Architecture of a High-Speed MPI_Bcast Leveraging Software-Defined Network

Khureltulga Dashdavaa[1], Susumu Date[1], Hiroaki Yamanaka[2], Eiji Kawai[2], Yasuhiro Watashiba[1], Kohei Ichikawa[3], Hirotake Abe[4], and Shinji Shimojo[1]

[1] Osaka University
[2] National Institute of Information and Communication Technology
[3] Nara Institute of Science and Technology
[4] University of Tsukuba

Abstract. Collective communication is of great importance in MPI because the execution time of an MPI program is affected by the communication performance it can gain. Particularly these days, when a cluster system composed of multiple computing nodes has become dominant as a large-scale computing system, the execution time of collective communication affects the total execution time of the MPI program. However, in many implementations of MPI, collective communication is developed to make use of unicast-based communication in a repeated and combined way, which may result in inefficient communication. In this paper, we explore the use of a Software-Defined Network, which was originally expected to help network administrators operate networks through central control in a software-programming manner, to accelerate `MPI_Bcast`, a basic collective communication used in MPI. The evaluation in this paper indicates that our prototyped `SDN_MPI_Bcast` is superior to `MPI_Bcast` in OpenMPI in communication performance. Also, the evaluation implies that `SDN_MPI_Bcast` is feasible.

1 Introduction

Message Passing Interface (MPI) [1,2] has been a *de facto* standard API in parallel and distributed computing for around two decades. MPI provides programmers with APIs for point-to-point and collective communication, by taking into consideration possible communication patterns occurring in parallel and distributed computing. Conventional cost-sensitive clusters are usually equipped and configured with gigabit Ethernet switches and famous MPI software, such as OpenMPI [3]. With those combinations, collective communications are implemented as a set of multiple point-to-point communications, which might incur performance penalties caused from packet congestion [4,5].

In this research, we explore a novel method for implementing MPI collective communication leveraging the controllability of interconnect network, brought by Software-Defined Network (SDN) [6,7]. SDN is a new concept of network architecture that enables dynamic control and management in a software-programming manner. SDN is expected to facilitate the daily operations of network administrators. As a proof of concept, we propose an SDN-enabled version of `MPI_Bcast`, the

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 885–894, 2014.

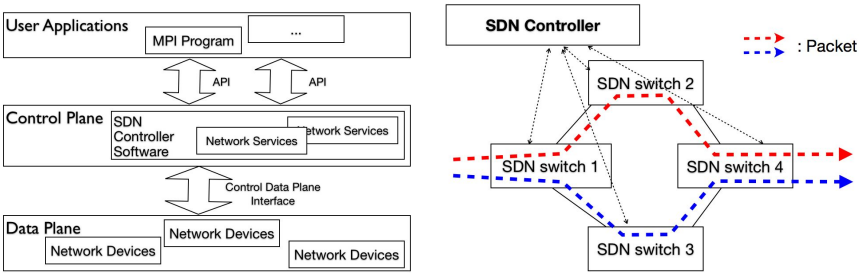
most basic collective communication, by overturning the conventional assumption that a network is not controllable during computation.

The remaining part of this paper is as follows. Section 2 briefly introduces SDN and then describes the basic idea and approach to MPI Broadcast leveraging SDN. Subsequently, in Section 3, we present the blueprint architecture of MPI Broadcast utilizing the functionalities offered by SDN and then we detail how it is prototyped. In Section 4, an evaluation is conducted to verify the feasibility of our approach of using SDN for MPI Broadcast. Section 5 describes related works. Section 6 concludes the paper.

2 Approach

2.1 Software-Defined Network

Software-Defined Network (SDN) is a new concept of network architecture that separates traditional networking functions into a network control plane (SDN controller) and a data plane (network devices such as switches) [6,7]. Figure 1a shows the layered architecture of SDN. In SDN, a program deployed on the Control Plane works as a SDN controller and is in charge of delivering a set of rules instructing how to deal with network flows to network devices through a Control Data Plane Interface. On the other hand, each of the network devices in the Data Plane is responsible for dealing with network flows according to the rule set delivered by the SDN controller. A promising feature of SDN is that it allows user applications such as MPI programs to interact with SDN controllers.



(a) Layered Architecture **(b)** Controlling Example

Fig. 1. Architecture of Software-Defined Network

From the administrative perspective, SDN is operated as follows. In the case that a network administrator wants to send a specific series of packets to SDN switch 4 from SDN switch 1 via SDN switch 2, all the administrator has to do in SDN is to write and deploy a program as a SDN controller, so that it sends a set of rules describing how to deal with the corresponding series of packets to SDN switches 1, 2 and 4 (Fig. 1b). This behavior is explained from the following SDN mechanism. Under the SDN, whenever the SDN switch receives any packet with

which it does not know how to deal, it sends a query in terms of how to deal the packet to the SDN controller. After receiving the packet, the SDN controller informs the SDN switch of a set of rules. In this example case, once the SDN controller installs rules to SDN switch 1, 2, and 4, the specific series of packets is set to flow along the red-arrow path in the figure. Also, in the case that the network administrator wants to set a network flow from SDN switch 1 to SDN switch 4 via SDN switch 3 for another specific series of packets, all the network administrator has to do is to modify the program as a SDN controller.

Currently, OpenFlow [8,9] is a *de facto* standard implementation of SDN. It provides a suite of communication interfaces between the Control Plane and Data Plane [6]. In this feasibility study, OpenFlow is integrated into `MPI_Bcast`.

2.2 Basic Idea Behind the Proposed Solution

The programmability of SDN allows the SDN controller to install a set of rules that duplicate and send incoming packets to specified multiple ports on switches. We approach the SDN-version of `MPI_Bcast`, based on the simple idea that execution time of `MPI_Bcast` can be reduced by duplicating packets on the switch along the data delivery route from a process to remained processes. Specifically, in the case that a process attempts to broadcast data to remained processes, duplicating packets from an incoming port to multiple outgoing ports based on a broadcast tree could lead to the reduction of `MPI_Bcast` execution time. This duplication method is considered to be possible because data on the fly to all remained processes from a source process are identical in MPI Broadcast except for packet headers (Ethernet and IP headers).

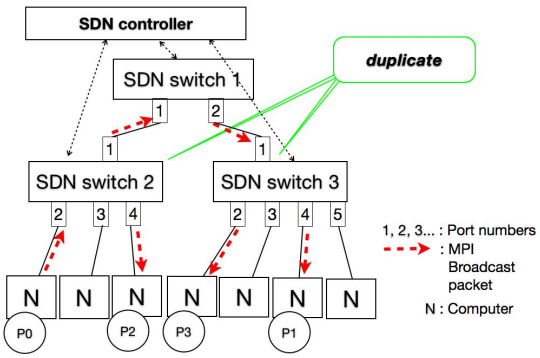


Fig. 2. Basic idea towards SDN MPI Broadcast

Figure 2 illustrates the basic idea regarding how to realize an `MPI_Bcast` using SDN. Assume that 4 processes of an MPI program are running on a computing environment composed of 7 computers, which are connected to a SDN. Also, the figure assumes that Process 0 (P0) is about to broadcast data to P1, P2, and P3. Under this assumption, our proposed method is expected to work as follows.

First, the packet sent by P0 is first delivered to port 2 on SDN switch 2. Then, based on a packet matching rule provided by the SDN controller, SDN switch 2 duplicates and sends it out to port 1 and 4 (respectively for SDN switch 1 and P2). Next, SDN switch 1 forwards data to port 2 (for SDN switch 3) based on the packet matching rule provided by the SDN controller. Finally, SDN switch 3 receives data through port 1 from SDN switch 1. Then SDN switch 3 duplicates data and sends it out to port 2 and 4 (respectively for P3 and P4).

3 Design and Implementation of SDN-Based MPI Broadcast

3.1 Design of SDN-Based MPI_Bcast

In order to realize MPI Broadcast using the proposed method explained in Section 2, MPI processes need to interact with a SDN controller for at least two kinds of information before computation. The following discussion assumes only one process runs on a computing node for simplification.

1. A list of IP addresses of computers, which are used in the MPI Broadcast. The SDN controller needs to understand which computers are used in the MPI Broadcast.
2. A special ID for identifying the target MPI Broadcast packets. The identification of MPI Broadcast packets is required because SDN switches need to duplicate only the target MPI Broadcast packets selectively.

To share the first piece of information between the MPI processes and the SDN controller, a master process of the MPI program is designed to send the list of IP addresses of computers that involve computation to the SDN controller. After obtaining the list, the SDN controller builds a broadcast tree based on it and then creates duplication rules of SDN switches.

To share the second piece of information, we have decided to use a special destination MAC address as an ID. For this purpose, the SDN controller is designed to generate a special MAC address and then send this address to a master process of the MPI program. Through this design, when a process of the MPI program broadcasts data, the process sets the special MAC address into a series of MPI broadcast packets before broadcasting the data.

Figure 3 shows how an MPI program should interact with the SDN controller to realize the SDN-based MPI Broadcast method in detail. The interaction between them takes place as follows.

1. The SDN controller obtains the topology of a network composed of switches and computers, where the computation of the MPI program is performed. Subsequently, the SDN controller waits for the MPI program's connection.
2. All processes of the MPI program initialize the MPI execution environment.
3. After the initialization, a master process of the MPI program sends the IP address list of computers participating in the computation. Then, the MPI program waits for a reply from the SDN controller.

4. After the SDN controller receives the list, it builds a broadcast tree based on the list and the topology.
5. After the broadcast tree is built, the SDN controller generates an ID (special MAC address) for a series of the target of MPI Broadcast packets. Importantly, the ID is assigned to each `MPI_Bcast` invoked in the computation.
6. The SDN controller makes a set of duplication rules, which instruct packets of the target `MPI_Bcast` to flow from a source process to the remained processes on a broadcast tree. The rules are then deployed onto SDN switches.
7. The SDN controller sends the ID for the MPI Broadcast to the MPI program.
8. After a master process receives the ID from the SDN controller, it sends the ID to processes so that the remained processes use it in receiving the target broadcast packets.
9. After the eight steps above, processes can invoke the SDN-based `MPI_Bcast`.

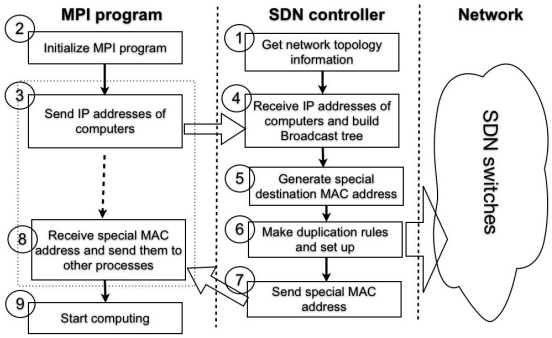


Fig. 3. Interaction model in SDN-based MPI Broadcast

3.2 Implementation of SDN MPI Broadcast

Based on the design in Section 3.1, we have developed the SDN-based MPI Broadcast utilizing OpenMPI version 1.6.3. The prototyped `MPI_Bcast` is composed of two modules; the SDN controller module and the SDN MPI Broadcast library.

SDN controller module. For implementation of the SDN controller, POX [10], which is a development framework for the OpenFlow controller developed in Python version 2.7, has been adopted. POX supports the OpenFlow version 1.0 specification at the time of writing this paper. The SDN controller module has been developed to offer the functions shown in Fig. 3. To understand the network topology, we plan to use an LLDP-based topology understanding mechanism, which our research team has developed in another project [11]. In the current stage of this research, however, the information on network topology is set manually because learning the topology is beyond this paper's scope. For the function of building the broadcast tree, the SDN controller was developed to simply use the minimum number of hops (using the Floyd-Warshall algorithm).

In the near future, however, a more sophisticated method of building the tree is necessary because we have currently assumed for simplification that only a set of MPI programs are executed on a cluster system and thus any background traffic does not exist.

SDN MPI Broadcast Library. The MPI program needs to call `MPI_Init` for initialization and `MPI_Finalize` to finalize the MPI execution environment. The MPI program performs a computation job between these APIs. Taking this execution mechanism into consideration, we have developed a new SDN MPI Broadcast Library called `sdn_mpi_bcast`. This library contains two APIs; `SDN_MPI_Init` and `SDN_MPI_Bcast`.

SDN_MPI_Init: This API is implemented to offer the functions shown in the dotted part in Fig. 3. This API provides MPI programs with the functions of exchanging the IP address list of computers participating in the computation as well as the ID for identifying the SDN MPI Broadcast packets between the MPI program and the SDN controller. When this API is called, a process receiving the ID sends it to the remained processes.

SDN_MPI_Bcast: This API broadcasts data by the proposed method. For this purpose, `SDN_MPI_Bcast` sets up a destination MAC address in broadcasting packets. We have developed a library, called `my_raw_packet`, which can send and receive raw packets by setting packet headers manually. This API has been developed on `my_raw_packet` library.

Importantly, the MPI program with `SDN_MPI_Bcast` is written the same way as OpenMPI. In detail, to use the proposed `SDN_MPI_Bcast`, the developer just needs to replace `MPI_Bcast` with `SDN_MPI_Bcast`.

4 Evaluation

4.1 Experimental Environment

An experiment has been conducted to compare the execution time of `MPI_Bcast` and `SDN_MPI_Bcast` implemented in OpenMPI. Also, we have measured the time required for `SDN_MPI_Init` to learn the setup cost required for our proposed method. Figure 4 illustrates the experimental environment. The experiment was performed on a cluster system which consists of 27 physical computing nodes and 3 OpenFlow switches. OpenFlow switches are 1 Gb/s bandwidth Programmable-Flow PF5240 Switch, and each node has Intel(R) Xeon(R) 2.00 GHz processors. For this experiment, two of three OpenFlow switches were configured to function as two logical OpenFlow switches.

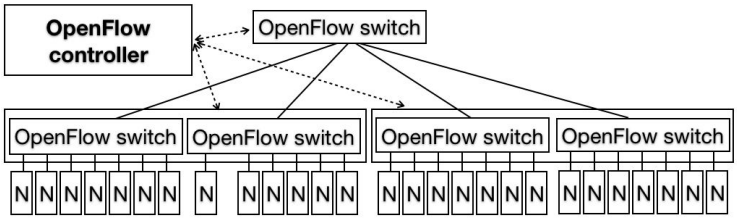


Fig. 4. Experimental environment

4.2 Experimental Method

In this experiment, execution time of our `SDN_MPI_Bcast` and `MPI_Bcast` in OpenMPI was measured and compared through the use of a simple MPI program that just broadcasts an integer array. In this measurement, we have adopted the average execution time spent for broadcast in each of the MPI processes. Furthermore, to obtain execution time, we performed the program 20 times and adopted a average execution time of 20 measurements as the measurement result. A fraction of the MPI program is shown in Fig. 5. `MPI_Wtime`, which returns time in seconds, since an arbitrary time in the past [2] was used for measuring time. The execution time of `SDN_MPI_Bcast` and `MPI_Bcast` in OpenMPI was measured in the following two ways.

```
Initialize MPI execution environment
...
SDN_MPI_Init ( ... ); // For connection with the SDN controller
start = MPI_Wtime ();
MPI_Bcast ( ... ) or SDN_MPI_Bcast ( ... );
end = MPI_Wtime ( ... );
diff_time_on_each_process[] <- end - start // in seconds
average_difference_time = average(diff_time_on_each_process[])
...
Finalize MPI execution environment
```

Fig. 5. A fraction of MPI program used in the experiment

(a) Measure the execution time of MPI Broadcast by running 27 processes for the MPI program, and by changing the data size of the integer array to be broadcast.
(b) Measure the execution time of MPI Broadcast by running the MPI program whose integer array is 4000 in size, and by changing the number of processes.

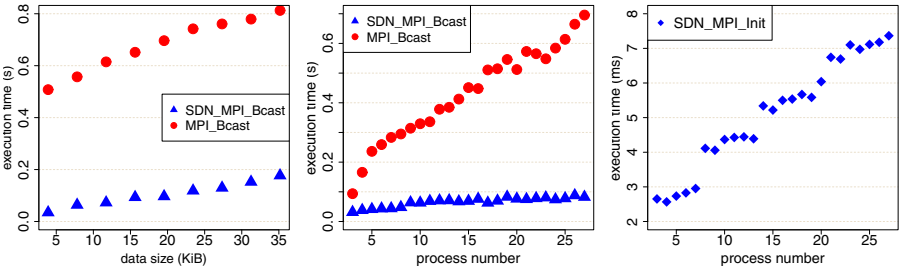
Furthermore, to investigate the overhead incurred for the initialization process (the dotted part in Fig. 3), we have measured the execution time of `SDN_MPI_Init`. The average execution time of `SDN_MPI_Init` of 20 measurements has been taken.

4.3 Experimental Result

Figure 6a plots two series of the obtained measurement time in both `MPI_Bcast` in OpenMPI and our `SDN_MPI_Bcast` in case (a). The vertical axis is the measured execution time, and the horizontal axis is the size of data to broadcast. The graph indicates that our `SDN_MPI_Bcast` has better performance than `MPI_Bcast` in OpenMPI in terms of execution time.

Figure 6b shows the measurement result of case (b). The vertical axis is the measured execution time, and the horizontal axis is the number of processes used in the experiment. The graph indicates that the proposed `SDN_MPI_Bcast` is superior to the `MPI_Bcast` and also that `SDN_MPI_Bcast` scales well against process number.

Figure 6c shows the measurement result of `SDN_MPI_Init`. The vertical axis is the measured execution time and the horizontal axis is the number of processes used in the experiment.



(a) Data size change **(b)** Process number change **(c)** SDN_MPI_Init

Fig. 6. Execution time

4.4 Insights

In case (a), as broadcasting data size increased, `SDN_MPI_Bcast` had a better performance than `MPI_Bcast`. Also, the execution time of `SDN_MPI_Bcast` did not increase as quickly as `MPI_Bcast` when the size of broadcasting data changed. In case (b), the graph shows that `SDN_MPI_Bcast` becomes faster than `MPI_Bcast` as process number grows. The reason can be explained from the fact that `MPI_Bcast` in OpenMPI uses point-to-point communication repeatedly for a call of `MPI_Bcast`, while `SDN_MPI_Bcast` delivers data from source to remained processes along a broadcast tree optimized on the network topology.

In addition, the execution time of `SDN_MPI_Init` was less than 10 milliseconds when process and switch numbers were 27 and 5, respectively. It could take more time with both the number of processes and switches increased. However, this API has to be called only once in the beginning of the program. Therefore, the execution time of `SDN_MPI_Init` is not considered a big problem for jobs which take a long time.

5 Related Works

To date, much research related to the acceleration of the MPI Broadcast has been reported. As to MPI Broadcast, several research groups have worked on acceleration of `MPI_Bcast` on InfiniBand, which is a major high-performance interconnect technology available today. Representative examples of such research include [12] and [13]. In these works, to realize `MPI_Bcast` on the top of hardware multicast supported by InfiniBand, the authors have tackled common issues such as unreliability of message delivery, large message handling, and in-order delivery. The experience and expertise reported in these works can be combined with our research from now on. Our research differs from previous research in terms of target interconnect and research motivation. We are motivated to challenge how HPC technology can be changed by regarding the interconnect as a dynamically controllable network resource.

6 Conclusion and Future Work

In this paper, we have explored the feasibility of a high-speed `MPI_Bcast`, or `SDN_MPI_Bcast`, that leverages software programmability brought by SDN for efficient data delivery from a source process to the remained processes of the MPI program. Evaluation shown in this paper implies that `SDN_MPI_Bcast` is superior to `MPI_Bcast` by measuring the execution times of both regarding data size and the number of processes changed. From the result, we believe the use of SDN might be a possible and feasible solution for reducing the execution time of `MPI_Bcast`. At the same time, through this preliminary study, we now recognize many issues to be tackled to realize a practical `SDN_MPI_Bcast`. As described in Section 3.1, we have assumed for simplification that only one process runs on a computing node. However, multiple MPI processes, each of which belongs to a different MPI job, run on a computing node in the usual cases. Although our proposed method allows us to distinguish network flows derived from a MPI job from others, the separation of MPI job groups on a contended environment remains an issue. Finally, how to deal with packet loss should also be considered.

Acknowledgments. This research was partly supported by collaborative research of the National Institute of Information and Communication Technology and Osaka University (Research on high functional network platform technology for large-scale distributed computing).

References

1. Gropp, W., Graham, R., Moody, A., Hoefler, T., Treumann, R., Träff, J.L., Bosilca, G., Solt, D., de Supinski, B.R., Thakur, R., Squyres, J.M., Rabenseifner, R., Supalov, A.: MPI: A Message-Passing Interface Standard Version 2.2. Technical report, Message Passing Interface Forum (September 2009)

2. Gropp, W., Lusk, E., Skjellum, A.: Using MPI, Portable Parallel Programming with the Message-Passing Interface, 2nd edn. The MIT Press (1999)
3. Graham, R.L., Woodall, T.S., Squyres, J.M.: Open MPI: A Flexible High Performance MPI. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 228–239. Springer, Heidelberg (2006)
4. Rabenseifner, R.: Automatic MPI Counter Profiling of All Users: First Results on a CRAY T3E 900-512. In: Proceedings of the 3rd Message Passing Interface Developer's and User's Conference (MPIDC 1999), pp. 77–85 (March 1999)
5. Rabenseifner, R.: Optimization of Collective Reduction Operations. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3036, pp. 1–9. Springer, Heidelberg (2004)
6. Open Network Foundation: Software-Defined Network, The New Norm for Networks, White Paper (April 2012)
7. Nascimento, M.R., Rothenberg, C.E., Salvador, M.R., Corrêa, C.N.A., de Lucena, S.C., Magalhães, M.F.: Virtual routers as a service: the RouteFlow approach leveraging Software-Defined Networks. In: Proceedings of the 6th International Conference on Future Internet Technologies (CFI 2011), pp. 34–37 (June 2011)
8. Kotani, D., Suzuki, K., Shimonishi, H.: A Design and Implementation of OpenFlow Controller Handling IP Multicast with Fast Tree Switching. In: Proceedings of the 12th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2012), pp. 60–67 (July 2012)
9. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: Enabling Innovation in Campus Networks. SIGCOMM Computer Communication Review 38(2), 69–74 (2008)
10. POX Wiki, `https://openflow.stanford.edu/display/ONL/POX+Wiki`
11. Furuichi, T., Date, S., Yamanaka, H., Ichikawa, K., Abe, H., Takemura, H., Kawai, E.: A prototype of network failure avoidance functionality for SAGE using OpenFlow. In: Proceedings of 36th IEEE International Conference on Computer Software and Applications Workshops (COMPSACW 2012), pp. 88–93 (July 2012)
12. Liu, J., Mamidala, A.R., Pand, D.K.: Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. In: Proceedings of 18th International Parallel and Distributed Processing Symposium 2004 (IPDPS 2004), pp. 26–30 (April 2004)
13. Hoefler, T., Siebert, C., Rehm, W.: A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast. In: Proceedings of the 21st IEEE International on Parallel and Distributed Processing Symposium (IPDPS 2007 Workshop), pp. 1–8 (March 2007)