Universal Execution of Parallel Processes: Penetrating NATs over the Grid

Insoon Jo[1], Hyuck Han[1], Heon Y. Yeom[1], and Ohkyoung Kwon[2]

[1] School of Computer Science and Engineering,
Seoul National University,
Seoul 151-742, Korea
{ischo,hhyuck,yeom}@dcslab.snu.ac.kr
[2] Supercomputing Center, KISTI, Daejeon, Korea
okkwon@kisti.re.kr

Abstract. Today, clusters are very important computing resources and many computing centers manage their clusters in private networks. But parallel programs may not work in private clusters. Because hosts in private clusters are not globally reachable, hosts behind different private clusters cannot be reached directly in order to communicate. It will certainly be a huge loss of resources if private clusters are excluded from the computing due to this reason. There has been much research on this issue, but most of them concentrate on user-level relaying because it is a general and easily-implementable solution. However, even well-implemented, user-level solutions result in much longer communication latency than kernel-level solutions. This paper adopted a novel kernel-level solution and applied it to MPICH-G2. Our scheme is generally applicable, simple and efficient. The experimental results show that our scheme incurs very little overhead except when small messages are transmitted. That is, it supports a more universal computing environment by including private clusters with remarkably little overhead.

1 Introduction

Grid is a computing paradigm aiming for large-scale resource sharing and clusters are one of its key components. By sharing globally distributed resources such as data, storage, and application, it pursues benefits of both cost and efficiency. Many parallel programming models on top of Grid are proposed to make full use of integrated resources. Currently, Message Passing Interface (MPI) is a *defacto* standard and many studies aim to advance MPI.

It is very typical that clusters assign private addresses to their working nodes and such nodes are unreachable from public networks. Therefore, it is impossible for any two nodes behind different private clusters to directly communicate. It could be a huge loss of resources if private clusters are excluded from Grid due to the reason above. Hence there has been much work on making parallel programs be executed over clusters in private networks [1,2,3]. Most of them

Y. Shi et al. (Eds.): ICCS 2007, Part II, LNCS 4488, pp. 577–584, 2007.

relay messages using a separate application because it could be a more general solution than kernel-level strategy. However, even well-implemented, user-level relaying results in an increase in communication latency since every message destined for cluster nodes in private networks should necessarily pass through a relaying application. This might lead to considerable decline of throughput, specially in case nodes frequently exchange large messages. Although the kernel-level approach is superior over user-level solution in terms of performance, it is generally not used due to its poor portability. Notice that the latter can be configurable at the user-level.

Main objective of our study is to devise a portable kernel-level relaying method and to apply it to MPICH-G2, which is a grid-enabled implementation of the MPI. MPICH-G2 identifies each node by hostname and uses it to make a point-to-point connection between the computing nodes. Hosts in private networks do not have a globally unique hostname. So connection trial to them will fail unless both source and destination hosts belong to the same private cluster. We call our scheme MPICH-GU because it is a more universal version to be runnable over private clusters. We measured communication latency in MPICH-GU and MPICH-G2. We also benchmarked them using NPB (NAS Parallel Benchmarks) suites [4]. The results show that the performance of our scheme is nearly the same as that of MPICH-G2 except small messages. In other words, it supports a more universal Grid environment including private clusters with very little overhead. There is no need to mention that it can remarkably outperform user-level solutions.

The rest of this paper is organized as follows. Section 2 introduces various approaches to traverse private networks. Section 3 details design and implementation of MPICH-GU. Section 4 presents our experimental results. Section 5 concludes this paper.

2 General Concepts

In this section, we briefly peruse generally known schemes for implementing inter-host communication across private networks.

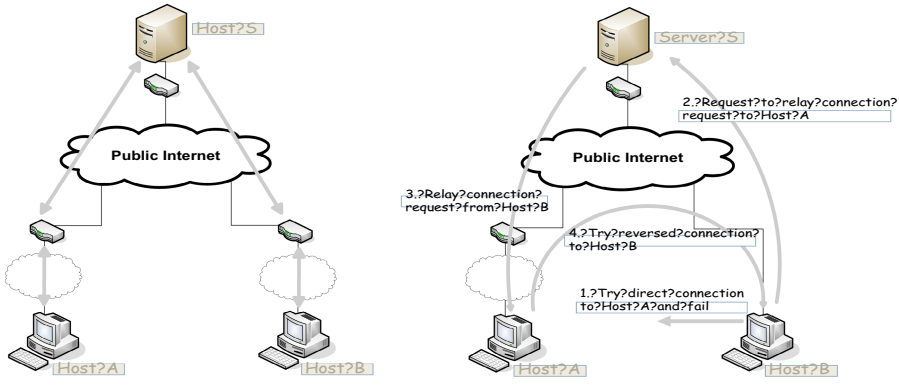
2.1 Problem Domain

In general, each host behind private networks does not have a unique address on public networks. Thus it remains anonymous and inaccessible from the external hosts. Such unreachability does not trouble outgoing-centric applications such as web browser.

However, as P2P programs for game, video conferencing, or file sharing are widely used, it is inevitable to solve this unreachability. P2P applications essentially need inter-host communication, but the incoming connections from the outside hosts toward private networks will not be setup due to their NATs [5]. Most of NATs function as an asymmetric bridge between a private network and public networks. That is, they permit outgoing traffic but discard incoming one

unless it belongs to an existing session initiated from the host within the private networks.

There has been some work on this issue [6,7,8], and the currently known schemes are relaying, connection reversal, and UDP hole punching.



(a) Relaying (b) Connection Reversal

Fig. 1. Generally Known NAT Traversal Schemes

2.2 Relaying

In the relaying scheme, a special host acts as an intermediary between normal hosts. Suppose that host A and B are behind different private networks and both know host S, which must have a public address to be reachable from any host. When A wants to send some message to B, it does not attempt a direct connection because such one will be certainly dropped by the other's NAT.

As shown in Figure 1(a), it just sends messages to S via existing connection between A and S, and S delivers them to B using the already-established connection between S and B. In case B wants to communicate, the reverse process occurs.

Relaying is the most general approach. It can function as long as the relaying host is adequately configured. However, it has major shortcomings. First, each host severely depends on the relaying host, so it can be a potential bottleneck. Secondly, communication latency is increased due to additional overhead of TCP/IP stack traversal and each switch between kernel and user mode. It has been reported by Müller et al. [9] that relaying had a higher latency and also achieved only about half the bandwidth of the kernel-level solutions such as gateway or NAT.

2.3 Connection Reversal

Connection reversal provides a direct connection only when at least one of two hosts is globally reachable. As in Figure 1(b), suppose that there are two hosts

A and B, and a server S. A is behind private networks, B has a public IP address and S can reach both of them. A can establish a direct connection to B. But B cannot because A is not globally reachable. After failing to initialize connection, B uses S to relay connection request to A. On receiving such request, A tries reversed connection to B. Connection reversal is too restrictive to be generally used, because it does not work at all if two hosts reside in the different private networks.

3 Design and Implementation

3.1 Overall Architecture of MPICH-GU

MPICH-GU consists of three kinds of hierarchical process managing modules, which are central manager, cluster manager and local job manager, and working processes as shown in Figure 2. Each manager manages lower-level managers or process and relays control messages between higher-level and lower-level. For example, when user requests a job, central manager distributes it to its cluster managers and individual cluster manager does the same to its local job managers. Then local job manager forks and executes a working process.

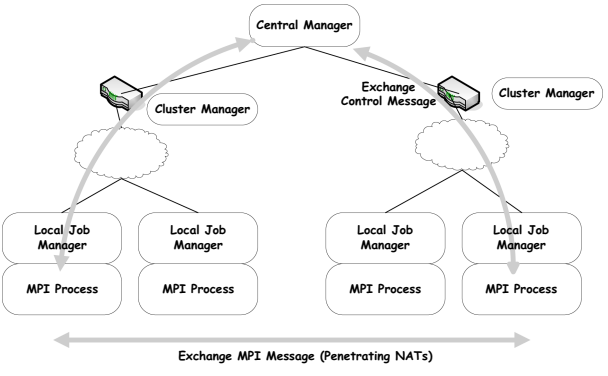


Fig. 2. Overall Architecture of MPICH-GU

The channel establishment of MPICH-GU proceeds as follows: Each MPI process creates listener socket to accept channel opening requests from others, and then transmits its protocol information containing the address of listener. It is sent to central manager through local job manager and cluster manager in turn. After gathering such information of all processes, central manager broadcasts it to every process through cluster managers and local job managers. On receiving this, process constructs *commworldchannels* table, in which *i-th* entry is the channel information of process with rank *i*. This represents real context for individual channel, namely, currently listening address, pointer to established

connection, pointer to sending queue, and etc. When a process has something to send, it attempts connection to destination process using information in this table.

3.2 Penetrating NATs

Our scheme utilizes the common property of NAT. Whenever receiving a message, NAT decides either to drop it or to redirect it to some host in its network based on its table. NAT will certainly drop the message unless it belongs to an existing session initiated from the inner host. If the host wants to receive all the messages sent from the unconnected hosts to oneself, its address should be registered in the NAT table. Therefore, such host would request its NAT to forward incoming messages, that is, to register its address in the table. Then all the messages sent to that address will be redirected to the owner of the address. Notice that any host outside the NAT connects to a host behind the NAT using the address of the NAT.

Our scheme requires a forwarding-enabled NAT as a precondition. Once this condition is satisfied, the following steps should be implemented:

- Every host behind private networks requests forwarding incoming messages, and such request is adequately handled by NAT.
- Each host can get address information of anyone with which it wants to communicate from the hierarchical process management system.
- Each host can connect to any host using information got from the process management system.

We implemented MPICH-GU library based on MPICH version 1.2.6 and the Globus Toolkit version 4.0.1. More details about implementation are as follows. As stated in Section 3.1, at initial stage, each MPI process creates listener socket, then reads the value of environment variable named `FRONT_NAME`. Only for a process behind private network, it represents the hostname of the machine running its NAT. Otherwise, it is null. Therefore, if it has some value, the process should request local job manager to register globally reachable address linked to its own listener socket.

Cluster manager delivers such request to NAT and returns the public address back to requestor. Then each process transmits its protocol information containing two endpoints consisting of the (hostname, TCP port) pair to central manager. For hosts behind private networks, they are private address and public address obtained from the NAT, and for the others, they are the identical public addresses. On receiving this from central manager, each process constructs *commworldchannels* tables in which each entry contains two endpoints of individual process.

When a process has something to send, it tries just one connection according to network topology. It first checks the public endpoint of destination process. If that hostname matches its own `FRONT_NAME`, it means that both belong to the same private network. Hence it attempts establishing a channel using a private endpoint. Otherwise it does using a public endpoint.

3.3 Discussion

Some might be concerned that how many NATs would support forwarding scheme. By the research of Bryan et al. [8], about 82% of the tested NATs support forwarding scheme for UDP, and about 64% support forwarding scheme for TCP. Saikat et al. [10] also showed that TCP forwarding scheme was successful at a average rate of 88% for existing NATs and a 100% for pairs of certain common type of NATs. These results show that running parallel processes over private clusters does not require a special mechanism or device and our scheme can be applied to many parallel systems such as OpenMP [11] and MPJ [12].

Our scheme can simply and efficiently manage NAT's forwarding entries. At initializing phase, each MPI process checks its own network topology. Then if it is needed, it requests forwarding via local job manager. Such request is deliverd to its NAT by the cluster manager and adequately handled. Notice that if the NAT is based on the Linux, well-known commands such as `iptables` can handle the request. The central manager can also detect abortion or completion of the job. If such events happen, the central manager notifies them to cluster managers. And each cluster manager requests its NAT to stop relaying.

4 Experimental Results

We have conducted two kinds of experiments to measure the performance of our scheme. The first is to measure the communication latency and the second is to research how it will affect the performance under real workload. To grasp relative performance, we carried out the same experiments using MPICH-G2 as well. We used 7 machines with 850MHz Pentium III processor and 1GB RAM, which are running Linux 2.4.20. The type of networks we used in all experiments is Gigabit Ethernet. For MPICH-GU experiments, we configured two private networked clusters with 3 nodes. For MPICH-G2 experiments, we used the same machines but assigned them public addresses.

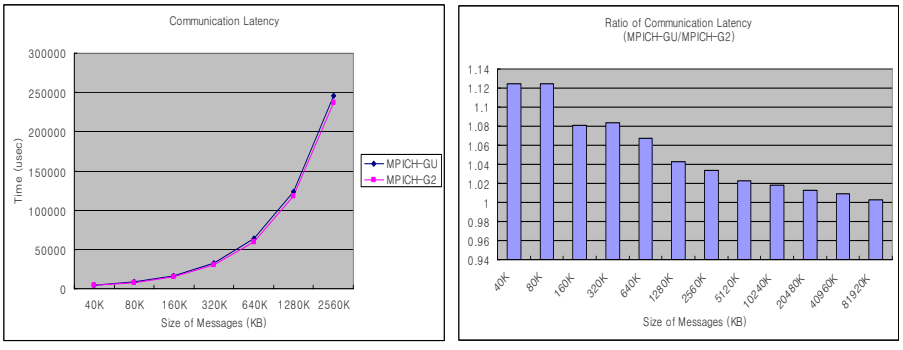
4.1 Communication Performance

To evaluate communication latency, we wrote a simple ping-pong application. It measures a round-trip time for various-sized messages by averaging the time spent to send and receive 20 messages.

As shown in Figure 3(a), the latency of MPICH-GU is very close to that of MPICH-G2. Figure 3(b) represents the round-trip time ratio of MPICH-GU to MPICH-G2. As can be seen from the figure, the bigger the message size, the smaller the latency gap becomes. For example, for messages of size 40KB, MPICH-GU takes 12.5% longer than MPICH-G2, but for messages of size 80MB, the overhead of MPICH-GU is negligible.

4.2 Workload Performance

To evaluate performance of actual workload, we used NPB suites of version 3.2. We executed two of them, which were BT and SP with class A and class B.



(a) Communication Latency (b) Ratio of Communication Latency

Fig. 3. Communication Performance. (a) represents communication latency in running applications using MPICH-GU and MPICH-G2. (b) shows their ratio.

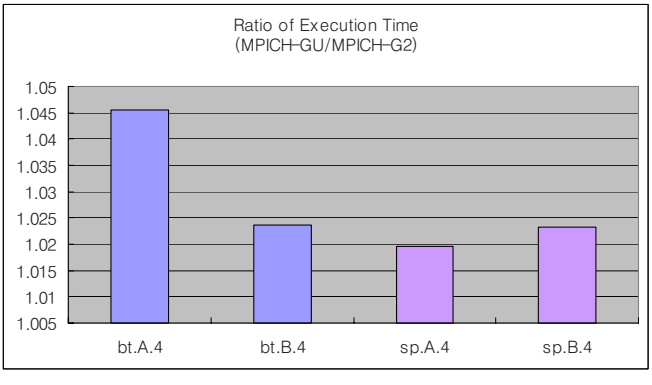


Fig. 4. Ratio of Execution Time (MPICH-GU to MPICH-G2)

Since they exchange relatively large messages, they are sufficient to show how MPICH-GU affects workload performance. Our experimental results are shown in Figure 4. As expected, our scheme incurs very little overhead compared with MPICH-G2 - at most 4.6%.

5 Conclusion

In this paper, we have presented MPICH-GU which enables a more universal Grid environment including private clusters. Penetrating scheme which MPICH-GU exploited is not only generally applicable but also simple kernel-level strategy. Implementing kernel-level scheme, MPICH-GU can deliver nearly as

identical performance as MPICH-G2 except when small messages are transmitted. Our experimental results show that it supports private clusters with very little overhead. It is a remarkable result compared with the performance of previously implemented user-level strategies.

References

1. K. Park, S. Park, O. Kwon and H. Park. MPICH-GP: A Private-IP-enabled MPI over Grid Environments. ISPA, 2004.
2. E. Gabriel, M. Resch, T. Beisel and R. Keller. Distributed computing in a heterogenous computing environment. EuroPVMMPI, 1998.
3. Jack Dongarra, Graham E. Fagg, Al Geist, James Arthur Kohl, Philip M. Papadopoulos, Stephen L. Scott, Vaidy S. Sunderam and M. Magliardi. HARNESS: Heterogeneous Adaptable Reconfigurable NEtworked SystemS. HPDC, 1998.
4. Michael Frumkin, Haoqiang Jin and Jerry Yan. Implementation of NAS Parallel Benchmarks in High Performance Fortran. NAS Techinical Report NAS-98-009, 1998.
5. P. Srisuresh and M. Holdrege. IP network address translator (NAT) terminology and considerations. RFC 2663, 1999.
6. B. Ford, P. Srisuresh and D. Kegel. Peer-to-peer communication across middleboxes. Internet Draft draftford- midcom-p2p-01, Internet Engineering Task Force, Work in progress, 2003.
7. Rosenberg, J. Traversal Using Relay NAT (TURN). draftrosenberg- midcom-turn-04, 2004.
8. Bryan Ford, Pyda Srisuresh and Dan Kegel. Peer-to-Peer Communication Across Network Address Translators. USENIX 2005, pp.179-192, 2005.
9. M. Muller, M. Hess and E. Gabriel. Grid enabled MPI solutions for Clusters. CCGRID'03, pp.18-24, 2003.
10. S. Guha and P. Francis. Characterization and Measurement of TCP Traversal through NATs and Firewalls. IMC 2005, 2005.
11. OpenMP Architecture Review Board. OpenMP, http://www.openmp.org.
12. B. Carpenter, V. Getov, G. Judd, T. Skjellum and G. Fox. MPJ: MPI-like Message Passing for Java. Concurrency: Practice and Experience, Volume 12, Number 11, September 2000.