Benchmarking a Grid Computing Environment Using Different Communication Bandwidth

Chao-Tung Yang[1] Chuan-Lin Lai[1] Po-Chi Shih[1] Kuan-Ching Li[2]

[1] *High-Performance Computing Laboratory*
Department of Computer Science and Information Engineering
Tunghai University
Taichung, 407 Taiwan, ROC
ctyang@mail.thu.edu.tw

[2] *Parallel and Distributed Processing Center*
Department of Computer Science and Information Management
Providence University
Shalu, Taichung, 433 Taiwan, ROC
kuancli@pu.edu.tw

Abstract- *Internet computing and grid technologies promise to change the way we tackle complex problems. They will enable large-scale aggregation and sharing of computational, data and other resources across institutional boundaries. And harnessing these new technologies effectively will transform scientific disciplines ranging from high-energy physics to the life sciences. In this paper, we construct two heterogeneous PC clusters to form a grid computing environment and setup the middleware - Globus Toolkit on the master node of each cluster. Then, CISCO 2511 router is used to connect these two PC clusters and control the communication bandwidths. Subsequently, we use two different communication styles to benchmark our grid computing system: one is tightly-coupled synchronization job and the other one is loosely-coupled communication job. The result show when tightly synchronization application is running with lower communication bandwidth will result in the overall performance down. But the lower communication bandwidth will not affect the performance of loosely-coupled application.*

Keywords: Benchmarking, Grid computing, Globus, SUN Grid Engine, Speedup, Cluster computing.

1. Introduction

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing [1, 14, 15].

The Infrastructure of grid is a form of networking. Unlike conventional networks that focus on communication among devices, grid computing harnesses unused processing cycles of all computers in a net-work for solving problems too intensive for any stand-alone machine. A well-known grid computing project is the SETI (Search for Extraterrestrial Intelligence) @Home project [30], in which PC users worldwide donate unused processor cycles to help the search for signs of extraterrestrial life by analyzing signals coming from outer space. The project relies on individual users to volunteer to allow the project to harness the unused processing power of the user's computer. This method saves the project both money and resources.

Another key technology in the development of grid networks is the set of middleware applications that allows resources to communicate across organizations using a wide variety of hardware and operating systems. The Globus Toolkit [2] is a set of tools useful for building a grid. Its strength is a good security model, with a provision for hierarchically collecting data about the grid, as well as the basic facilities for implementing a simple, yet world-spanning grid.

Globus will grow over time through the work of many organizations that are extending its capabilities. More information about Globus can be obtained at http://www.globus.org. The accepted standard in this application space is the Globus Toolkit. The Globus Toolkit is a middleware product designed to facilitate grid computing, and also like Linux is available under an "open source" licensing agreement for free use.

The promise of grid computing is to provide vast computing resources for computing problems like the SETI example that require supercomputer type re-sources in a more affordable way. Grid computing also offers interesting opportunities for firms to tackle tough computing tasks like financial modeling without incurring high costs for super computing resources. The developers of the Globus Toolkit envision that grid computing will become the pervasive paradigm for providing computing recourses to large collaborative projects and virtual organizations.

The organization of this paper is as follow. In section 2, we make a background review of Cluster Computing, MetaComputing and Grid Computing. In section 3, it is our hardware and software configuration. In section 4, grid computing environment is proposed and constructed on multiple Linux PC Clusters by using Globus Toolkit (GT) and SUN Grid Engine (SGE). The experimental results are also con-ducted by using these MPI programs: pi problem, prime problem, matrix multiplication and POVRAY to demonstrate the performance. The experimental results are presented and discussed. We conclude this study in section 5.

2. Background Review

2.1. Cluster Computing

The first cluster computing was a NASA effort called Beowulf. Beowulf was started in 1994 and the first effort consisted of a 16-node cluster made up of commodity off the shelf (COTS) systems interconnected with Ethernet. While this approach does not try to exploit the excess computing power in the network, the use of COTS computers and standard network architectures means that Beowulf class systems are inexpensive to build and operate and can offer supercomputer levels of processing power.

Scalable computing clusters, ranging from a cluster of (homogeneous or heterogeneous) PCs or workstations to SMP (Symmetric MultiProcessors), are rapidly becoming the standard platforms for high-performance and large-scale computing. A cluster is a group of independent computer systems and thus forms a loosely coupled multiprocessor system. A network is used to provide inter-processor communications. Applications that are distributed across the processors of the cluster use either message passing or network shared memory for communication. A cluster computing system is a compromise between a massively parallel processing system and a distributed system. An MPP (Massively Parallel Processors) system node typically cannot serve as a standalone computer; a cluster node usually contains its own disk and equipped with a complete operating systems, and therefore, it also can handle interactive jobs. In a distributed system, each node can function only as an individual resource while a cluster system presents itself as a single system to the user.

Since a Beowulf cluster is a parallel computer system, it suits applications that can be partitioned into tasks, which can then be executed concurrently by a number of processors. These applications range from high-end, floating-point intensive scientific and engineering problems to commercial data-intensive tasks. Uses of these applications include ocean and climate modeling for prediction of temperature and precipitation, seismic analysis for oil exploration, aerodynamic simulation for motor and aircraft design, and molecular modeling for biomedical research [10, 11, 21, 22]

The previous study [11] lists four benefits that can be achieved with clustering. These can also be thought of as objectives or design requirements:

- Absolute scalability: It is possible to create large clusters that far surpass the power of even the largest standalone machines. A cluster can have dozens of machines, each of which is a multiprocessor.
- Incremental Scalability: A cluster is configured in such a way that it is possible to add new systems to the cluster in small increments. Thus, a user can start out with a modest system and expand it as needs grow, without having to go through a major upgrade in which an existing small system is replaced with a larger system.
- High availability: Because each node in a cluster is a standalone computer, the failure of one node does not mean loss of service. In many products, fault tolerance is handled automatically in software.
- Superior price/performance: By using commodity building blocks, it is possible to put together a cluster with equal or greater computing power than a single large machine, at much lower cost.

2.2. Grid Computing

Grid computing (or the use of a computational grid) is applying the resources of many computers in a network to a single problem at the same time - usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. A well-known example of grid computing in the public domain is the ongoing SETI (Search for Extraterrestrial Intelligence) @Home project [30] in which thousands of people are sharing the unused processor cycles of their PCs in the vast search for signs of "rational" signals from outer space. According to John Patrick, IBM's vice-president for Internet strategies, "the next big thing will be grid computing."

Grid computing requires the use of software that can divide and farm out pieces of a program to as many as several thousand computers. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. It can be confined to the network of computer workstations within a corporation or it can be a public collaboration (in which case it is also sometimes known as a form of peer-to-peer computing).

A number of corporations, professional groups, university consortiums, and other groups have developed or are developing frameworks and software for managing grid computing projects. The European Community (EU) is sponsoring a project for a grid for high-energy physics, earth observation, and biology applications. In the United States, the National Technology Grid is prototyping a computational grid for infrastructure and an access grid for people.

Grid computing appears to be a promising trend for three reasons: (1) its ability to make more cost-effective use of a given amount of computer resources, (2) as a way to solve problems that can't be approached without an enormous amount of computing power, and (3) because it suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as a collaboration toward a common objective. In some grid computing systems, the computers may collaborate rather than being directed by one managing computer. One likely area for the use of grid computing will be pervasive computing applications - those in which computers pervade our environment without our necessary awareness.

The establishment, management, and exploitation of dynamic, cross-organizational sharing relationships require new technology. This technology is Grid architecture and supporting software protocols and middleware [1, 2, 7, 9, 13, 14, 15, 16, 17, 18, 20, 30]

2.2.1. Globus Toolkit

The Globus Project [2] provides software tools that make it easier to build computational grids and grid-based applications. These tools are collectively called The Globus Toolkit. The Globus Toolkit is used by many organizations to build computational grids that can support their applications.

The composition of the Globus Toolkit can be pictured as three pillars: Resource Management, Information Services, and Data Management. Each pillar represents a primary component of the Globus Toolkit and makes use of a common foundation of security. GRAM implements a resource management protocol, MDS implements an information services protocol, and GridFTP implements a data transfer protocol. They all use the GSI security protocol at the connection layer [2, 20].

GRAM, GRAM [1, 2, 26] is designed to provide a single common protocol and API for requesting and using remote system resources, by providing a uniform, flexible interface to, local job scheduling systems. The Grid Security Infrastructure (GSI) provides mutual authentication of both users and remote resources using GSI (Grid-wide) PKI-based identities. GRAM provides a simple authorization mechanism based on GSI identities and a mechanism to map GSI identities to local user accounts.

MDS, MDS [1, 2, 27, 28] is designed to provide a standard mechanism for publishing and discovering resource status and configuration information. It provides a uniform, flexible interface to data collected by lower-level information providers. It has a decentralized structure that allows it to scale, and it can handle static (e.g., OS, CPU types, system architectures) or dynamic data (e.g., disk availability, memory availability, and loading). A project can also restrict access to data by combining GSI (Grid Security Infrastructure) credentials and authorization features provided by MDS.

GridFTP [1, 2, 23, 24, 25] is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The GridFTP protocol is based on FTP, the highly-popular Internet file transfer protocol. GridFTP provides the following protocol features:

- GSI security on control and data channels.
- Multiple data channels for parallel transfers. Partial file transfers.
- Direct server-to-server transfers.
- Authenticated data channels.
- Reusable data channels.
- Command pipelining.

2.2.2. MPICH-G2

MPI is a message-passing library standard that was published in May 1994. The "standard" of MPI is based on the consensus of the participants in the MPI Forums [3], organized by over 40 organizations. Participants include vendors, researchers, academics, software library developers and users. MPI offers portability, standardization, performance and functionality [22].

The advantage for the user is that MPI is standardized on many levels. For example, since the syntax is standardized, you can rely on your MPI code to execute under any MPI implementation running on your architecture. Since the functional behavior of MPI calls is also standardized, your MPI calls should behave the same regardless of the implementation. This guarantees the portability of your parallel programs. Performance, however, may vary between different implementations.

MPICH-G2 [4, 5] is a grid-enabled implementation of the MPI v1.1 standard. That is, using services from the Globus Toolkit® (e.g., job startup, security), MPICH-G2 allows you to couple multiple machines, potentially of different architectures, to run MPI applications. MPICH-G2 automatically converts data in messages sent between machines of different architectures and supports multiprotocol communication by automatically selecting TCP for intermachine messaging and (where available) vendor-supplied MPI for intramachine messaging. Existing parallel programs written for MPI can be executed over the Globus infrastructure just after recompilation [19].

2.2.3. SUN Grid Engine

Sun Grid Engine is new generation distributed resource management software which dynamically matches users' hardware and software requirements to the available (heterogeneous) resources in the network, according to policies usually defined by management.

Sun Grid Engine acts as the central nervous system of a cluster of networked computers. Via so-called daemons, the Grid Engine Master supervises all resources in the network to allow full control and achieve optimum utilization of the resources available.

Sun Grid Engine aggregates the compute power available in dedicated compute farms, networked servers and desktop workstations, and presents a single access point to users needing compute cycles. This is accomplished by distributing computational workload to available systems, simultaneously increasing the productivity of machines and application licenses while maximizing the number of jobs that can be completed.

In addition, Sun Grid Engine software helps lower the costs of purchasing, installing, setting up and administering the computing environment because it allows: Maximized use of new/existing resources, Lower administration costs, Lower upgrade costs, More efficient reuse of existing legacy, Resources [6, 22].

3. Hardware and Software Configurations

We construct a grid computing testbed which includes four Linux PC clusters:

- Alpha site: 4 PC with dual Athlon MP 2000MHz processor, 512MB DDRAM and Intel PRO100 VE interface.
- Beta Site: 4 PC with single Celeron 1700 processor, 256MB DDRAM, and 3Com 3c9051 interface.
- Gamma site: 4 PC with Dual Pentium 3 866 MHz processors, 256MB SDRAM and 3Com 3c9051 interface.
- Sigma site: 2 PC with single Pentium 4 2.4GHz processor, 256MB DDRAM and Accton EN-1216interface.

Each master node of the cluster is running SGE QMaster daemon and SGE EXECUTER daemon to running, manage and monitor incoming job and Globus Toolkit v2.42 also installed. Each slave node is running SGE EXECUTER daemon to execute incoming job only.

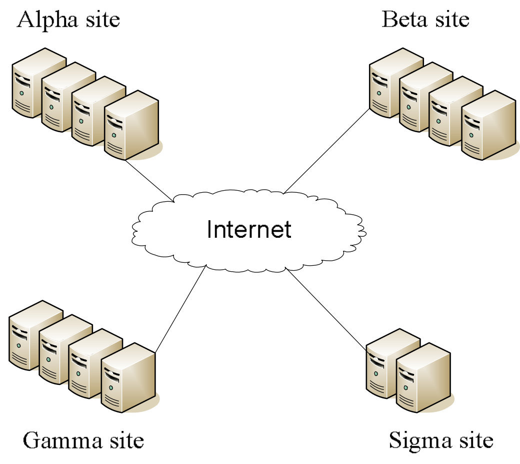
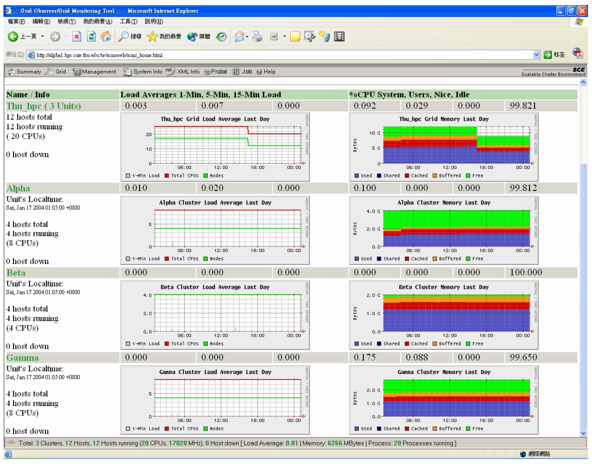


Figure 1: Our grid testbed

Sites 1 to 3, it locates at different department and lab in Tunghai University, Taiwan. Site4 locates at NCHC (National Center for High-Performance Computing), Tainan, Taiwan. We use general application to benchmark network traffic from Site 1 to Site4. Between Site (1, 2, 3) and Site (1, 2, 3), the average network latency is 3ms and the maximum transfer speed is 7600KBytes. Between Site (1, 2, 3) and Site4, the average network latency 5ms and the maximum transfer speed is 2000KBytes.


::: Grid Observer/Grid Monitoring Tool ::: - Microsoft Internet Explorer

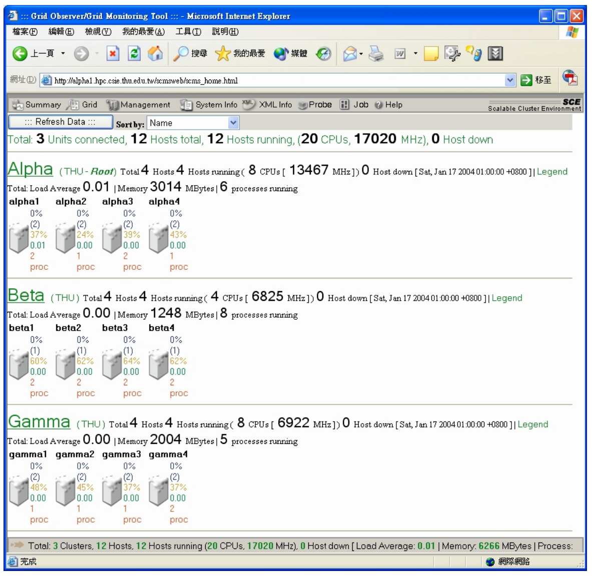
http://alpha1.hpc.csie.thu.edu.tw/scmsweb/scms_home.html

Name / Info	Load Averages 1-Min, 5-Min, 15-Min Load			%CPU System, Users, Nice, Idle			
Thu_hpc (3 Units)	0.003	0.007	0.000	0.092	0.029	0.000	99.821
12 hosts total 12 hosts running (20 CPUs) 0 host down							
Alpha	0.010	0.020	0.000	0.100	0.000	0.000	99.812
Unit's Localtime: Sat, Jan 17 2004 01:05:00 +0800 4 hosts total 4 hosts running (8 CPUs) 0 host down							
Beta	0.000	0.000	0.000	0.000	0.000	0.000	100.000
Unit's Localtime: Sat, Jan 17 2004 01:05:00 +0800 4 hosts total 4 hosts running (4 CPUs) 0 host down							
Gamma	0.000	0.000	0.000	0.175	0.088	0.000	99.650
Unit's Localtime: Sat, Jan 17 2004 01:05:00 +0800 4 hosts total 4 hosts running (8 CPUs) 0 host down							

Total: 3 Clusters, 12 Hosts, 12 Hosts running (20 CPUs, 17020 MHz), 0 Host down [Load Average: 0.01 | Memory: 6266 MBytes | Process: 20 Processes running]

Summary
Grid
Management
System Info
XML Info
Probe
Job
Help
SCE Scalable Cluster Environment
Thu_hpc Grid Load Average Last Day
Thu_hpc Grid Memory Last Day
Alpha Cluster Load Average Last Day
Alpha Cluster Memory Last Day
Beta Cluster Load Average Last Day
Beta Cluster Memory Last Day
Gamma Cluster Load Average Last Day
Gamma Cluster Memory Last Day
1-Min Load
Total CPUs
Nodes
Used
Shared
Cached
Buffered
Free
Bytes
06:00
12:00
18:00
00:00

Figure 2: Grid static summary monitoring


::: Grid Observer/Grid Monitoring Tool ::: - Microsoft Internet Explorer

http://alpha1.hpc.csie.thu.edu.tw/scmsweb/scms_home.html

Total: 3 Units connected, 12 Hosts total, 12 Hosts running, (20 CPUs, 17020 MHz), 0 Host down

Alpha (THU - Root) Total 4 Hosts 4 Hosts running (8 CPUs [13467 MHz]) 0 Host down [Sat, Jan 17 2004 01:00:00 +0800] | Legend
Total: Load Average 0.01 | Memory 3014 MBytes | 6 processes running

alpha1	alpha2	alpha3	alpha4
0%	0%	0%	0%
(2)	(2)	(2)	(2)
37%	24%	39%	43%
0.01	0.00	0.00	0.00
2 proc	1 proc	2 proc	1 proc

Beta (THU) Total 4 Hosts 4 Hosts running (4 CPUs [6825 MHz]) 0 Host down [Sat, Jan 17 2004 01:00:00 +0800] | Legend
Total: Load Average 0.00 | Memory 1248 MBytes | 8 processes running

beta1	beta2	beta3	beta4
0%	0%	0%	0%
(1)	(1)	(1)	(1)
60%	62%	64%	62%
0.00	0.00	0.00	0.00
2 proc	2 proc	2 proc	2 proc

Gamma (THU) Total 4 Hosts 4 Hosts running (8 CPUs [6922 MHz]) 0 Host down [Sat, Jan 17 2004 01:00:00 +0800] | Legend
Total: Load Average 0.00 | Memory 2004 MBytes | 5 processes running

gamma1	gamma2	gamma3	gamma4
0%	0%	0%	0%
(2)	(2)	(2)	(2)
48%	45%	37%	37%
0.00	0.00	0.00	0.00
1 proc	1 proc	1 proc	2 proc

Total: 3 Clusters, 12 Hosts, 12 Hosts running (20 CPUs, 17020 MHz), 0 Host down [Load Average: 0.01 | Memory: 6266 MBytes | Process:

Figure 3: Grid summary view

4. Experimented Results

4.1. Application Performance Evaluation

We use the MPI-based parallel applications for experimentation, the detail are listed as below:

Matrix multiplication: The matrix operation derives a resultant matrix by multiplying two input matrices, a and b, where matrix a is a matrix of N rows by P columns and matrix b is of P rows by M columns. The resultant matrix c is of N rows by M columns. The serial realization of this operation is quite straightforward as listed in the following:

```
for(k=0; k<M; k++)
        for(i=0; i<N; i++){
                c[i][k]=0.0;
                for(j=0; j<P; j++)
                        c[i][k]+=a[i][j]*b[j][k];
        }
```

Its algorithm requires n3 multiplications and n3 additions, leading to a sequential time complexity of O(n3). Let's consider what we need to change in order to use MPI. The first activity is to partition the problem so each slave node can perform on its own assignment in parallel. For matrix multiplication, the smallest sensible unit of work is the computation of one element in the result matrix. It is possible to divide the work into even smaller chunks, but any finer division would not be beneficial because of the number of processor is not enough to process, i.e., n2 processors are needed.

Prime Number: For prime problem, for example, if you want to find the prime numbers between 1 and 20,000,000 (20 million). It proceeds to write code that initially runs on a lead node and sends the task of testing 101-200 to node 1, and sends the task of testing 201-300 to node 2, and so on. Along with the testing task, there would also be an instruction to return whatever primes a slave node discovered to the lead node. When all nodes have completed their tasks, there will have a message to tell you how many prime be found and what the biggest prime number is.

PI Problem: It computes the value of by numerical integration. Since

$$\int_0^1 \frac{1}{1+x^2}dx = \tan^{-1}(1) = \frac{\pi}{4}$$

We can compute by integration the function from 0 to 1. We compute an approximation by dividing the interval [0, 1] into some number of subintervals and then computing the total area of these rectangles by having each process compute the areas of some subset.

CFD: A computational technology that enables you to study the dynamics of things that flow. Using CFD, you build a computational model that represents a system or device that you want to study. Then you apply the fluid flow physics to this virtual prototype, and the software outputs a prediction of the fluid dynamics. CFD is a sophisticated analysis technique. It not only predicts fluid flow behavior, but also the transfer of heat, mass (such as in perspiration or dissolution), phase change (such as in freezing or boiling), chemical reaction (such as combustion), mechanical movement (such as an impeller turning), and stress or deformation of related solid structures (such as a mast bending in the wind).

Jacobi: The Jacobi method is a method of solving a tridiagonal matrix equation with largest absolute values in each row and column dominated by the diagonal element. Each diagonal element is solved for, and an approximate value plugged in. The process is then iterated until it converges.

Merge sort: The merge sort splits the list to be sorted into two equal halves, and places them in separate arrays. Each array is recursively sorted, and then merged back together to form the final sorted list. Like most recursive sorts, the merge sort has an algorithmic complexity of $O(n \log n)$

Elementary implementations of the merge sort make use of three arrays - one for each half of the data set and one to store the sorted list in. The below algorithm merges the arrays in-place, so only two arrays are required.

Heap sort: The heap sort is the slowest of the $O(n \log n)$) sorting algorithms, but unlike the merge and quick sorts it doesn't require massive recursion or multiple arrays to work. This makes it the most attractive option for very large data sets of millions of items.

The heap sort works as it name suggests - it begins by building a heap out of the data set, and then removing the largest item and placing it at the end of the sorted array. After removing the largest item, it reconstructs the heap and removes the largest remaining item and places it in the next open position from the end of the sorted array. This is repeated until there are no items left in the heap and the sorted array is full. Elementary implementations require two arrays - one to hold the heap and the other to hold the sorted elements.

POVRAY: Given a number of computers and a demanding POVRay scene to render, there are a number of techniques to distribute the rendering among the available resources. If one is rendering an animation then obviously each computer can render a subset of the total number of frames. The frames can be sent to each computer in contiguous chunks or in an interleaved order, in either case a preview (every Nth frame) of the animation can generally be viewed as the frames are being computed.

MPIPOV has the ability to distribute a rendering across multiple heterogeneous systems. Parallel execution is only active if the user gives the "+N" option to POV. Otherwise, MPIPOV behaves the same as regular POV-Ray and runs a single task only on the local machine. Using the MPI code, there is one master and many slave tasks. The master has the responsibility of dividing the image up into small blocks, which are assigned to the slaves. When the slaves have finished rendering the blocks, they are sent back to the master, which combines them to form the final image. The master does not render anything by itself, although there is usually a slave running on the same machine as the master, since the master doesn't use very much CPU power.

If one or more slaves fail, it is usually possible for MPIPOV to complete the rendering. MPIPOV starts the slaves at a reduced priority by default, to avoid annoying the users on the other machines. The slave tasks will also automatically time out if the master fails, to avoid having lots of lingering slave tasks if you kill the master. MPIPOV can also work on a single machine, like the regular POV-Ray, if so desired. The code is designed to keep the available slaves busy, regardless of system loading and network bandwidth. We have run MPIPOV on our grid testbed for skyvase pov model.

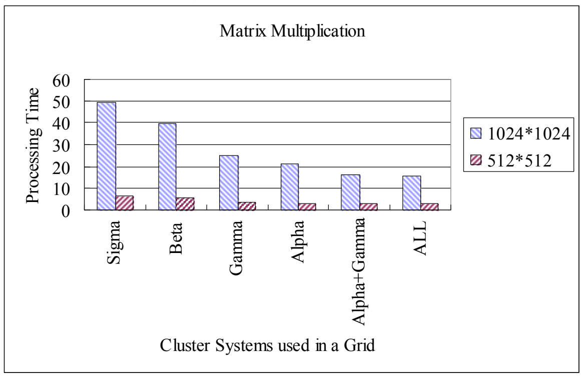


Figure 4: Matrix multiplication

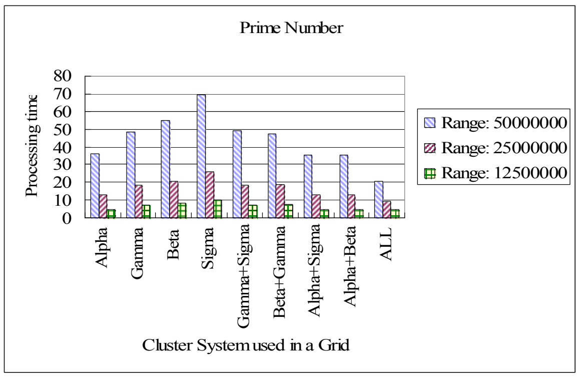


Figure 5: Prime Number

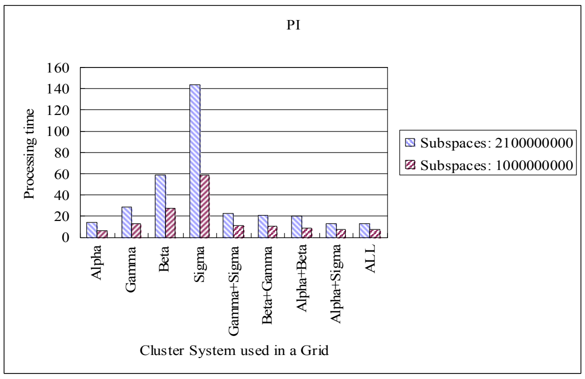


Figure 6: PI problem

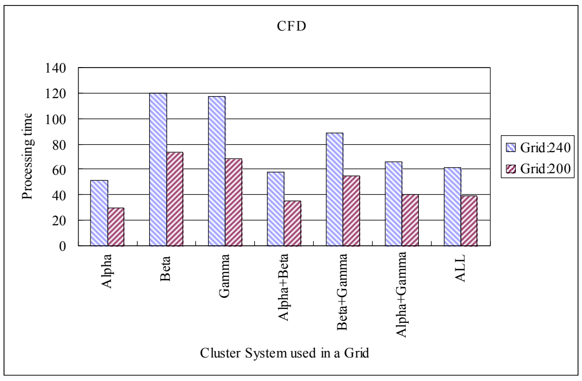


Figure 7: CFD

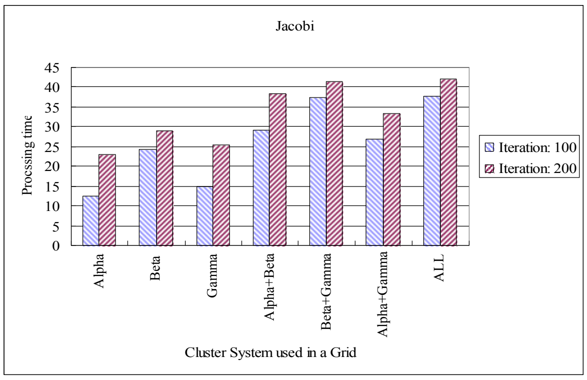


Figure 8: Jacobi iteration

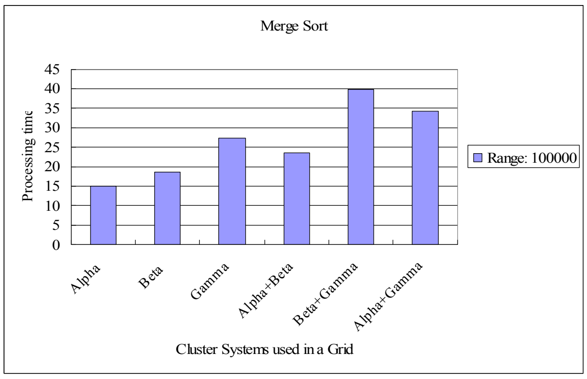


Figure 9: Merge sort

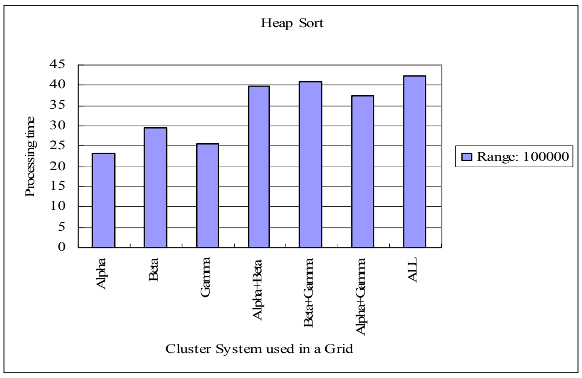


Figure 10: Heap sort

In those experiments, we can easily find that different type or style application running on different platform can get the different computing speed. In the past, when we want to solve a complex problem, we may submit the problem to the large computing center or self-made cluster. But, if we submit the job to the large computing center, ours job may get into the job queue to waiting for processing, it doesn't process instantly. If we submit the job to the self-made cluster, have a problem that the cluster computing power is not enough. It results in that we get the result for long time. Grid computing solves above problem. When we want to submit a job, first, we can search the resource list to find the best site suit for computing the job. Then, we submit the job to the best site. Multi-site computing is possible, but it not suit for every application. The experiment result you can find in Figures 4, 5, 6, and 7.

Multi-site computing, In Figures 8, 9, 10 when we want to solve a problem then we must think we need more computing resource to help us get the result in the shortest time. But, on the grid computing, it is not absolutely, grid computing is a heterogeneous and distributed computing environment, each resource is connected by the WAN connection between the node and node. Network bandwidth, latency and overhead can not to be compared with the system bus or LAN connection.

4.2. Performance Evaluation on Different Communication Bandwidth

In this experiment, we select alpha and gamma to perform the tests. We use this testbed with 2 CISCO 2511 router to control and analysis the grid system performance on different communication bandwidth.

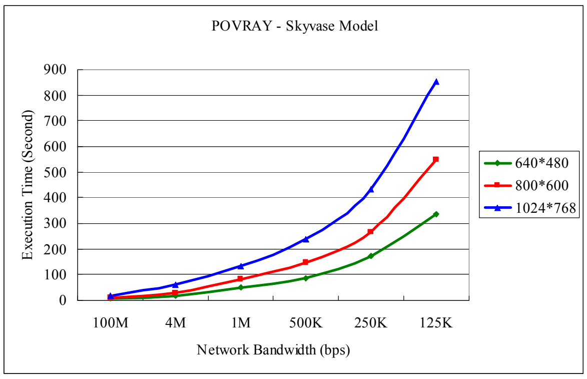


Figure 11: POVRAY – Skyvase

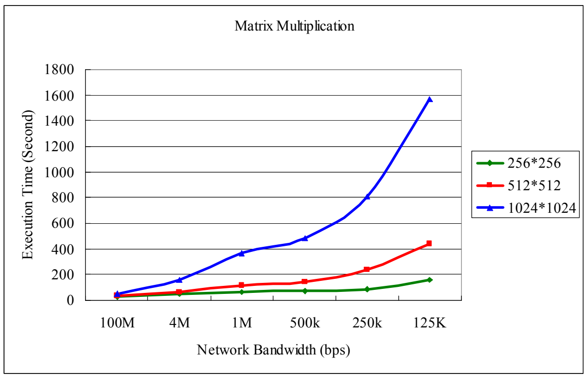


Figure 12: Matrix multiplication

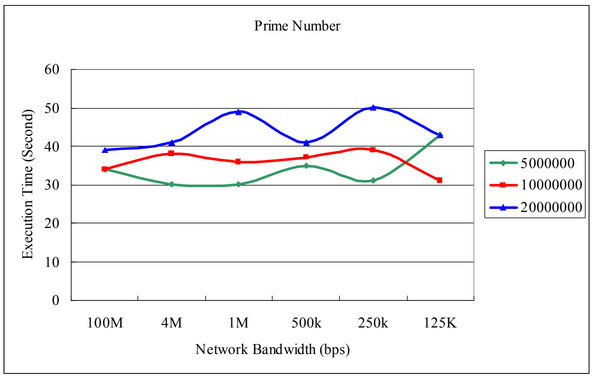


Figure 13: Prime Number

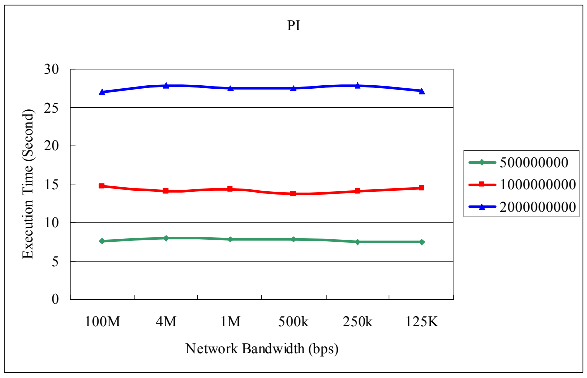


Figure 14: PI problem

From Figures 11 and 12, the result show when the application is running, the application may have significant computation, memory/data usage and large numbers of data communication. Lower communication bandwidth will result in the grid system overall system performance down. We can call this style of application "Tightly Synchronized".

From Figures 13 and 14, when application is running, it did not require large numbers of data communication. At the first, the program will divide the total job into the sub-job then dispatch the sub-job to each computing node. Then each node will focus on its computing job, it will not have any or little communication with other nodes before the computing task is completed. The result show different communication bandwidth will not affect the grid system overall performance. We can call this style of application "Loosely Coupled".

5. Conclusion and Future Work

In this paper, we construct two heterogeneous grid-connected PC Clusters. Then we use this testbed and 2 CISCO 2511 router to analysis the grid system performance on different communication bandwidth. The result show when tightly synchronization application is running lower communication bandwidth will result in the overall performance down. But the lower communication bandwidth will not affect the loosely coupled application's performance.

Towards actual use of this work, two further research projects will be carried out. First is a job scheduler between cluster should be developed a cooperative job scheduler. When the node of the cluster CPU load is busy or the communication bandwidth jam. The job will not send to that node, the cooperative scheduler will send the job to other node. Second, it is the network QoS. In the internet, packet loss or network latency will also affect the Grid system performance. To develop a QoS algorithm to redirect the routing path is also important.

References

[1] Global Grid Forum, http://www.ggf.org
[2] The Globus Project, http://www.globus.org/
[3] MPI Forum, http://www.mpi-forum.org/
[4] MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/
[5] MPICH-G2, http://www.hpclab.niu.edu/mpi/
[6] Sun ONE Grid Engine, http://wwws.sun.com/software/gridware/
[7] LHC - The Large Hadron Collider Home Page, http://lhc-new-homepage.web.cern.ch/
[8] TeraGrid, http://www.teragrid.org/
[9] KISTI Grid Testbed, http://gridtest.hpcnet.ne.kr/
[10] Chao-Tung Yang and Chi-Chu Hung, "*High-Performance Computing on Low-Cost PC-Based SMPs Clusters*," Proc. of the 2001 National Computer Symposium (NCS 2001), Taipei, Taiwan, pp 149-156, Dec. 2001
[11] E. Brewer, "*Clustering: Multiply and Conquer.*" Data Communications, July 1997.
[12] Catlett C., Smarr L., *Metacomputing*, Communications of the ACM, vol. 35(6), pages 44-52, 1992.
[13] Heath A. James, *Scheduling in Metacomputing Systems*, BSc (Ma&Comp Sc) (Hons)
[14] I. Foster, C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann; 1st edition (January 1999)
[15] I. Foster, *The Grid: A New Infrastructure for 21st Century Science.* Physics Today, 55(2):42-47, 2002.
[16] I. Foster, C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit.* Intl J. Supercomputer Applications, 11(2):115-128, 1997
[17] Mark A. Baker and Geoffery C. Fox.

Metacomputing: Harnessing Informal Supercomputers. High Performance Cluster Computing. Prentice-Hall, May 1999. ISBN 0-13-013784-7.

[18] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International J. Supercomputer Applications, 15(3), 2001..

[19] I. Foster, N. Karonis, *A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems*. Proc. 1998 SC Conference, November, 1998.

[20] *Introduction to Grid Computing with Globus*, ibm.com/redbooks, 2002.

[21] R. Buyya, *High Performance Cluster Computing:System and Architectures*, vol. 1, Prentice Hall PTR, NJ, 1999.

[22] Thomas Sterling, Gordon Bell, Janusz S. Kowalik, *Beowulf Cluster Computing with Linux*, MIT Press, Paperback, Published March 2002.

[23] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, S. Tuecke, *GridFTP Protocol Specification*. GGF GridFTP Working Group Document, September 2002.

[24] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke, *Data Management and Transfer in High Performance Computational Grid Environments*. Parallel Computing Journal, Vol. 28 (5), May 2002, pp. 749-771.

[25] B. Allcock, S. Tuecke, I. Foster, A. Chervenak, and C. Kesselman, *Protocols and Services for Distributed Data-Intensive Science*. ACAT2000 Proceedings, pp. 161-163, 2000.

[26] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[27] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, *Grid Information Services for Distributed Resource Sharing*. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[28] X. Zhang, J. Freschl, and J. Schopf, *A Performance Study of Monitoring and Information Services for Distributed Systems*. Proceedings of HPDC, August 2003.

[29] Sun ONE Grid Engine Administration and User's Guide, Sun Microsystem, Inc. (2002)

[30] Chuan-Lin Lai, Chao-Tung Yang, "*Construct a Grid Computing Environment on Multiple Linux PC Clusters*" International Conference on Open Source 2003.

[31] Chuan-Lin Lai, Chao-Tung Yang, "*Construct a Grid Computing Environment for Parallel Rendering*" 2003 Computer Graphics Workshop.