High Performance Relay Mechanism for MPI Communication Libraries Run on Multiple Private IP Address Clusters

Ryousei Takano [1,2], Motohiko Matsuda [3], Tomohiro Kudoh [1]
Yuetsu Kodama [1], Fumihiro Okazaki [1], Yutaka Ishikawa [3,1], Yasufumi Yoshizawa [4]

[1]*Grid Technology Research Center*
National Institute of Advanced Industrial Science and Technology (AIST), Japan
[2]*AXE, Inc.*
[3]*University of Tokyo*
[4]*Tokyo University of Agriculture and Technology*

***Abstract*— We have been developing a Grid-enabled MPI communication library called GridMPI, which is designed to run on multiple clusters connected to a wide-area network. Some of these clusters may use private IP addresses. Therefore, some mechanism to enable communication between private IP address clusters is required. Such a mechanism should be widely adoptable, and should provide high communication performance. In this paper, we propose a message relay mechanism to support private IP address clusters in the manner of the Interoperable MPI (IMPI) standard. Therefore, any MPI implementations which follow the IMPI standard can communicate with the relay. Furthermore, we also propose a trunking method in which multiple pairs of relay nodes simultaneously communicate between clusters to improve the available communication bandwidth. While the relay mechanism introduces an one-way latency of about 25 μsec, the extra overhead is negligible, since the communication latency through a wide area network is a few hundred times as large as this. By using trunking, the inter-cluster communication bandwidth can improve as the number of trunks increases. We confirmed the effectiveness of the proposed method by experiments using a 10 Gbps emulated WAN environment. When relay nodes with 1 Gbps NICs are used, the performance of most of the NAS Parallel Benchmarks improved proportional to the number of trunks. Especially, using 8 trunks, FT and IS are 4.4 and 3.4 times faster, respectively, compared with the single trunk case. The results showed that the proposed method is effective for running MPI programs over high bandwidth-delay product networks.**

I. Introduction

With the recent progress in network technology, running large-scale applications using a set of geographically distributed computing resources has become practical. Several Grid-enabled MPI systems, including MPICH-G2 [7], PACX-MPI [8], StaMPI [9], and MC-MPI [10], have been proposed for this purpose. Using these systems, users are able to seamlessly deploy their applications from a small-scale cluster system at a laboratory to a Grid environment, and process a very large data set that is too large to run on a single system.

When such a Grid environment is under the control of several organizations, we can not assume that all the compute nodes have global IP addresses and can communicate with each other directly. Some of the clusters may use private IP addresses and/or firewalls. Therefore, for broad deployment of Grid-enabled MPI systems, private IP address clusters should be supported.

User level proxy, virtual private networks (VPNs), and network address translation (NAT) traversal [14] are possible methods to enable communication between private IP address nodes, and proposed MPI systems which support private IP addresses use at least one of them. Each of these methods has advantages and disadvantages. A user-level proxy is portable and easy to implement. In addition, setting of network parameters for wide-area networks (e.g., rules of packet filtering, socket buffer size, etc.) can be consolidated on the proxy node. However, its performance (communication bandwidth) is lower than that of the others, because an extra communication between the proxy process and the kernel is necessary. VPN can tunnel NAT or firewall boxes. However, all the nodes that participate in communication should have different IP addresses. TCP NAT traversal techniques have higher performance, but they are not standardized, and not always applicable, depending on the behavior of NAT boxes.

We assume an environment in which the inter-cluster bandwidth (e.g., 10 Gbps) is higher than the bandwidth of the network interface (e.g., 1 Gbps) of each cluster node, and the aggregate bandwidth of nodes in a cluster is higher than the inter-cluster bandwidth. In such an environment, when there is only one pair of relay nodes (one at each cluster), the available bandwidth for inter-cluster communication is limited by the bandwidth of the network interface of the front-end node at each cluster. Therefore, the relay node may become the bottleneck of the performance. For example, when the front-end node has a Gigabit Ethernet interface, the bandwidth is limited to 1 Gbps, even if the inter-cluster network bandwidth is higher. To utilize the inter-cluster bandwidth efficiently, multiple relay nodes at each cluster should participate in the communication simultaneously.

We have developed GridMPI [1][2], which complies with the MPI-1.2 and MPI-2.0 standards. GridMPI is implemented

978-0-7695-3156-4/08 $25.00 © 2008 IEEE
DOI 10.1109/CCGRID.2008.88

using YAMPI [3] for intra-cluster communication, and supports the Interoperable MPI (IMPI) protocol [4] for inter-cluster communication over TCP/IP. [1]

In this paper, we propose a message relay mechanism to support private IP address clusters, which is transparent for MPI processes that communicate using the IMPI protocol. We present a user level proxy implementation called the IMPI Relay. We choose a user level proxy approach since it can be widely deployed, regardless of the configuration of NAT boxes. Furthermore, to solve the problem of lower bandwidth of the user level proxy, we will introduce a trunking technique, which is a connection aggregation (bonding) using multiple pairs of the IMPI Relays.

In the IMPI protocol, each MPI process is identified by a pair consisting of a host id and a process id. The IMPI relay translates the source and destination ids in each packet to achieve communication between nodes in different private IP address clusters. The communication between the IMPI relays and global IP address clusters is subject to the IMPI standard. Therefore, any MPI implementation which follows the IMPI standard can communicate with the relay.

The rest of the paper is organized as follows. Section II describes the design of the IMPI Relay and the proposed trunking method. The experimental results obtained using a 10 Gbps emulated WAN environment are shown in Section III. In Section IV, we discuss an optimization of the proposed method. We briefly mention related work in Section V, and conclude the paper in Section VI.

II. IMPI Relay

A. Terminology

The following terms are used in this paper, and they are based on the terms defined in the IMPI specification.

- **Client**: An instance of a cluster used in a single IMPI application. An application may have several clients.
- **Proc**: A proc is equivalent to an MPI process, and it is identified by a combination of a `Host_ID` and a `Proc_ID`. Every proc has exactly one host. A host might have one or many procs.
- **Server**: An IMPI Server used to gather and distribute `Host_ID`s, `Proc_ID`s, and other information among clients. There is one IMPI Server for an application.
- **Agent**: A entity in a client, which contacts the IMPI Server on behalf of all processes of the client.
- **Packet**: An IMPI message is divided into packets, and they are sent in sequential order. The header field contains its source and destination proc, data length, and so on.

`Host_ID` is a system-wide unique host identifier. A host normally corresponds to a node in a cluster. Typically an IPv6 address is used for the purpose. [2] `Proc_ID` is a process identifier which is unique within a host. In the current GridMPI implementation, a host has one proc, and the host which has the smallest rank proc acts as an agent.

[1]The following document describes the implementation status and provides notes on the use of MPI-2 features. http://www.gridmpi.org/gridmpi-2-x/impl-status-mpi2.html

[2]For IPv4 networks, an IPv4-mapped IPv6 address is used.

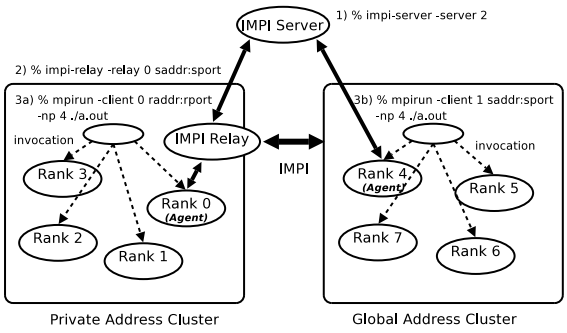


Fig. 1. Overview of GridMPI and IMPI Relay. IMPI Relay forwards IMPI command packets between the IMPI Server and the agent, and also forwards IMPI data packets between clusters.

B. Overview

An overview of GridMPI and IMPI Relay execution steps is shown in Figure 1. The left rounded-box shows a private IP address cluster and the right one shows a global IP address cluster. The IMPI Relay runs on a front-end node, which has both private and global IP addresses. The private address is used to communicate with nodes inside the cluster. The global address is used to communicate with the IMPI Server and nodes outside the cluster. The IMPI Relay forwards two types of packets. The first one consists of IMPI command packets. IMPI command packets are used to exchange client information between the IMPI Server and an agent, at the time of initialization (i.e., `MPI_Init`) and finalization (i.e., `MPI_Finalize`). From the viewpoint of the IMPI Server, IMPI Relay acts as an agent of the client. The second one consists of data packets exchanged among clusters.

The IMPI Relay translates both `IMPI_Host` and `IMPI_Proc` in each packet on the fly. Any implementation of the IMPI protocol should work with IMPI Relay. Using IMPI Relay, from the outside of the cluster, all procs inside the cluster are seen as a set of procs on the IMPI Relay node. On the other hand, from the inside, all procs outside the cluster are seen as a set of procs on the IMPI Relay node.

The execution flow is as follows. 1) IMPI Server is launched at a globally reachable node by an `impi-server` command. The command outputs the IP address/port pair (e.g., saddr:sport) which will be used by agents of public IP address clusters or the IMPI Relays of private IP address clusters to contact the server. 2) an `impi-relay` command is issued at a front-end node in a private IP address cluster to launch an instance of IMPI Relay. For this command, (saddr:sport) is provided as a parameter. The command outputs another IP address/port pair (e.g., raddr:rport), which will be used by procs inside the cluster to contact the relay. 3a, 3b) a `mpirun` command is issued at both clusters, to invoke MPI processes. For this invocation of MPI processes, at a

private IP address cluster, the IP address/port pair of the IMPI Relay of that cluster (raddr:rport) is used as the parameter of the "`-client`" option. On the other hand, at a global IP address cluster, the IP address/port pair of the IMPI Server (saddr:sport) is used as the parameter.

In the first half of the initialization phase, the agent first attempts to communicate with the IMPI Server to gather and distribute the client information. In a private IP address cluster, the agent establishes connection with the IMPI Relay in the cluster instead of the IMPI Server, and the agent sends IMPI command packets to the IMPI Server via the IMPI Relay. The IMPI Relay dynamically generates host-proc mapping tables, by snooping the IMPI command packets between the IMPI Server and the agent. In the second half of the initialization phase, each host establishes all-to-all connections according to the client information distributed by the IMPI Server. Then rank numbers are renumbered to make them globally unique, because they are only unique in each cluster.

After the initialization phase, the IMPI Relay transfers IMPI data packets from the internal nodes to the external nodes. At this time the IMPI Relay replaces the original source and destination proc fields, according to the host-proc mappings.

C. Host-proc id translation

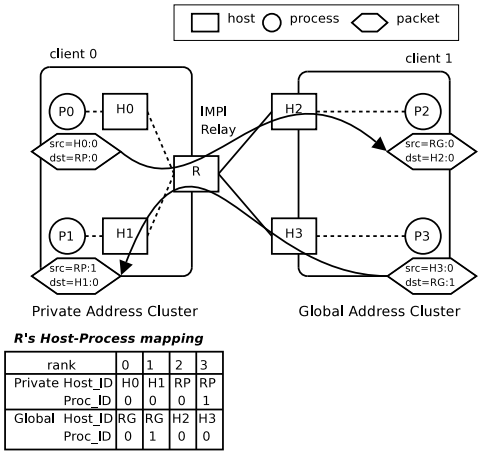
The IMPI Relay serves as the *gateway* host of a private IP address cluster as shown in Figure 2. The dashed-lines show host-proc id mappings. For a global IP address cluster (i.e., cluster 1), one proc is paired with one host. On the other hand, for a private IP address cluster (i.e., cluster 0), the IMPI Relay acts as a host, and all procs of the cluster at the other side are seen to be procs belonging to that host. "src=H0:0" in each packet indicates that the source `Host_ID` and `Proc_ID` are $H0$ and 0, respectively. For example, a packet is sent from $P0$ to $P2$. R replaces the source and destination proc fields of the packet, and forwards it to $H2$.

To realize this remapping, the IMPI Relay maintains two individual mapping tables of host-proc id translation for both the internal and the external cluster. The global host-proc mapping tables are identical among IMPI Relays. The private host-proc mapping tables, by contrast, differ from each other. From client 1, procs at client 0 are mapped to 2 procs $\{P0, P1\}$ of the host RG, where RG is a global IP address of R. From client 0, procs at client 1 are mapped to 2 procs $\{P2, P3\}$ of the host RP, where RP is a private IP address of R. Note that the IMPI Relay does not preserve the original host-proc relationship, and flattens one-tier mapping at each client.

Figure 3 shows the case in which clients on both sides are private IP address clusters. For example, $P0$ sends a packet to $P2$. The packet is forwarded to $H0$, $R0$, $R1$, and $H2$. Both $R0$ and $R1$ perform host-proc id translation according to the respective host-proc mapping tables.

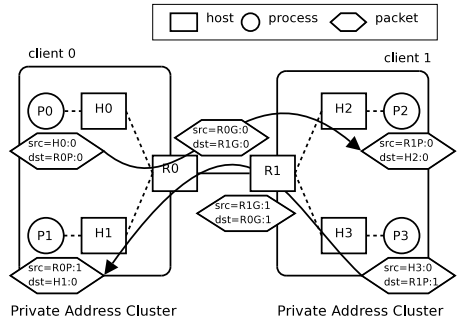
D. Trunking

To improve communication performance, we propose trunking, a connection aggregation technique using multiple pairs



rank	0	1	2	3
Private Host_ID	H0	H1	RP	RP
Proc_ID	0	0	0	1
Global Host_ID	RG	RG	H2	H3
Proc_ID	0	1	0	0

Fig. 2. IMPI Relay (P-G). IMPI Relay forwards packets, and replaces the source and destination fields according to the host-proc mapping.



rank	0	1	2	3
Private Host_ID	H0	H1	R0P	R0P
Proc_ID	0	0	0	1
Global Host_ID	R0G	R0G	R1G	R1G
Proc_ID	0	1	0	1

rank	0	1	2	3
Private Host_ID	R1P	R1P	H2	H3
Proc_ID	0	1	0	0
Global Host_ID	R0G	R0G	R1G	R1G
Proc_ID	0	1	0	1

Fig. 3. IMPI Relay (P-P). Each IMPI Relay has a host-proc mapping table, and host-proc id translation is performed at both $R0$ and $R1$.

of the IMPI Relay. Multiple front-end nodes are used for inter-cluster communication. Trunking can be realized as a straightforward extension to the host-proc id translation. We propose two schemes for the mapping, as shown in Figure 4. This figure shows a trunking implementation using two IMPI Relays at the private IP address cluster. In both schemes, IMPI Relays and hosts with global IP addresses establish all-to-all connections in the manner of IMPI. In scheme#1, packets are forwarded by an IMPI Relay selected according to the destination host. On the other hand, in scheme#2, an IMPI Relay is selected according to the source host. In this paper, we employ scheme#1, since it can be implemented with less modification than that required by scheme#2.

In scheme#1, 2 hosts $\{R0, R1\}$ and 4 procs $\{P0, P1, P2, P3\}$ establish all-to-all connections. The behavior of each proc is the same as that without IMPI Relay. Each IMPI Relay has an identifier called a relay rank, and

the IMPI Relay which has the smallest relay rank is called a *leader*. Only the leader of IMPI Relays acts as an agent of the client.

For the external cluster (i.e., client 1), IMPI Relays use a mapping scheme in which $R0$ represents 2 procs $\{P0, P1\}$, and $R1$ represents 2 procs $\{P2, P3\}$. For the internal cluster (i.e., client 0), they use a mapping scheme in which $R0$ represents 2 procs $\{P4, P5\}$, and $R1$ represents 2 procs $\{P6, P7\}$. However, this scheme has an issue such that the outbound route and inbound route are not the same, but are asymmetric. For example, the round-trip route between $P0$ and $P7$ is $H0 \rightarrow R1 \rightarrow H7 \rightarrow R0 \rightarrow H0$. When $R0$ forwards a packet to $H0$, the source `Host_ID` is replaced from $H7$ to $R1$. Here, although $P0$ expects that $P7$ is to be represented by $R1$, $P0$ receives the packet whose source `Proc_ID` is $P7$ from $R0$. In terms of inside the private IP address cluster, this behavior results in a discrepancy with the IMPI protocol. Hence we need to slightly modify the GridMPI implementation so as to relax the validation check to allow the difference between the source `Host_ID` (i.e., $R1$) and the actual source IP address (i.e., $R0$).

Scheme#2 looks straightforward, and the round-trip route is symmetric. In addition, it does not establish all-to-all connections. Therefore, the number of connections inside a private IP address cluster is smaller than that of scheme#1. For client 1, IMPI Relays use a mapping scheme in which $R0$ represents 2 procs $\{P0, P1\}$, and $R1$ represents 2 procs $\{P2, P3\}$, which are the same as most of those of scheme#1. On the other hand, for client 0, IMPI Relays use respective mapping tables, in which $R0$ and $R1$ represent the same set of 4 procs $\{P4, P5, P6, P7\}$ each other. To provide different mappings to the IMPI Relays, the IMPI Relays should be invoked by different agents. Therefore, multiple agents should be used for one cluster. This results in a drastic extension of the IMPI protocol in terms of the intra-cluster communication of the private IP address cluster. Note that for both schemes, inter-cluster communication follows the IMPI standard. Therefore, any IMPI standard-based MPI implementation on public IP address clusters should work with IMPI Relays on private IP address clusters. To use an IMPI standard-based MPI implementation other than GridMPI for intra-cluster communication of private IP address clusters, slight modification of the implementation may be required.

III. Experiment

A. Experimental setting

Figure 5 shows the experimental setting, and Table I shows the specifications of the node PC and the switches.[3] Each cluster consisted of 16 compute nodes and 8 relay nodes. A compute node has both Gigabit Ethernet and Myrinet-2000 interfaces. Gigabit Ethernet was used for inter-cluster communication, and Myrinet/MX was used for intra-cluster

[3]We used a single S5648 switch at each cluster, each of which was divided into two by using a VLAN. Figure 5 lacks connections between the bottom switches (i.e., VLANs) and GtrcNET-10, which are used for 'G-G' setting.

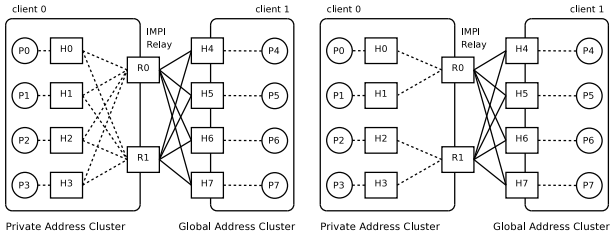


Fig. 4. Trunking using multiple IMPI Relays. Scheme#1 forwards packets according to the destination host; scheme#2 forwards packets according to the source host.

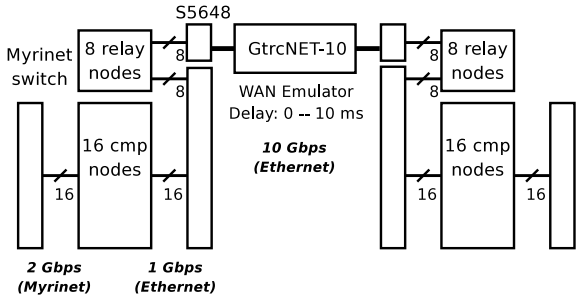


Fig. 5. Experimental Setting

communication. A relay node has two Gigabit Ethernet interfaces, with a private IP address is assigned to one of the interfaces, and a global IP address is assigned to the other.

Two clusters were connected through a 10 Gbps WAN emulator called GtrcNET-10 [5][6]. GtrcNET-10 consists of a large-scale Field Programmable Gate Array (FPGA), three 10 Gbps Ethernet XENPAK ports, and three blocks of 1 GB DDR-SDRAM. The FPGA is a Xilinx XC2VP100, which includes three 10 Gbps Ethernet MAC and XAUI interfaces. GtrcNET-10 provides many functions, such as traffic monitoring in microsecond resolution, traffic shaping, and WAN emulation at 10 Gbps wire speed. In the experiment, GtrcNET-10 was used to insert delay for both directions, which varied from 0 msec to 10 msec, one-way.

Some networking parameters of the Linux kernel were changed from the default value, as shown in Table II, because default parameters, such as socket buffer sizes, are not adequate for the experiment. In addition, tcp_no_metrics_save disables reuse of the parameters of the previous connection.

IMPI_C_DATALEN is the maximum IMPI packet size in bytes. The default value is 64 KBytes. IMPI uses the rendezvous protocol instead of the eager protocol if the size of an MPI message is larger than the value of IMPI_C_DATALEN.

B. Micro benchmarks

1) Latency: Table III shows the one-way latency between two clusters. Round-trip latency was measured by a *ping-pong* program using a 0 byte data message, and divided by two to get the one-way latency. The labels 'G' and 'P' indicate

TABLE I
PC CLUSTER SPECIFICATIONS

Node PC	
CPU	Opteron/2.0 GHz dual
Memory	6 GB DDR333
Ethernet	Broadcom BCM5704
Myrinet	Myricom M3F-PCIXD-2 (MX-1.2.2)
OS	SuSE Enterprise Server 9
Kernel	Linux 2.6.23
Compiler	Intel Compiler 9.1
Switch	
Huawei-3Com Quidway S5648 + optional 10 Gbps port Myricom M3-SW16-8F + M3-SPINE-8F	

TABLE II
PARAMETERS

sysctl parameters	
net.core.wmem_max	2500000
net.core.rmem_max	2500000
net.ipv4.tcp_rmem	2500000 2500000 2500000
net.ipv4.tcp_wmem	2500000 2500000 2500000
net.ipv4.tcp_no_metrics_save	1
Environment variables for GridMPI	
_YAMPI_SOCBUF	65536
IMPI_SOCBUF	2500000
IMPI_RELAY_SOCBUF	2500000

a global IP address cluster and a private IP address cluster, respectively. The global IP address node communicates with the cluster at the other side directly, and the private IP address node communicates via the IMPI Relay. The one-way latency increases by about 25 μsec per each IMPI Relay.

The IMPI Relay stores an incoming packet in the buffer and forwards it. Therefore, the extra latency is introduced according to the size and the number of the packet. The one-way latency increases by $(N_P + N_R)T$, where N_P is the number of packets within a message, N_R is the number of IMPI Relays, and T is the transmission time of a packet. When the packet size is set to 64 KBytes, T becomes 512 μsec at gigabit wire speed.

2) Point-to-point communication performance: Figure 6 shows the *point-to-point* communication bandwidth between two clusters. The bandwidths of both 'P-G' and 'P-P' are lower than those of 'G-G' with small messages, because the latency increases due to the overhead of the IMPI Relay, as shown in Table III. When the message size is larger than 4 KBytes, the results of all cases are almost the same. The bandwidths drop at 256 KBytes, since the IMPI communication protocol changes from the eager protocol to the rendezvous protocol at this point. The rendezvous protocol yields the extra communication latency, and the performance degrades as the delay increases. Using the IMPI Relay during the rendezvous protocol, the bandwidth decreases $N_P/(N_P + N_R)$ times compared with 'G-G'. Therefore, the overhead of the IMPI Relay relatively decreases as the message size increases. For example, the degradation ratio becomes $2/3$ with 256 KBytes of packet and 'P-P'. The experimental result corresponds

TABLE III
ONE-WAY LATENCY BETWEEN TWO CLUSTERS

	G-G	P-G	P-P
latency (μsec)	29.8	54.45	78.9

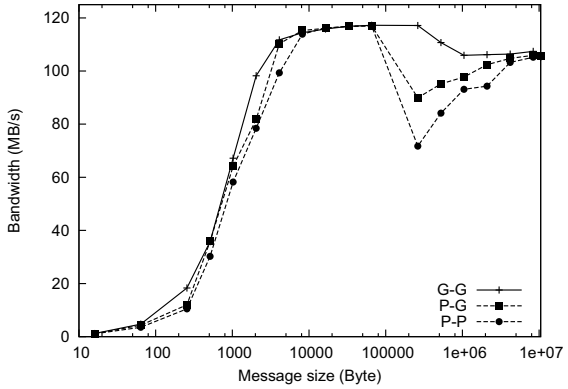


Fig. 6. Point-to-point communication bandwidth between two clusters

approximately to it.

3) All-to-all communication performance: Figure 7 shows the *all-to-all* communication execution time with 32 processes (16 processes for each cluster). *All-to-all* is a collective operation exchanging data from all processes to all processes. The performance depends mostly on the bisection bandwidth. The label 'no relay' shows the execution time with 'G-G'; the labels 'N trunk' show the execution time with 'P-P' and trunking N pairs of IMPI Relays.

In the case where the message size is smaller than 1024 Bytes, there is no improvement in performance due to trunking, since collective operations with message sizes less than 1024 Bytes use the *short* algorithm defined in the IMPI specification. [4] In the *short* algorithm, a master node gathers and distributes data from nodes inside the cluster, and the master node only participates in the inter-cluster communication. Therefore, the effect of trunking can not be obtained.

In the case where the message size is larger than 1024 Bytes, the performance increases as the number of trunks increases. However, we observed a non-intuitive phenomenon when the message size ranged from 4096 Bytes to 512 KBytes. In particular, the performance of 'no relay' is worst when the message size is between 4096 Bytes and 32 KBytes. We also observed a linearly increasing number of packet losses as the message size increases. However, the number of retransmission timeouts (RTOs) are 195, 21049, and 2303 when the message sizes are 1024 Bytes, 4096 Bytes, and 512 KBytes, respectively.

These phenomena can be explained as follows. The aggregate inter-cluster communication traffic (maximum of 16 Gbps) exceeds the physical bandwidth of the inter-cluster

[4]The cross-over point can be specified by the environment variable IMPI_COLL_XSIZE. The default value is 1024 Bytes.

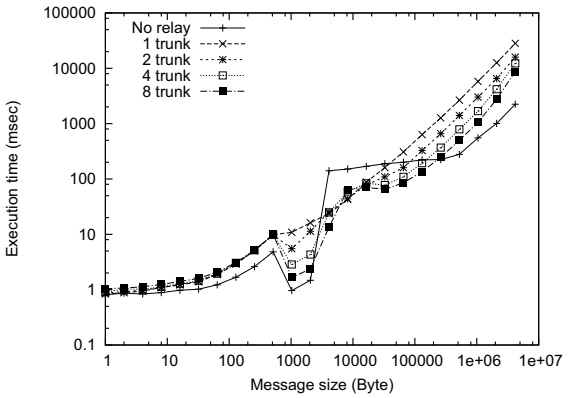


Fig. 7. *All-to-all* communication execution time with 32 processes (16 processes for each cluster)

TABLE IV
RESULTS OF NPB BENCHMARKS WITH 0 MSEC OF DELAY (MOP/S TOTAL)

	CG	EP	FT	IS	LU	MG
no relay	3998.2	615.3	8205.0	73.8	20317.6	14523.6
1 trunk	1400.4	619.2	1493.2	46.2	18035.2	5755.7
2 trunk	2287.4	616.4	2860.0	82.9	18792.8	8776.1
4 trunk	2660.3	612.6	4575.3	120.4	19177.5	10781.2
8 trunk	2978.8	616.6	6585.6	159.1	19657.3	11884.4

	BT	SP
no relay	18468.4	7733.5
1 trunk	12498.9	4180.3
2 trunk	13658.7	5080.2
4 trunk	15917.7	6116.7
8 trunk	16873.5	6950.1

link (10 Gbps). Packets are queued at the switch at the entry point of the inter-cluster link, and some packets are discarded. When packets at the tail of a message within an `MPI_Alltoall` function are dropped, the receiver can not recognize the loss of packets since there are no successive packets within the function. Therefore, the sender can not retransmit the lost packet until it recognizes the loss by an RTO. Execution of the program is suspended while it is waiting for the RTO, and thus it can not proceed to the next iteration. In the Linux TCP implementation, the RTO is approximately $RTT + 200\ msec$. Thus, the communication stops for 200 msec or longer, and results in low bandwidth utilization. Severe performance degradation was observed when many RTOs occur. Of course, this point depends on several parameters, including the number of nodes, the inter-cluster bandwidth, the delay, and the buffer size of switches.

C. NAS Parallel Benchmarks

We confirmed the effects of the trunking method by running NAS Parallel Benchmark 3.2, which is a typical high performance computing benchmark program. The problem size used was class B. The number of processes was 32 for the CG, EP, FT, IS, LU, and MG benchmarks, and 25 for the BT and SP benchmarks. In the latter case, compute nodes were divided into 13 and 12 at each cluster. We disabled use of the rendezvous protocol by setting IMPI_C_DATALEN to a very large value (0x7fffffff). We ran each benchmark three times and took the highest performance value among the results.

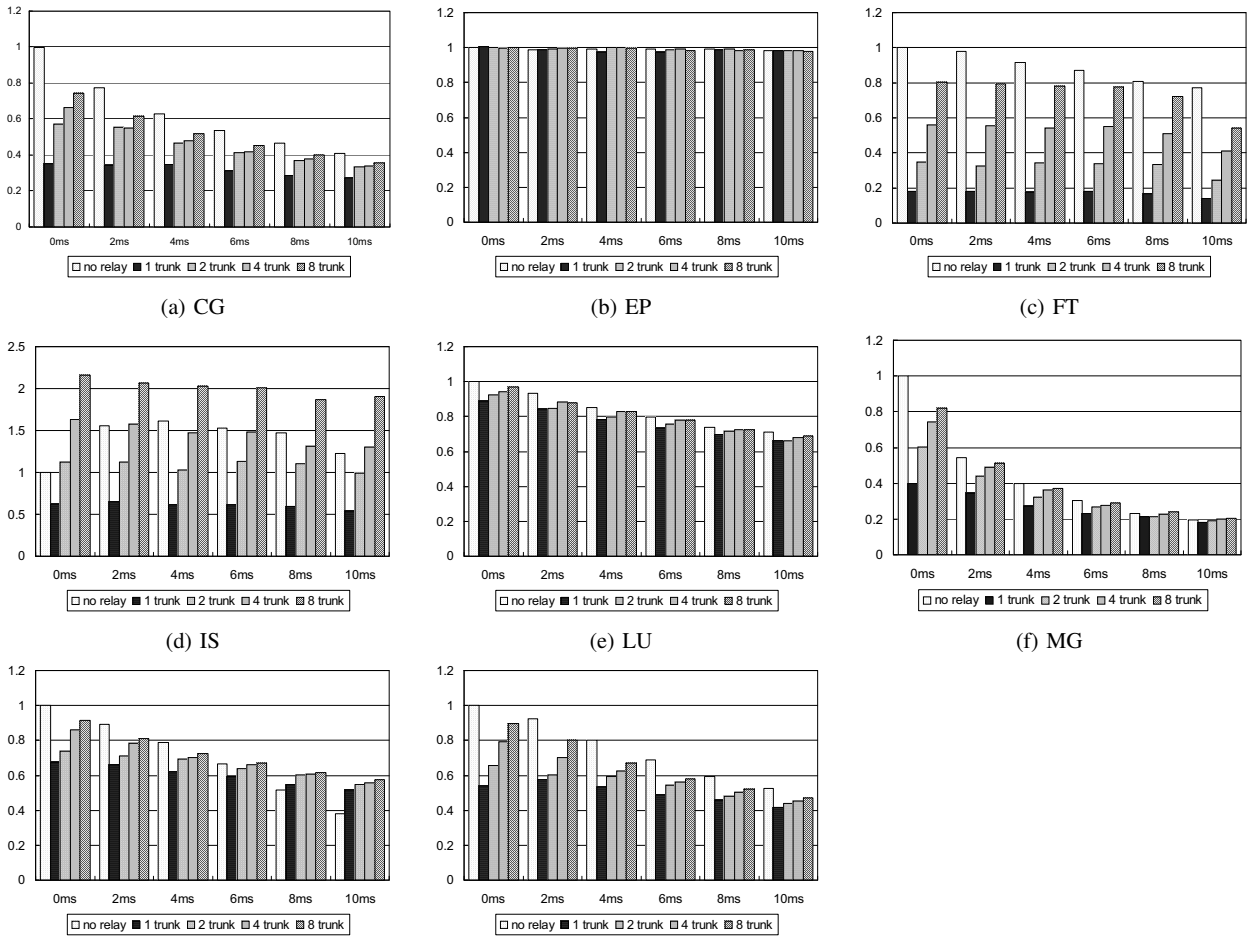
Table IV shows the absolute performance in Mop/s total with 0 msec delay. We compare the results of 'no relay' with those of trunking, where the number of trunks are 1, 2, 4, and 8. In most of the results, the performance increases in proportion to the number of trunks. When the number of trunks increases from 1 to 8, the performance improvement ratio of each benchmark is as follows. FT and IS perform 4.4 and 3.4 times faster, respectively. These benchmarks require larger bandwidth than the others, thus trunking significantly improves the performance. CG, MG, BT and SP perform 2.1, 2.1, 1.3, and 1.7 times faster, respectively. EP and LU show no significant effect on their performance, because these benchmarks generate very low traffic and the inter-cluster link is not fully utilized, even if no trunking is used.

A further distinctive point is that the IS benchmark results with two or more trunks outperform the 'no relay' cases. '2 trunk', '4 trunk', and '8 trunk' perform 1.1, 1.6, and 2.2 times faster, respectively. This can be considered as due to the same reason described in Section III-B.3, because the IS benchmark iteratively uses the `MPI_Alltoallv` function with about 128 KBytes of message. Heavy congestion is avoided since the number of nodes which simultaneously transfer messages through inter-cluster communication links is limited to the number of trunks.

Figure 8 shows the relative performance normalized to the 'no relay' with 0 msec of delay cases. The one-way delay was changed from 0 msec to 10 msec. In most of the results, the overhead of the IMPI Relay becomes relatively smaller as the delay increases, while the IMPI Relay introduces a latency of about 25 μsec, as shown in Table III. For IS, the results with the '8 trunk' cases always outperform the 'no relay' cases. Table V shows the total number of RTOs that occurred in the IS benchmark with 0 msec and 10 msec of delay. Basically, the number of RTOs increases linearly as the number of trunks increases. In the case of 'no relay' with 0 msec, however, the number of RTOs is significantly larger than that of the others and results in a performance degradation. Furthermore, in the case of 'no relay' with 10 msec, although the number of RTOs is smallest, the performance is lower than that of '4 trunk' and '8 trunk'. TCP communication is split by the IMPI Relay, and the congestion window size quickly grows. Therefore, the use of IMPI Relay makes the influence of the delay small. The effect can be considered similar to that with Performance Enhancing Proxies [15]. For BT, the results with the IMPI Relay become better than the 'no relay' cases when the delay is larger than 8 msec. These results indicate that the proposed trunking method is efficient for running MPI programs over high bandwidth-delay product networks.

IV. DISCUSSION

In this paper, we explained IMPI Relay communication mainly using two clusters, either or both of which is a private



(a) CG (b) EP (c) FT

(d) IS (e) LU (f) MG

(g) BT (h) SP

Fig. 8. Relative performance of NPB benchmarks normalized to the 'no relay' with 0 msec of delay cases

.

TABLE V
THE TOTAL NUMBER OF TCP RETRANSMIT TIMEOUTS IN THE IS

delay	no relay	1 trunk	2 trunk	4 trunk	8 trunk
0 ms	1457	61	59	170	493
10 ms	25	45	76	183	455

IP address cluster. When there are more than two clusters (e.g., N clusters), from the viewpoint of nodes inside a private IP address cluster, the IMPI Relay acts as multiple hosts (e.g., $N-1$ hosts), each of the hosts represent procs at one of the external clusters. We decided to employ this scheme since the number of clients viewed is the same at each client. However, the hosts represented by an IMPI Relay have the same `Host_ID`, which might cause a problem if an MPI implementation other than GridMPI is used for intra-cluster communication of private IP address clusters.

The proposed trunking method requires clusters to have multiple front-end nodes. Many private IP address clusters may have only one front-end node, and the trunking method can not be used on such a cluster. Preparing multiple front-end nodes is required to utilize inter-cluster bandwidth which is larger than the interface bandwidth of a front-end node.

The optimal number of trunks depends on several factors, including the number of nodes, the problem size, the inter-cluster bandwidth, the delay, and the communication pattern of the MPI program to be run. For programs which generate huge traffic, trunking has a significant impact on performance. In those cases, the number of trunks should be as large as possible, as long as the aggregated bandwidth of trunks does not exceed the inter-cluster network bandwidth. On the other hand, many programs do not require so much bandwidth when delay is large, and single or a small amount of trunking of relays is sufficient in terms of performance. Therefore, performance with a small number of trunks is comparable to that of 'no relay'.

We have proposed two schemes of trunking multiple relay connections, and implemented scheme#1. However, both

schemes have discrepancies with the IMPI specification. IMPI Relay does not support the dynamic process creation defined in MPI-2, because the current IMPI specification does not support it. The IMPI specification should be extended to support the MPI-2 features, including dynamic process creation, one-sided communication, and I/O.

V. Related Work

Several MPI systems designed for multiple clusters and Grid environments, including MPICH-G2 [7], PACX-MPI [8], StaMPI [9], and MC-MPI [10], have been proposed.

MPICH-G2 does not support private IP address clusters, and assumes full connectivity among all participating nodes. Other implementations support private IP address clusters, but they do not support trunking.

MC-MPI [10] provides a locality-aware connection management mechanism which forwards messages from processes which can not communicate with remote processes directly to a process which can communicate with remote processes. MC-MPI discovers connectivity among processes automatically. On the other hand, IMPI Relay requires a static and manual configuration according to the IMPI specification. However, IMPI Relay can provide higher inter-cluster communication performance by trunking.

MPICH-GP [11] extends the MPICH-G2 functionality to support private IP address clusters. It uses a communication relay scheme which combines a kernel-level NAT service and a user-level proxy. Only incoming messages are relayed by a proxy process, while outgoing messages are forwarded by a NAT mechanism. MPICH-GP introduces GP_GUID into the MPICH-G protocol to guarantee the system-wide uniqueness of ids of nodes inside private IP address clusters. On the other hand, IMPI Relay follows the standardized IMPI protocol, instead of deploying a proprietary protocol. MPICH-GX [12], which is a successor project of MPICH-GP, employs TCP hole-punching to communicate among private IP address clusters. Although the performance is higher than that of a user-level proxy, TCP hole-punching is not always usable depending on the functionality of the NAT boxes used. In [14], S. Guha and P. Francis report an 88% average success rate of TCP connection establishment using TCP hole-punching.

MPI/PRO [13] enables use of private IP address clusters by combining NAT and IMPI Server. However, details of the implementation are not known.

Communicating over IPv6 is another solution, and some MPI libraries as well as GridMPI support message passing over IPv6. However, currently most sites do not allow the provision of IPv6 connectivity. The deployment of IPv6 for the high performance computing community is still a pending issue.

VI. Conclusion

We proposed a high performance relay mechanism for MPI libraries run on multiple private IP address clusters. The proposed trunking method is one in which multiple relay nodes at each cluster simultaneously communicate to improve the available communication bandwidth between clusters. We have shown the proposed method works with the GridMPI, and confirmed the efficiency by experiments using an emulated WAN environment with 10 Gbps bandwidth and a delay that ranged from 0 to 10 msec. We have shown that most of the NAS Parallel benchmarks improved in performance proportional to the number of trunks, since the inter-cluster communication bandwidth increases due to aggregation of relay connections. Especially, using 8 trunks, FT and IS perform 4.4 and 3.4 times faster compared with the single trunk case. While the relay mechanism introduces an approximately 25 μsec one-way latency, the overhead is relatively smaller than the communication latency between clusters connected to a wide area network. The results have indicated that the proposed method is effective and efficient for running MPI programs over high bandwidth-delay product networks.

In this paper, although we have presented a user-level proxy implementation called IMPI Relay, the proposed trunking idea is applicable to other solutions for supporting private IP address clusters. We also addressed the proposition that the IMPI specification should be extended to support a connection trunking feature among clusters, as well as the MPI-2 features, and that is an open issue.

Acknowledgment

Part of this research was supported by a grant from the Ministry of Education, Sports, Culture, Science and Technology (MEXT) of Japan through the NAREGI (National Research Grid Initiative) Project.

References

[1] GridMPI, http://www.gridmpi.org/
[2] R. Takano et al., "Effects of Packet Pacing for MPI Programs in a Grid Environment," IEEE Cluster 2007, September 2007.
[3] YAMPI, http://www.il.is.s.u-tokyo.ac.jp/yampi/
[4] W. L. George et al., "IMPI: Making MPI Interoperable," Journal of Research of the National Institute of Standards and Technology, Vol.105, No.3, May 2000.
[5] GtrcNET-10, http://www.gtrc.aist.go.jp/gnet/
[6] Y. Kodama et al., "GNET-1: Gigabit Ethernet Network Testbed," IEEE Cluster 2004, September 2004.
[7] N. T. Karonis et al., "MPICH-G2: A Grid-Enabled implementation of the Message Passing Interface," Technical Report ANL/MCS-P942-0402, April 2002.
[8] E. Gabriel et al., "Implementing MPI with Optimized Algorithms for Metacomputing," The 3rd MPI Developer's and User's Conference, March 1999.
[9] T. Imamura et al., "An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers," EuroPVM/MPI 2000, LNCS1908, pp.200–207, September 2000.
[10] H. Saito and K. Taura, "Locality-aware Connection Management and Rank Assignment for Wide-area MPI," IEEE CCGrid 2007, May 2007.
[11] K. Park et al., "MPICH-GP: A Private-IP-enabled MPI over Grid Environments," The 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA 2004), December 2004.
[12] MPICH-GX, http://supercom.yonsei.ac.kr/project/mpi/mpi.html.
[13] P. Raman et al., "High Productivity MPI - Grid, Multi-Cluster, and Embedded System Extensions," HPEC 2004, September 2004.
[14] S. Guha and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," Internet Measurement Conference 2005, October 2005.
[15] J. Border et al., "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, June 2001.