Performance Evaluation of a Firewall-compliant Globus-based Wide-area Cluster System

Yoshio Tanaka*, Mitsuhisa Sato
Real World Computing Partnership
Mitsui bldg. 14F, 1-6-1 Takezono Tsukuba
Ibaraki 305-0032, Japan
{yoshio,msato}@trc.rwcp.or.jp

Motonori Hirano
Software Research Associates, Inc.
1-1-1 Hirakawachou Chiyoda
Tokyo 102-8605, Japan
m-hirano@sra.co.jp

Hidemoto Nakada, Satoshi Sekiguchi
Electrotechnical Laboratory
1-1-4 Umezono Tsukuba
Ibaraki 305-8568, Japan
{nakada,sekiguchi}@etl.go.jp

Abstract

In this paper, we present a performance evaluation of a wide-area cluster system based on a firewall-enabled Globus metacomputing toolkit. In order to establish communication links beyond the firewall, we have designed and implemented a resource manager called RMF (Resource Manager beyond the Firewall) and the Nexus Proxy, which relays TCP communication links beyond the firewall. In order to extend the Globus Metacomputing Toolkit to the firewall-enabled toolkit, we have built the Nexus Proxy into the Globus toolkit. We have built a firewall-enabled Globus-based wide-area cluster system in Japan and run some benchmarks on it. In this paper, we report various performance results such as the communication bandwidth and latencies obtained as well as application performance involving a tree search problem. In a wide-area environment, the communication latency through the Nexus Proxy is approximately six times larger when compared to that of direct communications. As message size increases however, the communication overhead caused by the Nexus Proxy can be negligible. We have developed a tree search problem using MPICH-G. We used a self-scheduling algorithm, which is considered to be suitable for a distributed heterogeneous metacomputing environment since it performs dynamic load balancing with low overhead. The performance results indicate that the communication overhead caused by the Nexus Proxy is not a severe problem in metacomputing environments.

1 Introduction

As cluster systems become more widely available, it becomes feasible to run parallel applications on multiple clusters at different geographic locations (**wide-area cluster systems**) (See Figure 1). The Globus Metacomputing Toolkit[3] can be one of the best tools to build such metacomputing environments. The Globus system provides basic mechanisms such as communication, authentication, network information, and data access processes for building software infrastructure for a global computing environment. These mechanisms are used to construct various higher-level metacomputing services, such as parallel programming tools and schedulers.

Upon utilization of the Globus system at our site however, we were confronted with a firewall problem. A firewall is one of the most common security systems. A gateway machine which distinguishes the local site (inside) and remote sites (outside, the Internet) is called a firewall. The firewall watches communication packets between the inside and the outside and it can reject specific communication packets according to the configuration of the firewall, which is defined for a site-specific security policy. The two following configurations are possible:

- **An allow based configuration**

*He is working at the Electrotechnical Laboratory since Apr. 2000.

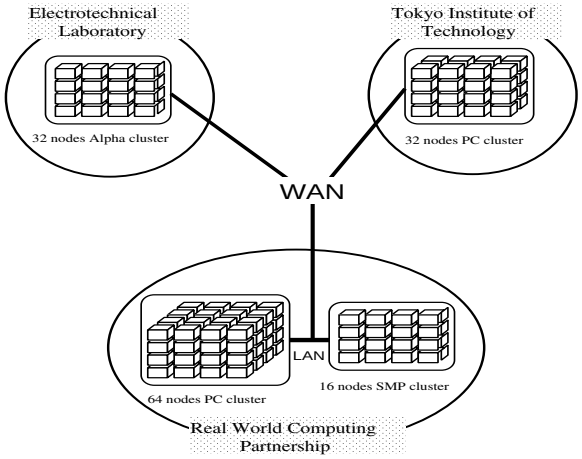


Figure 1. Wide-area cluster system.

All communication ports are basically open and all communication packets are allowed to pass through the firewall. In order to intensify the security, several ports should be configured as closed and communication packets on those ports should be denied.

- **A deny based configuration**
  All communication ports are basically closed and all communication packets are denied permission to pass through the firewall. In order to make specific ports open, they must be configured as open and communication packets on those ports must be allowed.

Although there are various configurations of firewall, we can identify the most typical as follows:

A deny based configuration is used for **incoming** packets.

An allow based configuration is used for **outgoing** packets.

In this paper, we assume that this configuration is used for the firewall.

Communication services within the Globus tookit are provided by the Nexus communication library[5]. Since the Nexus library in Globus ver. 1.0 allocates TCP communication ports dynamically and there is no mechanism in Nexus for specifying the port that the TCP protocol module listens on, a communication link cannot be established beyond the deny based firewall. Thus, resources inside the firewall cannot contribute to wide-area cluster systems. In Globus ver. 1.1, the Nexus library provides a mechanism to limit the port range that the TCP protocol module listens on by specifying the environment variables TCP_MAX_PORT and TCP_MIN_PORT. So, even in the deny based firewall, communication links can be established beyond the firewall by configuring the firewall to allow communications on that port range. However, this configuration is basically the same as the allow based firewall and loses the advantages of a deny based firewall.

In order to spread the global computing environment over various sites (including universities, laboratories, and companies), global computing systems should be firewall-enabled.

In order to make the Globus tookit firewall-enabled, we have designed and implemented a resource management system called RMF (Resource Manager beyond the Firewall)[9], which can be used as a Globus Resource Allocation Manager (GRAM). RMF manages multiple computing resources such as cluster systems and supercomputers. When the RMF type GRAM is used, computing resources inside the firewall can be utilized via a Globus gatekeeper which is running outside the firewall. In order to establish communication links between job processes beyond the firewall, we also designed and implemented the Nexus Proxy which relays TCP communications to provide a communication mechanism beyond the firewall. Compared with an approach of the Globus v1.1 to the firewall problem, binding the Nexus Proxy to a privileged port strengthens the security since binding to privileged ports needs *root* authority.

There are several reports concerning the performance evaluations of metacomputing environments. The Globus team has reported its experience constructing global computing environments using some testbeds including I-WAY[2] and GUSTO[3]. G. Mahinthakumar et.al.'s research[6] is very close to our research. They reported on the multivariate geographic clustering in a metacomputing environment using Globus. However, all testbeds described here have no firewall. In order to aggregate sites which are protected by a firewall into metacomputing environments, it is necessary to provide a mechanism for communications beyond the firewall and performance analysis for that mechanism.

Using RMF and the Nexus Proxy, we have built a firewall-enabled Globus-based wide-area cluster system. In this paper, we present a performance evaluation of that cluster system. In the next section, we briefly describe the RMF. The design and implementation of the Nexus Proxy is described in section 3. We measured the performance using a tree search problem as an example. We take the 0-1 knapsack problem using the branch-and-bound algorithm as our workload.

We implemented the knapsack problem using MPICH-G[4]. The experimental results and a discussion are presented in section 4 and finally, we summarize our work in section 5.

2 RMF: Resource Manager beyond the Firewall

The basic mechanism of RMF is a job queuing system and its behavior is similar to LSF, however RMF can utilize computing resources beyond the firewall. The most important design issue of RMF is how to "provide computing resources such as cluster systems and supercomputers inside the firewall to global computing environments". RMF consists of two basic modules, a **Job Queuing System (Q system)** and a **Resource Allocator**. The Q system is based on the client-server model. It provides a remote job execution mechanism using job queues. A server of the Q system (Q server) runs on every computing resource inside the firewall. A client of the Q system (Q client) is invoked by a job manager running outside the firewall. A resource allocator manages computing resources and runs as a daemon process inside the the firewall. Figure 2 illustrates the architecture of RMF and the relationships between the modules.

The following steps show the outline of the execution flow of a submitted job request.

0. Run a Globus gatekeeper for an RMF type GRAM outside the firewall. Run a resource allocator inside the firewall. Run a Q server on every computing resource.
1. A job request is submitted to the RMF gatekeeper.
2. The job manager invoked by the gatekeeper creates a Q client process.
3. The Q client inquires of a resource allocator which resources are best to execute a job.
4. A resource allocator selects resources and reports their names to the Q client.
5. The Q client submits a job request to the Q server running on the resources reported by the resource allocator.
6. The Q server receives the job request from the Q client and creates job processes according to the job type.

The firewall must be configured to allow communications between the Q client the resource allocator, and the Q client and the Q server. The Q system provides a mechanism to create job processes on different resources from the one on which the gatekeeper and the job manager are running. Since the Globus GASS facility uses files for input/output, the Q system also transfers the files to remote resources.

3 The Nexus Proxy

The Nexus Proxy relays TCP communications to provide a communication mechanism beyond the firewall. The Nexus Proxy outer server runs outside the firewall and the Nexus Proxy inner server runs inside the firewall as daemon processes. When a client process running inside the firewall tries to connect to remote hosts or bind a socket to listen on, the client process sends a relay request (either a connect request or a bind request) to the Nexus Proxy outer server instead of directly connecting/binding a port by calling *connect*() and *bind*() functions. Some library functions shown in Table 1 are provided to utilize the Nexus Proxy.

Figures 3 and 4 illustrate the communication mechanism via the Nexus Proxy. In both figures,

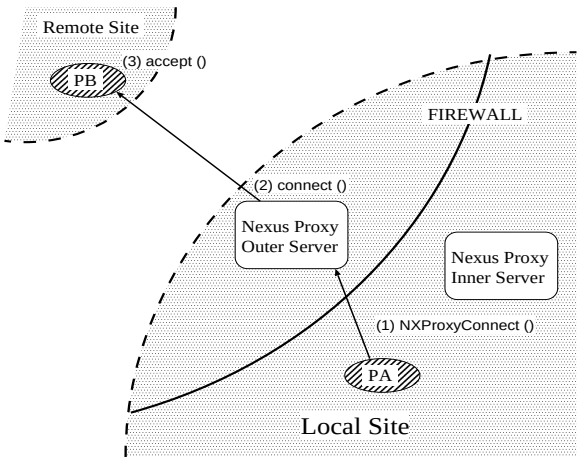


Figure 3. Communication mechanism via the Nexus Proxy (active connection)

$ProcessA(PA)$ is running in the local site which has a firewall and $ProcessB(PB)$ is running on a remote site which has no firewall. The Nexus Proxy inner server and the outer server bind a port called *nxport* and listen on it. Communication packets from the Nexus Proxy outer server to the Nexus Proxy inner server via *nxport* must be allowed. Figure 3 illustrates what happens when PA intends to connect to PB.

1. PA calls $NXProxyConnect()$ instead of

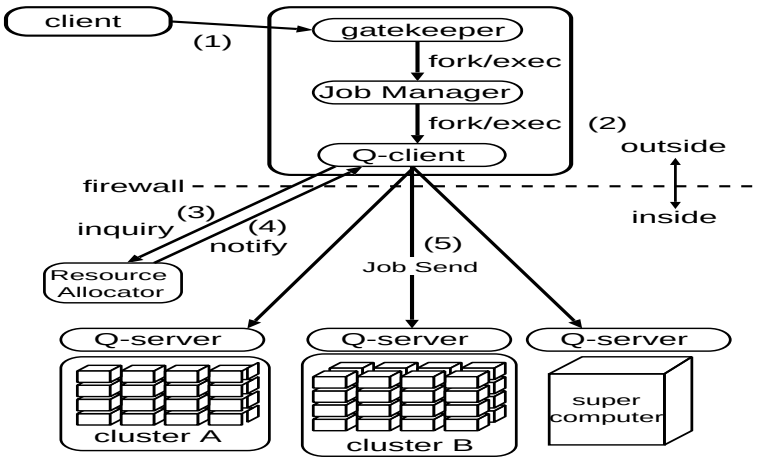


Figure 2. The architecture of RMF.

Table 1. Nexus Proxy Library Functions

Function	Description
$NXProxyConnect()$	Sends a connect request to the Nexus Proxy outer server and returns a file descriptor on which the client can communicate with the destination process.
$NXProxyBind()$	Sends a bind request to the Nexus Proxy outer server and returns a file descriptor on which the client can listen for requests.
$NXProxyAccept()$	Tries to accept a connection request.

connect(). Then, a connect request is sent to the Nexus Proxy outer server.

2. When the Nexus Proxy outer server receives a request from PA, a communication link between PA and the outer server is established. The outer server connects to PB via the *connect*() function.

3. When PB accepts the connect request from the outer server, a communication link between PA and PB is established through the outer server. From that point on, the outer server relays the communication between PA and PB.

Figure 4 illustrates what happens when PA intends to bind a port and listen for connect requests, and then PB tries to connect to PA.

1. PA calls $NXProxyBind()$ instead of *bind*(). $NXProxyBind()$ returns a port number on which PA can indirectly accept connection requests from clients.

2. The outer server binds a port on which the server listens for connect requests.

3. When PB intends to connect to PA, PB must connect to the outer server instead of PA.

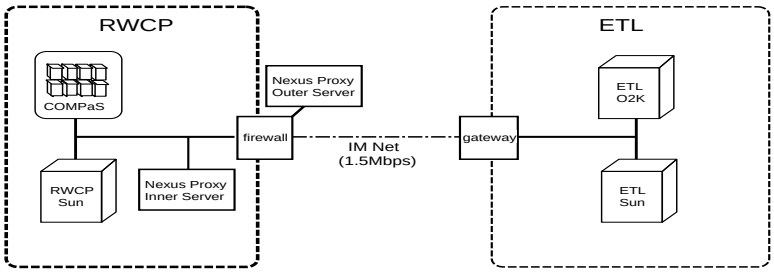
4. When the outer server accept the connect request from PB, the outer server connects to the inner server. Then, the inner server connects to PA.

5. PA calls $NXProxyAccept()$ instead of *accept*() to accept a connect request on the port returned by $NXProxyBind()$. When PA accepts a connection request from the inner server, the communication link between PA and PB is established through the inner server and the outer server.

In the Nexus Proxy system, only the communication port from the outer server to the inner server must be opened in advance, i.e. the firewall must be configured to open the port.

The basic mechanism of the Nexus Proxy is similar to the SOCKS protocol. Since the SOCKS protocol does not support the handling of passive open sockets however, we can not utilize SOCKS as a proxy server in the Globus system.

We have built the Nexus Proxy system into Globus system. We modified the source code of Globus to use $NXProxyConnect()$ and $NXProxyBind()$ instead of *connect*() and *bind*(). Address information for the communication startpoint/endpoint should also be changed to indicate the Nexus



site	nickname	system
RWCP	RWCP-Sun	Sun Enterprise 450 (4CPU)
RWCP	COMPaS	Pentium Pro SMP cluster (4CPU × 8nodes)
ETL	ETL-Sun	Sun Enterprise 450 (6CPU)
ETL	ETL-O2K	SGI Origin 2000 (16CPU)
RWCP	Inner Server	Sun Ultra Enterprise 450 (2CPU)
RWCP	Outer Server	Sun Ultra 80 (2CPU)

Figure 5. Experimental Environment

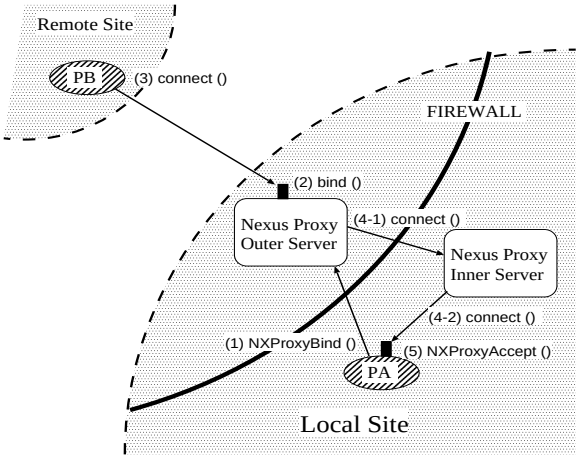


Figure 4. Communication mechanism via the Nexus Proxy (passive connection)

Proxy server. A communication utilizes the Nexus Proxy system when environment variables $NEXUS_PROXY_OUTER_SERVER$ and $NEXUS_PROXY_INNER_SERVER$ are defined. Otherwise, the original communication is done.

4 Results of the Experiment

We ran the 0-1 knapsack problem on our firewall-enabled Globus-based wide-area cluster system.

4.1 Experimental Environment

We have installed the Globus system, which is patched to use the Nexus Proxy if needed, at the RWCP Tsukuba Research Center and the Electrotechnical Laboratory. An experimental environment is shown in Figure 5. COMPaS[8] consists of eight quad-processor Pentium Pro (200MHz) SMPs connected by a 100Base-T Ethernet. RWCP has a firewall and RWCP-Sun and COMPaS cannot be accessed directly from the Internet. Although ETL also has a firewall, ETL-Sun and ETL-O2K can be accessed directly from RWCP.

4.2 Performance of the Nexus Proxy

In order to evaluate the performance of the Nexus Proxy, we measured the communication latency and the bandwidth between 1) RWCP-Sun and COMPaS, 2) RWCP-Sun and ETL-Sun. In both cases, we measured the latency and the bandwidth for the direct communications and the indirect (through the Nexus Proxy) communications[1].

- In direct communications between RWCP-Sun and COMPaS, communication packets pass through the intranet.
- In indirect communications between RWCP-Sun and COMPaS, communication packets pass through the Nexus Proxy outer server and the Nexus Proxy inner server.

[1] For the experiments, we have temporarily changed the configuration of the firewall to enable direct communication between RWCP-Sun and ETL-Sun.

Table 2. Communication latency and bandwidth

	latency	bandwidth(4096byte message)	bandwidth (1MB message)
RWCP-Sun ↔ COMPaS (direct)	0.41 msec	3.29 MB/sec	6.32MB/sec
RWCP-Sun ↔ COMPaS (indirect)	25.0 msec	70.5 KB/sec	538.6 KB/sec
RWCP-Sun ↔ ETL-Sun (direct)	3.9 msec	112.0 KB/sec	174.4 KB/sec
RWCP-Sun ↔ ETL-Sun (indirect)	25.1 msec	109.5 KB/sec	176.1 KB/sec

Table 3. Experimental Testbed

Nickname	Description
COMPaS	**8 processors**, 1 processor on each node. mpich **ch_p4 device** is used.
ETL-O2K	**8 processors** on ETL-O2K. vendor provided mpi is used.
Local-area Cluster	**RWCP-Sun + COMPaS**. total **12 processors**, 4 on RWCP-Sun, and 8 on COMPaS. mpich **Globus device** which utilize the Nexus Proxy is used.
Wide-area Cluster	**RWCP-Sun + COMPaS + ETL-O2K**. total **20 processors**, 4 on RWCP-Sun, 8 on COMPaS, and 8 on ETL-O2K. mpich **Globus device** which utilize the Nexus Proxy is used.

- In communications between RWCP-Sun and ETL-Sun, communication packets pass through the Internet (1.5Mbps IMNet).

Table 2 shows the results of the experiment.
In indirect communications between RWCP-Sun and COMPaS, the latency is 60 times larger and a drop in bandwidth for 4KB and 1MB message is order of magnitude compared to direct communications. Since the network between RWCP-Sun and COMPaS utilizes a 100Base-T Ethernet, the overhead of the Nexus Proxy is clearly revealed. Since both of COMPaS and RWCP-Sun utilize the Nexus Proxy, bandwidth for 4KB message is smaller than the bandwidth between RWCP-Sun and ETL-Sun. For communications between RWCP-Sun and ETL-Sun, the network latency when utilizing the Nexus Proxy is approximately six times larger compared to direct communications. This is caused by the increase in the number of hops when utilizing the Nexus Proxy. As message size increases however, the bandwidth when utilizing the Nexus Proxy is close to the bandwidth of the direct communication. Between RWCP-Sun and ETL-Sun, the overhead of the Nexus Proxy can be negligible when the message size is large.

4.3 Implementation of the 0-1 knapsack problem

We have implemented the 0-1 knapsack problem using MPICH-G. The algorithm is based on the branch-and-bound algorithm. Each node of a search tree is represented by a set of *index*, *value*, and *capacity*. Here, *index* is the index of the first item, which is not fixed yet, *value* is the sum of the profits of items which are already fixed to 1, and *capacity* is the sum of the weights. The search tree is represented by a stack onto which nodes are pushed in a search procedure. For a parallel implementation of the knapsack problem, we have given great care to the following issues:

1. The search trees tend to be highly irregular so that dynamic load balancing among processors is needed. Furthermore, since the wide-area cluster system is a heterogeneous cluster, the dynamic load balancing is more important.

2. One goal of our implementation is to make communication overhead caused by procedures for dynamic load balancing as small as possible.

We have used the master-slave algorithm. The following gives details of the algorithm:

- A master reads a data file and pushes a root node onto the stack.

- The **branch** operation represents the following procedures:

    1. pop a node from a stack
    2. check the node

Table 4. Execution time for the 0-1 knapsack problem

System	Num. of processors	Execution Time (sec)	Speedup
RWCP-Sun	1	26547	1
COMPaS	8	3135	8.47
ETL-O2K	8	6723	3.95
Local-area Cluster	12	2936	9.04
Wide-area Cluster (use Nexus Proxy)	20	2074	12.80
Wide-area Cluster (Not use Nexus Proxy)	20	2003	13.25

3. if the node has sub nodes, push them (one or two sub nodes) onto a stack.

- The master repeats the **branch** operation *interval* times.
- The master sends *stealunit* nodes on top of its stack to the slave which has sent a **steal** request to the master.
- If a slave has sent back nodes to the master, the master receives them and pushes them onto the stack.
- A slave repeats the **branch** operation until its stack becomes empty. If the stack is empty, the slave sends a **steal** request to the master.
- A slave sends back *backunit* nodes when the slave has too many nodes on the stack.

Our algorithm is based on a self-scheduling algorithm. The algorithm is considered to be suitable for distributed heterogeneous metacomputing environments since it performs dynamic load balancing with low overhead, i.e. a slave tries to steal its master's nodes when the slave's stack becomes empty. *interval* is the frequency of the master's check of a slave's steal requests, and *stealunit* is the amount of nodes to steal.

4.4 Performance when running the 0-1 Knapsack Problem

We ran the 0-1 knapsack problem on four local/wide-area cluster systems shown in Table 3. When solving the knapsack problem using branch-and-bound algorithm, the execution time is heavily affected by the characteristics of input data. In order to evaluate the performance characteristics of the cluster system clear and normalize the problem, we used such data as **no branches were pruned**, meaning entire search space is traced by processes. The number of items was 50. We varied a *stealunit*, *interval*, and *backunit* and took the best combination. In order to observe the performance of the Nexus Proxy system, we ran the knapsack problem on Wide-area Cluster under the following two conditions:

1. Communications between RWCP and ETL utilize the Nexus Proxy system.
2. Communications between RWCP and ETL did not utilize the Nexus Proxy system. In order to do this, we modified the configuration of the firewall temporarily.

Table 4 shows the execution time and the speedup of the 0-1 knapsack problem on the four clusters. For the measurement of the speedup, we ran the sequential version of the 0-1 knapsack problem on RWCP-Sun, and its execution time was used to calculate the speedup. By comparing the execution time on the wide-area cluster system, we can observe that the overhead of the Nexus Proxy is approximately 3.5% and this can be negligible under the wide-area network computing environment. We obtained a reasonable performance on COMPaS and Local-area Cluster. Table 5 shows the total number of steal request handled by the master, and the maximum, minimum, average number of steal request by the clients on Local/Wide-area Clusters. The number of steals indicates the grain of stolen jobs. When the job is coarse grained, the number of steal requests, i.e. the number of communications decreases. On the other hand, when the job is fine grained, we can expect to obtain good load balance. Table 6 shows the number of nodes which is traversed by the master, and the maximum, minimum, average number of nodes traversed by the clients on Local/Wide-area Clusters. The number of nodes in the table is shown in billions. We can observe that slaves frequently send a steal request to the master. As a result, although the communication overhead increased, we obtained good load balance and reasonable performance even in a Wide-area Cluster System.

Table 5. Number of steals

System	Master	RWCP-Sun			COMPaS			ETL-O2K		
		Max	Min	Average	Max	Min	Average	Max	Min	Average
Local-area Cluster	160459	13869	15649	14981	17219	11385	14436			
Wide-area Cluster	217330	11603	8394	10563	13289	8007	11465	8508	2105	5693

Table 6. Number of traversed nodes (in billions)

System	Master	RWCP-Sun			COMPaS			ETL-O2K		
		Max	Min	Average	Max	Min	Average	Max	Min	Average
Local-area Cluster	2.66	4.56	4.44	4.48	4.70	4.34	4.50			
Wide-area Cluster	1.47	3.43	3.17	3.27	3.49	3.10	3.25	2.03	1.74	1.85

5 Summary

We have reported the performance of a firewall-enabled Globus-based wide-area cluster system. In order to spread the global computing environment over various sites, a mechanism to handle a firewall is needed in the software architecture of the global computing environment. A proxy mechanism is one solution. Although the communication latency increases by several times, the overhead of the proxy system can be negligible for large message sizes in metacomputing environments.

We used a tree search problem as a benchmark to measure the performance of the wide-area cluster system. Since a parallel tree search problem has a coarse grained and asynchronous parallelism, it is considered suitable for metacomputing environments. We implemented the parallel 0-1 knapsack problem, which performs dynamic load balancing with low overhead. As a result, our experimental results have proved that metacomputing environments can provide good performance for such problems.

References

[1] I. Foster et.al., "The GRID: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers (1998).

[2] I. Foster, et.al., "Software Infrastructure for the I-WAY high performance distributed computing experiment", Proc. 5th IEEE Symp. on High Performance Distributed Computing, pp. 562–572 (1996).

[3] I. Foster et.al., "The Globus Project: A status report", Proc. Heterogeneous Computing Workshop, pp. 4–18 (1998).

[4] I. Foster et.al., "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems", Proc. Supercomputing (1998).

[5] I. Foster et.al., "The Nexus approach to integrating multithreading and communication", Journal of Parallel and Distributed Computing, Vol. 37, pp. 70–82 (1996).

[6] G. Mahinthakumar et.al., Nicholas T. Karonis, "Multivariate Geographic Clustering in a Metacomputing Environment using Globus", Proc. Supercomputing (1999).

[7] T. Kielmann et.al., R. A. F. Bhoedjang, "MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems", Proc. Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 131–140 (1999).

[8] Y. Tanaka et.al., "COMPaS: A Pentium Pro PC-based SMP Cluster and its Experience", Proc. Int'l Workshop on PC-NOW'98 (1998) (to appear).

[9] Y. Tanaka et.al., "Resource Manager for Globus-based Wide-area Cluster Computing", 1st IEEE International Workshop on Cluster Computing (IWCC'99), pp. 237–244 (1999).

[10] S. Martello et.al., "KNAPSACK PROBLEMS – Algorithms and Computer Implementations", Wiley-Interscience Series in Discrete Mathematics and Optimization (1989).