datasets containing 1000 urls to 50000 urls

**MPI implementation:**
The MPI implementation of this code is written in C.
There are two files:
1) mpi_main.c initiates and finalizes the MPI calls,
2) mpi_pagerank.c actually calculates the pagerank with loop iterations

Details of the code are as follows:
MPI_Main.c takes the arguments from command line, read local adjacency matrix from file , Initialize the MPI execution environment using MPI_Init , Computes initial rank values at root node. broadcast the initial rank values to all other compute nodes. Spawns the processes to do Page rank calculation. Calls MPI_PageRank function. Then Writes results to the file.
Root(rank 0) computes the initial rank value for each web page

MPI_PageRank.c Core computation of PageRank values. There is a local and Global copy of rank_values _table.This function stops further calculation of page rank when either the difference between current and old page rank values is less than the threshold value or the no of iterations exceeds the value entered by the user.

MPI_Comm_rank() - Determines the rank of the calling process in the communicator. Using. MPI_Bcast() Broadcast values to all other nodes.
MPI_AllReduce Combines the values from all processes and distributes the result back to all processes.

**Output:**
Top 10 ranked URL using MPI Algorithm using pagerank.input as input file were obtained as follows:

<table>
<thead>
<tr>
<th>URL</th>
<th>PageRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.401764</td>
</tr>
<tr>
<td>2</td>
<td>0.324932</td>
</tr>
<tr>
<td>4</td>
<td>0.081136</td>
</tr>
<tr>
<td>3</td>
<td>0.039172</td>
</tr>
<tr>
<td>5</td>
<td>0.039172</td>
</tr>
<tr>
<td>0</td>
<td>0.032919</td>
</tr>
<tr>
<td>6</td>
<td>0.016181</td>
</tr>
<tr>
<td>7</td>
<td>0.016181</td>
</tr>
<tr>
<td>8</td>
<td>0.016181</td>
</tr>
<tr>
<td>9</td>
<td>0.016181</td>
</tr>
</tbody>
</table>
Running MPI PageRank on a cluster and Eucalyptus Cloud infrastructure

Experimental set-up

FutureGrid, a national scale computing facility, provides both bare-metal as well as cloud platforms for researchers to solve their complex problems.

We ran this program first on bare-metal cluster and then on virtual machine platform (eucalyptus).

The bare metal cluster has infiniband interconnect while the Eucalyptus would only have Gigabit Ethernet as the fastest interconnect between two compute nodes. Our program showed distinct differences between the outputs as obtained from these two systems thereby yielding the performance benchmark for them.

Methods:

Two sets of run were performed: A) bare-metal platform – Here the MPI-PageRank was run for various combinations of number of CPUs and number of URLs on the bare-metal platform connected by Infini-Band network; and B) Eucalyptus – This is cloud platform that was used to test the PageRank program. This one only uses Gigabit Ethernet network connection.

The other details from the set up are as follows:

size of dataset = 1
number of URLs = 10000 to 50000
threshold = 0.000001
num_iterations = 10
number of worker nodes = 4 (4 VM instances)
class instance = 4 (with 1 head node)

Results and Discussion:

The results from the runs of MPI-PageRank program are summarized in Fig. 4 (A to D).

Figures A and B depict the results obtained from the runs on bare-metal platform. In figure A, we see the speedup versus number of URLs performance for the fixed number of processes (16). As can be seen from the figure, the performance is low for small dataset as well as for very large dataset. For very small dataset, the interconnect network becomes bottleneck as each processor waits for more time in data-communication than it can process. In the very large dataset regime, the interconnect communication may not be a problem; however each processor may be
overloaded to its full capacity of processing. Thus one has to find a good balance or trade-off for the size of the dataset for a given number of processors one can deploy for the problem. Figure B shows the performance in terms of speed up for a fixed size of the URLs but when the program is run using different number of processors. As can be expected as the number of processors go on increasing the performance gets better, utilizing the parallel efficiency of the algorithm.

Figure C and D show the results for the performance obtained from the Eucalyptus platform. In each case we used 4 VM instances. Figure C is similar to Figure A where the size of the dataset was varied for a fixed number of processors (6), and Figure D shows similar results as in Figure B where the performance in terms of speed-up was measured for a fixed data size but using varying number of processors. Interestingly, the performance for single processor is the best we achieve, and as the number of processors increase the performance decreases. In fact, this is expected as well, because on the cloud platform, because of VM instances, the interconnect becomes slow pertaining to only Gigabit efficiency instead of low-latency InfiniBand connection in bare-metal platform. This affects the performance seriously since; the data-communication takes over and becomes the bottleneck in the program runtime.

A)  No of processors =16

B)  PageRank URL size 30 000
A MVC based cluster monitoring system using pub/sub messaging middleware

In the java-based monitoring system implementation the following features were used:

Classes:

Constants.Java:

This class has static members which are referred by all other classes while initialization. These static members are TOPIC (which topic to publish/subscribe), BROKER_PORT and BROKER_HOST

Monitor.java:

This is the entry point class for running the monitor i.e. Subscriber. This class initializes the communication parameters and calls the StatClient class start method.

StatClient.java
This class creates the GenericGraph instance and calls the graphupdate method after every 5 seconds. Catches the exceptions and prints to the stack trace.

**GenericGraph.java**

This is the class which handles all the jfreechart display functionality. The updatechart method of this method is called after every 5 seconds to update the chart with new values.

All the graph properties are set in this class.

**MessageInfo.java**

This class is used to wrap message which contains NodeID, CPU Usage and Memory usage. This class has appropriate getters and setters.

Publisher wraps the message into the object of this class it is serialized and sent over the network, the Consumer receives them, deserializes them.

**MonitorDaemon.java**

This is the main class for Publisher. It creates the threads of deamonworker (PubSubBase) and runs them.

This class initializes the communication parameters and calls the StatClient class start method.

**PubSubBase.java**

This is the class which interacts with SIGAR gets the system parameters.

**Results of the monitoring system**

Successful implementation of the monitoring system yielded CPU and memory usage monitoring in real time. Some snapshots of the implementation shown below:
Fig 5. Snapshots of monitoring implementation
Conclusion:

The PageRank algorithm is based on a simple idea of giving votes to individual webpages rather than sites. The more votes any page gets, the more popular it is, hence gets higher rank. Going from the serial implementation of this algorithm to the parallel one, requires a careful treatment of how to divide the work between nodes. Moreover it is possible to further optimize this parallel implementation for memory and storage requirement (not performed in this project).

It seems that bare-metal platform is suitable when you already have data at disposal and you want to perform high-performance computing. The cloud platform (such as eucalyptus) takes long time in data-communication so it becomes useful in instances where this communication is required least (for example, net-based search such as bioinformatics-data where one does not need to communicate data at every time step/iteration).

Finally, a monitoring system was successfully implemented and studied. This implementation provided means of monitoring CPU and memory usage in real time.

Acknowledgement
We are grateful for FutureGrid administrators for providing FutureGrid access for our work and their help in running the program successfully. Special thanks are to Andrew Young who helped solving major issues whenever technical problems regarding FutureGrid arose. We are also thankful to all the questions-answers raised by the class-mates as the forums helped to solve problems while executing our tasks. Last but not the least, we thank Prof Qiu and the AIs for both guiding in each task as well as showing the distributed system approach of the overall project.

References
[13] Prof. Judy Qiu Lectures B534 course