Int. Computer Symposium, Dec. 15-17, 2004, Taipei, Taiwan.

Grid Enabled MPI: PACX-MPI Optimization

MingShen Lin
Department of Electrical Engineering
National Tsing Hua University, Taiwan
Email: g913972@oz.nthu.edu.tw

Yarsun Hsu
Department of Electrical Engineering
National Tsing Hua University, Taiwan
Email: yshsu@ee.nthu.edu.tw

Abstract— **As the power of computer progresses, software gets increasingly more complex and intelligent. Meanwhile, network infrastructure has also improved at a high speed. Because of these advancements, there is a strong interest in sharing computing resources scattered over many different places. Therefore, the large-scale resources sharing and management become very important issues.**

At the beginning, the sharing of computational resources is the primary interest of resource sharing. A typical case is the so-called cluster of cluster (CoC), which consists of a group of clusters working together to solve a large computation-intensive application. To facilitate MPI programs to function properly across clusters, MPI extensions for parallel system, such as MPICH-Globus2 and MPICH-VMI have been introduced and are briefly discussed here. We also discuss how these MPI implementations work across clusters behind firewalls. In addition, an optimization of PACX-MPI for data transmission between two clusters has been implemented. Communications between two clusters are studied and handled with multithreading. System throughputs are measured by use of NAS Parallel Benchmark [16] and Persistence of Vision Raytracer [17].

I. Introduction

A. Grid

What is Grid? The word "Grid" is chosen by analogy with the electric power grid, which provides pervasive access to power [1]. In the article authored by Ian Foster and Steve Tuecke [2], it provides a concise statement of Grid problem.

> "controlled and coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations"

To achieve the goal, OGSA (Open Grid Service Architecture standard) and OGSI (Open Grid Service Infrastructure) are developed. OGSA defines an architecture for Grid and the architecture is based on Web Service [3]. Each resource is treated as a Web Service. Each Web Service can interoperate with each other. OGSI is developed, because of the lack of Web Service. In summary, Grid services are Web Services that conform to a set of conventions. OGSI is an enhancement of Web Services. OGSI defines mechanisms for creating, managing, and exchanging information among Grid services. OGSI defines several features listed below.

1) Factory and Instance: When a client accesses the service the first time, it will request the service factory for creating a new instance. Then the client is serviced by the instance.
2) Lifetime: Grid service instances are created with a specified lifetime. An instances will be destroyed automatically if time-out expires and the client does not extend the lifecycle of an instance.
3) State Management: OGSI specifies a framework for representing these states associated operations. There is a service data aggregator within a factory service for collecting and managing service data.
4) Notification: Notification enables Web Services to be event driven oriented.
5) GSH and GSR: Grid Service Handle is the identity of Grid service and is unique. It acts as Universal Resource Identity. GSR (Grid Service Reference) consists of information about how to communicate with Grid services.
6) Service Group: There is a Grid service instance that maintains information about the group relationship among the other services.

This work was supported by National Science Council, ROC under Grants NSC 93-2752-E-007-PAE and NSC 92-2213-E-007-052

B. Globus Toolkits

Globus Toolkits are developed by Globus Alliance, including academia and commercial corporations. The Globus Alliance's goal is to create fundamental technologies behind the "Grid". Globus toolkits provide APIs, protocols, and some simple services. Existing technologies, such as PKI (Public Key Infrastructure) or FTP (File Transfer Protocol) are combined in Globus toolkits. Globus toolkits include four primary components, GSI (Globus Security Infrastructure), GRAM (Globus Resource Allocation Manager), Data management, and Information management. Globus toolkits version 3 are the first full-scale implementation of the OGSI standard. Globus toolkits implement OGSI standard with Java language. GSI and GRAM can be combined for remote procedure call.

C. MPI

MPI [7] is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, developers, and users. In distributed systems, message passing is in common use for communication with each other. Another architecture of distributed systems is DSM (Distributed Share Memory) system that computers use load/store to communicate with each other. Because only some special network adaptors support load/store functions and DSM needs

OS support, thus distributed virtual share memory architecture has been developed. The DVSM (Distributed Virtual Share Memory) is a middleware to emulate DSM architecture. The performance of message passing is better than the performance of DVSM. Nowadays, large computational problems are solved by message passing programming. The most popular implementations are MPICH and LAM/MPI.

1) MPICH: MPICH [11] is a portable MPI implementation developed by Argonne National Laboratory (ANL). In order to be portable, its architecture is hierarchical. MPICH development team develops an ADI (Abstract Device Interface) and a channel interface for ADI [5]. If developing a different implementation is needed for special hardware or software algorithm, only writing a MPICH channel device is needed instead of rewriting all source codes. Therefore there are several channel device implementations of specific hardware. MPICH-Globus2 implements a Globus device in the channel interface level. MPICH-VMI uses VMI libraries to implement a device in the channel interface level. PACX-MPI implements a set of new functions based on MPICH point-to-point level.

D. Motivation

Now, a cluster is a good solution for high performance computing. For some applications, a cluster can not solve the problem alone. Therefore CoC(Cluster of Cluster) is developed. The most important issue of CoC is software. How do the existing applications support CoC? MPI standard does not deal with the issue. An MPI implementation which can work fine in CoC environment should be developed.

II. Grid-Enabled MPI

What does grid-enabled MPI mean? Grid-enabled MPI makes several clusters work together toward a final goal like one cluster. Every cluster in the environment is treated as an independent node. There are many researches and issues in the field of grid enabled MPI, for example, management, reliability, heterogeneity, availability, etc.

A. MPICH-VMI

Virtual Machine Interface [14](VMI) is a set of message layer libraries, and MPICH-VMI is a channel implementation of MPICH using VMI libraries. VMI is a middleware. The goal of VMI is to develop a library that is suitable for Grid environment. It focuses on availability, usability, and management [8]. The MPICH-VMI can support multiple network interfaces at the same time. That means if there are two clusters joining the mission, one cluster uses Myrinet, the other uses infiniband architecture. Communication between two clusters uses Ethernet network.

B. MPICH-Globus2

MPICH-Globus2 [15](MPICH-G2) is a channel implementation of MPICH using Globus libraries. MPICH-G2 uses Globus service for job startup and security. The startup method (mpirun) is redesigned for Globus toolkits. When a job is submitted by mpirun, the script transforms job request into Resource Specification Language [24] format and uses globusrun to submit the job. Globusrun is a script for submitting jobs in Globus toolkits version 2. The Globus channel device could automatically convert data into TCP/IP packets or vendor MPI packets.

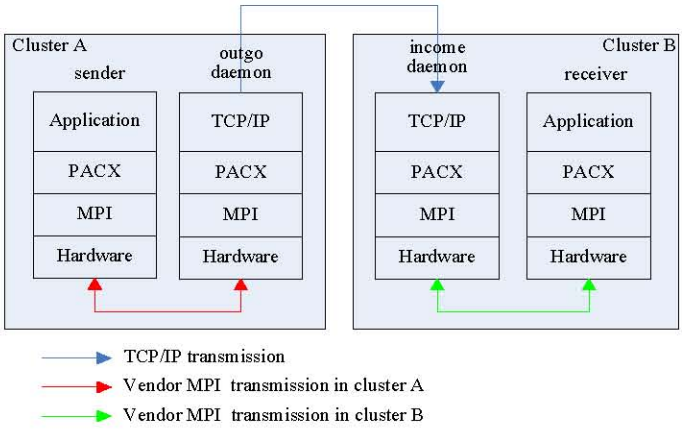

Fig. 1. PACX-MPI transmission

C. PACX-MPI

PACX-MPI [13] is based on vendor MPI and includes API library and compile tools. The compile tools consist of C compiler (pacxcc) and FORTRAN compiler (paxcfc). It doesn't provide a different mpirun script and thus the start-up method used before needs not be changed. When MPI program source codes are compiled, the pacxcc is used instead of the mpicc of vendor MPI. Although the pacxcc is called C compiler, it is not a real compiler. When compiling the source codes, pacxcc replaces the original MPI function (MPI_Send) with the PACX function (PACX_Send). Finally pacxcc links the object file to the necessary libraries.

As shown in figure 1, communications between two clusters rely on an outgo daemon and an income daemon. There are one outgo daemon and one income daemon in each cluster. Notice how a point-to-point communication functions. When a sender calls the function (MPI_Send), the function will determine where the receiver is. If the receiver is within the same cluster as the sender, the sender transmits data directly to the receiver via MPI_Send and MPI_Recv. If the receiver is not within the same cluster, the sender transmits data to the outgo daemon via MPI_Send and MPI_Recv. The detailed procedure is that the sender first generates a command packet that contains information about the sender, the receiver, data type, data length, command type, and two magic numbers to check integrity. After generating a command packet, the sender sends the command packet to the outgo daemon and then sends the data packet to the outgo daemon. In this way, a point-to-point communication is completed.

The PACX_MPI needs a file named ".hostfile" in the home directory or working directory. The file contains information about how many clusters are joining the CoC and how many nodes are in the cluster.

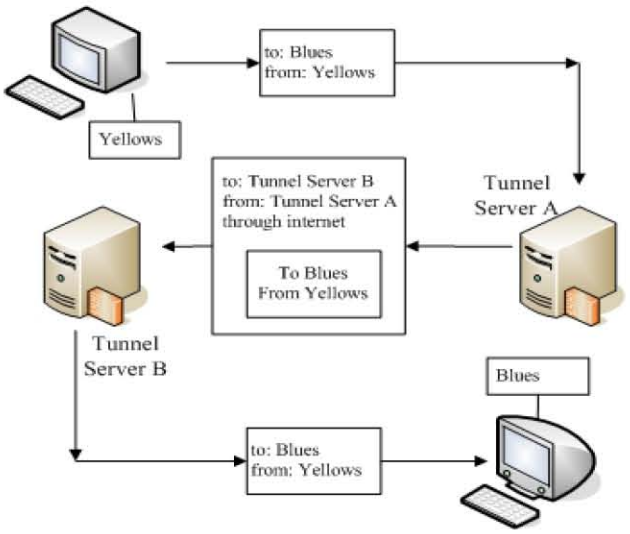

Fig. 2. IP tunneling

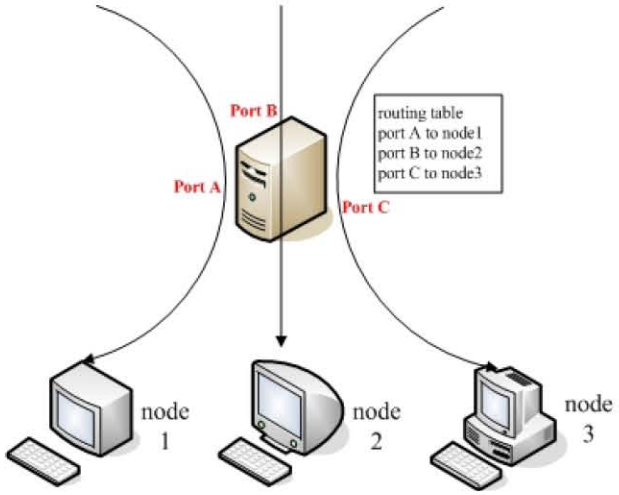

Fig. 3. Port forwarding

D. MPI Programs Across Clusters With Private IP Address

For security issue, a typical cluster architecture uses private IPs. How can a MPI program communicate between two nodes in separate clusters. There are two methods making existing MPI programs support private IP transparently instead of rewriting source code. One is IP tunneling and the other is port forwarding [9].

1) IP Tunneling: The concept of IP tunneling is illustrated in figure 2. Tunnel Server A and Tunnel Server B use public IPs and they are also NAT (network address translation) server. Both Yellows and Blues are two nodes within two different clusters. According to the routing table, when Yellows sends a packet to Blues, the server encapsulates the packet. Therefore Maximum Transfer Unit of IP tunneling device is 1480 bytes instead of 1500 bytes. When Yellows sends a packet to another node whose IP address is not specified in routing table, the server acts as a NAT server. When Tunnel Server B receives the packet from Tunnel Server A, it will forward the packet to Blues. There are two disadvantages. The first is that it is difficult to configure if there are many clusters joining the virtual organization. The second is that these nodes' private IP addresses can not be the same. There is a IP tunneling module in Linux kernel. Because it is a kernel module, it exhibits better performance.

2) Port Forwarding: Port forwarding is another method for solving private IP address issue. Figure 3 shows the concept of port forwarding. The NAT server transfers packets from a specific port to a specific node. The nodes outside the NAT server can send packets to the nodes inside NAT server through specific port. Its disadvantage is that the listening port of MPI receiving socket can not be predicted. There is no solution except writing a new channel device for MPICH that can specify the communication port. The article,Globus Toolkits Firewall Requirements [9], presents the requirements of Globus Toolkit behind a firewall. The method mentioned in the article does not solve the problem completely.

E. Summary

In the environment that each node owns a public IP, all MPI implementations mentioned above are qualified to work across clusters. In the environment that some nodes only have private IPs, PACX-MPI is currently the only solution.

III. Optimization for PACX-MPI

Packets in the outgo daemon are handled one by one. If a packet takes a long time, other non-related packets must also stall. There is a method letting the non-related packets be handled concurrently. We modify the code of outgo daemon to support this function.

A. Multithread

Thread [6] is called light weight process, because the newly spawned thread is in the same address space with the parent instead of a new process. Therefore threads belonging to the same process share the same resource. The procedure of spawning a new process needs to allocate a range of memory. After allocating the memory, OS copies the memory in the parent's memory space to the allocated memory. The procedure may have a little difference in different OS. Therefore the overhead of creating a thread is less than that of spawning a process. There is another advantage that threads belonging to the same parent share the resource in text region and data region. That means communications among threads does not need Inter-Process Communication. On the other hand, that may be a nightmare due to race condition.

In Linux, the pthread library is used widely. Pthread stands POSIX (Portable Operating System Interface for Computer) thread. For UNIX systems, this library has been specified by the IEEE POSIX 1003.1c standard (1995). Linux kernel doesn't support a real thread environment yet. The minimum scheduler unit is a process instead of a thread.

B. Pipeline Optimization

The original schedule policy of the daemon is serial, the next request begins once the previous request finishes. If two requests occur at the same time, one request must stall until the other request finishes. The concept of multithread and pipeline is needed.

As shown in figure 4(a), packet transmission from a sender in one cluster to a receiver in the other cluster follows the following procedures. First of all, MPI_Recv Command is sent from sender to the outgo daemon to specify the type of packet to be transmitted. This is followed by MPI_Recv Data, also from the sender to the outgo daemon to actually send the data. A thread is then spawned by the outgo daemon to handle TCP/IP connection with the income daemon in the other cluster so that data can actually be transmitted across clusters.

Due to MPI limitation, all MPI_Recv parts can not be handled concurrently in all cases. The TCP/IP parts can not be handled concurrently either if the destinations of packets are the same cluster. As shown in figure 4(b), only MPI_Recv and TCP/IP parts can overlap and separate TCP/IP parts can not overlap. However, if the destinations of packets are not the same cluster, as in figure 4(c), TCP/IP parts can be handled concurrently with separate threads. Pseudo codes of our optimization are listed below. The pthread_join and pthread_detach functions enable MPI_Recv parts to be received in turn. The pthread_mutex_lock and pthread_mutex_unlock enable TCP/IP parts to be handled in turn if the destinations of packets are the same.

```
pthread_join(wait unless thread_detach)
MPI_Recv Command
pthread_create
MPI_Recv Data
pthread_mutex_lock, pthread_detach
TCP/IP transmission
pthread_mutex_unlock
thread_exit
```

IV. Measurement Result

MPICH-P4, MPICH-GM, PACX-MPI are used in the measurement. MPICH-P4(p4) is MPICH over TCP/IP. MPICH-GM(gm) is MPICH over Myrinet. The item, pacx-m, means PACX-MPI with our optimization.

A. Testbed Description

As shown in figure 5, there are one front-end node and fifteen computing nodes. The font-end node is an SMP and joins others for computing.

1) CPU: AMD Athlon(tm) MP 2400+.
2) Memory: 512 MB with ECC function.
3) Linux kernel version: 2.4.20-19.7smp built by RedHat.
4) GCC version: 2.96 built by RedHat.
5) Network: Myrinet: up to ~490MB/s user-level bidirectional data rate. Ethernet: 100Mbps kernel-level data rate.

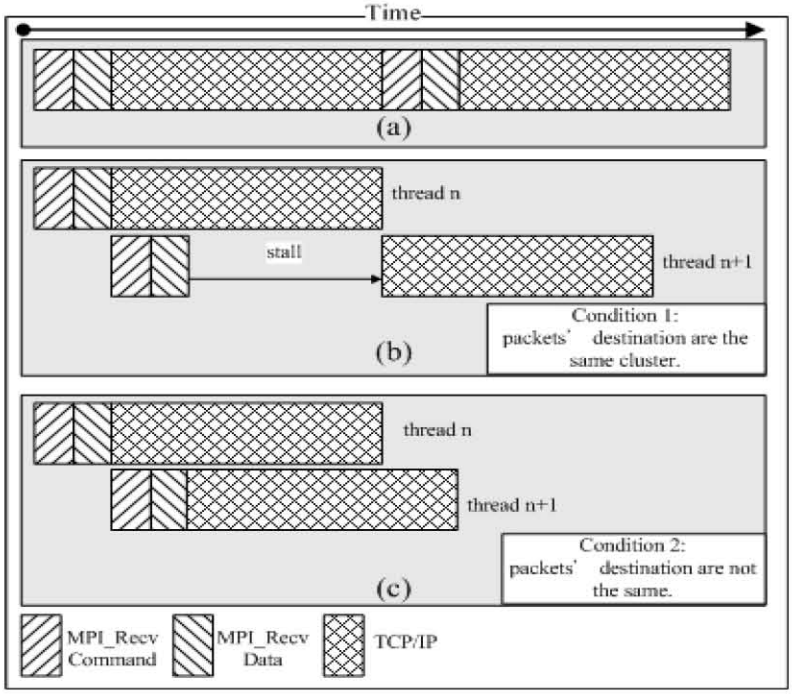

Fig. 4. Comparison

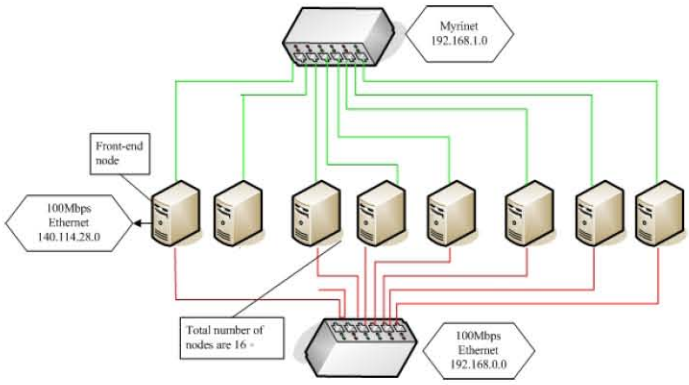

Fig. 5. Testbed description

B. NPB

NPB (NAS Parallel Benchmark) [16] is a set of eight programs developed by NASA Advanced Supercomputing (NAS). BT, FT, and MG can not pass the compiling process. In the statistical charts of IS and SP, the data of MPICH-GM is neglected because the value of the data is too large. EP is a communication insensitive program [21]. As shown in figure 7, all MPI implementations including MPICH-GM show the same performance. LU is sensitive to the small message communication performance of an MPI implementation [20]. As shown in figure 9, MPICH-GM shows the best performance and the others show the same performance. CG, IS and SP are communication sensitive programs [20][21]. As shown in figure 6,8,10, PACX-M (PACX with multithreading) shows the best performance in IS, PACX shows the best performance in

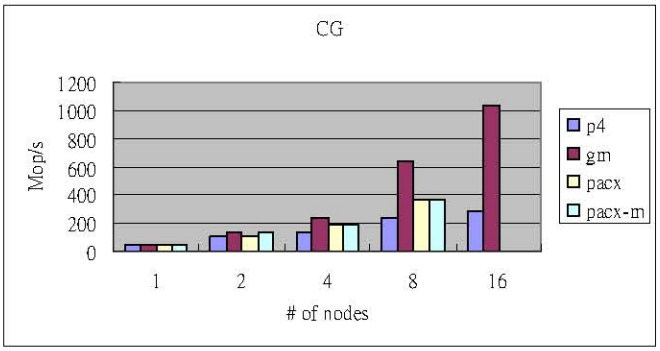

Fig. 6. NPB CG p4/gm/pacx/pacx-m

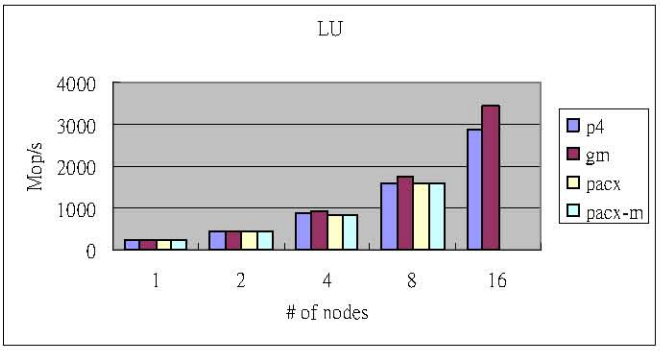

Fig. 9. NPB LU p4/gm/pacx/pacx-m

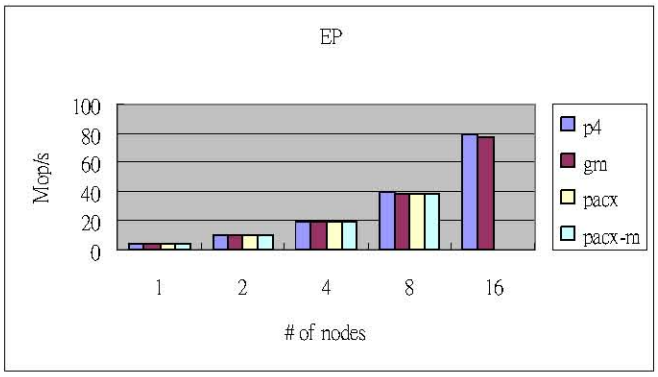

Fig. 7. NPB EP p4/gm/pacx/pacx-m

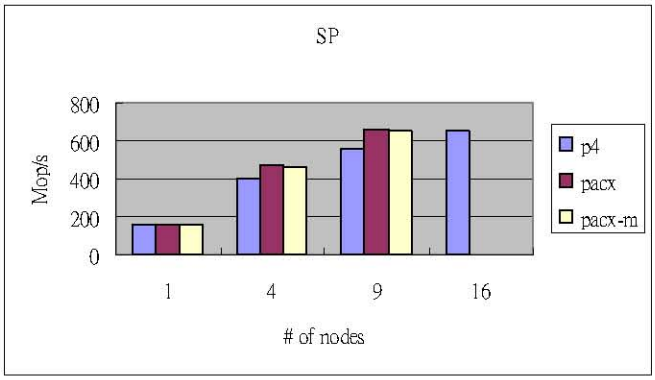

Fig. 10. NPB SP p4/pacx/pacx-gm

SP, and MPICH-P4 shows the worst performance.

Comparing with MPICH-P4, PACX-MPI shows better performance in all benchmark. The reason is that communications of PACX-MPI within a cluster are through vendor MPI. Only in IS and SP benchmark, PACX-MPI with multithreading gets better performance. Because IS benchmark is a communication sensitive program, the bottleneck of the communications between two clusters occurs in the daemon.

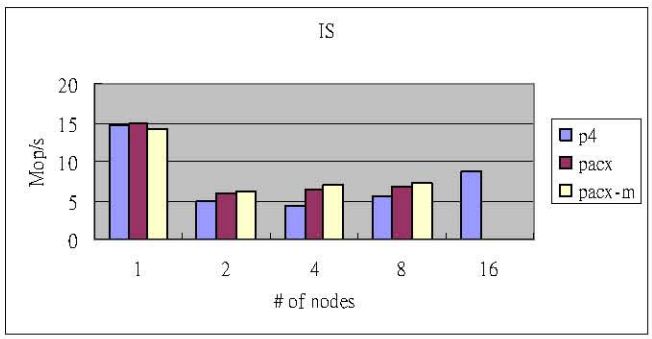

Fig. 8. NPB IS p4/pacx/pacx-m

C. POVRAY

POVRAY [17] (Persistence of Vision Raytracer) with MPI patch [18] is a communication insensitive program. If there are n nodes joining a job, one node becomes a server node and the others become client nodes. The server node partitions data and sends pieces of data to client nodes. When a client node finishes the processing of partitioned data, it sends result back and requests for new partitioned data. Generally, the transmission time of partitioned data is much shorter than the transmission time of result. There is no transmission except transmissions of partitioned data and result. Assume that the latency of network between two clusters is x, the process time for each node is y and the transmission time of partitioned data is ignored. When $x * (n-2) = y$, the system gets the best throughput. In actual condition, the network latency and number of computing nodes are determined. Therefore the performance is affected by the size of the partitioned data.

From figure 11, each MPI implementation shows almost the same performance in POVRAY benchmark. Comparing with the latency of data process, the latency of data transmission is extremely small within a cluster. Therefore MPICH-P4 and MPICH-GM show the same performance. The latency of data transmission can not be ignored between two clusters in separate organizations. The size of partitioned data is tuned for suiting the latency between two clusters in separate

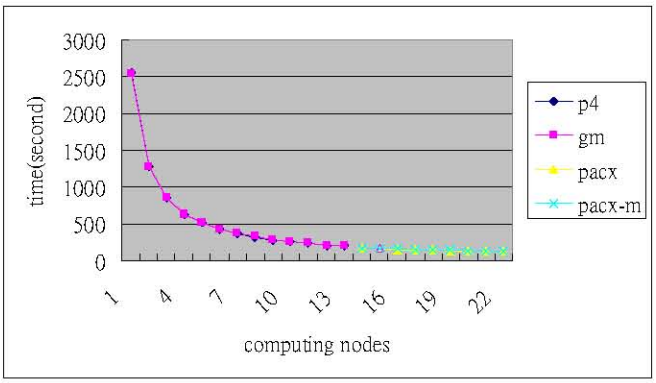

Fig. 11. POVRAY p4/gm/pacx/pacx-m

organizations. In our measurement, the size of partitioned data used by PACX-MPI is 16 times the size of partitioned data used by MPICH-P4.

V. Conclusion and Future Work

The results show that both MPICH-P4 and PACX-MPI show worse performance for the communication sensitive applications, but PACX-MPI may exhibit better performance. All the MPI implementations show almost the same performance for the communication insensitive applications. Only PACX-MPI can support private IP address. That means the PACX-MPI can extend easily its size, because existing clusters often use private IP address. In Grid, PACX-MPI can provide an on-demand resource for communication insensitive applications, such as 3D rendering.

Although the communication sensitive applications will show bad performance in Grid environment, there may be performance gains by some optimizations. According to the features of MPICH-G2 and MPICH-VMI, there are some potential optimizations for communications between two clusters. These optimizations are listed below.

A. Collective Operation With Topology-aware Mechanism

For broadcasting a message, the sender sends the same message to all receivers within the same cluster directly. Then the sender sends that message to the outgo daemon which will then forward it to the income daemons of the other clusters. The income daemons will then broadcast messages to all the receivers within those clusters.

B. Parallel TCP Streams

When sending large messages, there will be significant performance gains by partitioning a large message into several smaller messages. The detailed procedure is opening multiple sockets, partitioning the large message into packets, sending those packets in parallel, and re-assembling the large message. MPICH-G2 uses Globus Toolkits API to do that. There is a measurement of large data transfer in [19] which shows much improvement can be made with this method.

C. Real Time Compression

PACX-MPI has the ability to compress messages real time, but it is not bug-free. Both PACX-MPI and MPICH-VMI use Lempel-Ziv-Oberhumer [23] compression library to implement the compression function. LZO is a lossless real time compression library. As shown in the thesis [10], the compression makes much improvement.

References

[1] Ian Foster and Carl Kesselman, The Grid: blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc. San Francisco, California, 1999.
[2] Ian Foster, Carl Kesselman, Steven Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International J. Supercomputer Applications, 15(3), 2001.
[3] Ian Foster, C. Kesselman, J. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
[4] Ian Foster, Nicholas T. Karonis, A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, Proc. 1998 SC Conference, November, 1998.
[5] Gropp and Ewing Lusk, An abstract device definition to support the implementation of a high-level point-to-point message-passing interface, Preprint MCS-P342-1193, Argonne National Laboratory, 1994.
[6] Steve Kleiman, Devang Shah, and Bart Smaalders, Programming with Threads, Prentice Hall.
[7] Barry Wilkinson and Michael Allen, Parallel Programming, Techniques and Application Using Networked Workstations and Parallel Computers, Prentice Hall.
[8] Scott Pakin and Avneesh Pant, VMI 2.0: A Dynamically Reconfigurable Messaging Layer for Availability, Usability, and Management.
[9] Von Welch, Software Architect, Globus Project, Globus Toolkits Firewall Requirements, http://www.globus.org/security/firewalls/Globus%20Firewall%20Requirements-5.pdf, 22 July,2003
[10] Pradeep Kumar Panjwani, Monitoring And Compression Framework In Virtual Machine Interface 2.0, B.E., University of Bombay, 2000, Thesis.
[11] MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/.
[12] MiMPI, http://www.arcos.inf.uc3m.es/~mimpi/
[13] PACX-MPI, Rainer Keller, Matthias Muller, http://www.hlrs.de/organization/pds/projects/pacx-mpi/.
[14] MPICH-Virtual Machine Interface (VMI),http://vmi.ncsa.uiuc.edu/
[15] MPICH-Globus2, http://www3.niu.edu/mpi/.
[16] NASA Perfermance Benchmark (NPB), http://www.nas.nasa.gov/Software/NPB/.
[17] POVRAY, http://www.povray.org.
[18] POVRAY with MPI patch, Leon Verrall, http://www.verrall.demon.co.uk/mpipov/
[19] Nicholas T. Karonis, Michael E. Papka, Justin Binns, John Bresnahan, Joseph A. Insley, David Jones, Joseph M. Link, High-Resolution Remote Rendering of Large Datasets in a Collaborative Environment.
[20] The NAS Parallel Benchmarks 2.0, David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, Maurice Yarrow, Report NAS-95-020, December 1995, http://www.nas.nasa.gov/Research/Reports/Techreports/1995/PDF/nas-95-020.pdf
[21] New Implementations and Results for the NAS Parallel Benchmarks 2, William Saphir, Rob Van der Wijngaart, Alex Woo, Maurice Yarrow, http://www.nas.nasa.gov/Software/NPB/Specs/npb2.2_new_implementations.ps
[22] NAS Parallel Benchmarks Version 2.4, Rob F. Van der Wijngaart, NAS Technical Report NAS-02-007, Oct 2002, http://www.nas.nasa.gov/Research/Reports/Techreports/2002/PDF/nas-02-007.pdf.
[23] Lempel-Ziv-Oberhumer compression library, Markus F.X.J. Oberhumer, http://www.oberhumer.com/opensource/lzo/.
[24] Resource Specification Language, Globus project, http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/developer/mjs_rsl_schema.html