IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 16, 2020, accepted October 13, 2020, date of publication November 2, 2020,
date of current version November 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035158

Dynamic Traffic Control of Staging Traffic on the Interconnect of the HPC Cluster System

ARATA ENDO[1], (Member, IEEE), HIROKI OHTSUJI[2], ERIKA HAYASHI[2], EIJI YOSHIDA[2], CHUNGHAN LEE[3], SUSUMU DATE[1], (Member, IEEE), AND SHINJI SHIMOJO[1], (Member, IEEE)

[1]Cybermedia Center, Osaka University, Osaka 567-0047, Japan
[2]Fujitsu Laboratories Ltd., Kawasaki 211-8588, Japan
[3]Toyota Motor Corporation, Tokyo 100-0004, Japan

Corresponding author: Arata Endo (endou.arata@ais.cmc.osaka-u.ac.jp)

This work was supported by the Japan Society for the Promotion of Science (JSPS) Grants-in-Aid for Scientific Research (KAKENHI) Number JP16H02802, JP17KT0083, and JP17K00168.

ABSTRACT High-performance computing (HPC) cluster systems sometimes adopt a two-layered file system composed of local and global file systems to achieve both capacity and performance in storage. In such a cluster system, the input data of an application needs to be staged from the global storage into the local storage, and the output data needs to be staged from the local storage out to the global storage. This staging operation must be efficiently and quickly performed to gain higher job throughput because an inefficient staging operation prevents waiting job requests from being executed. In particular, in the case of the cluster system with the oversubscribed interconnect shared by the storage and the computing nodes, the inter-node communication and this staging operation traffic collides, which may degrade the job throughput. In this research, we focus on the traffic collision of the inter-node communication and the staging traffic to improve job throughput, targeting the cluster system with the oversubscribed interconnect where these two types of traffic flow. In other words, whether the dynamic control of the traffic flow derived from the staging operation leads to the improvement in the job throughput or not is investigated. For the investigation, we present a traffic collision avoidance method to dynamically configure a set of data paths for each type of the traffic only while the staging operation is conducted. The evaluation in this article shows that the proposed method avoids a traffic collision and accelerates the staging operation by 22.0% on our cluster system. Also, this evaluation indicates the overhead of the application incurred by the proposed method is negligible. Furthermore, 8.7% of the job execution time is reduced by the proposed method.

INDEX TERMS Dynamic traffic control, high-performance computing, interconnect, software-defined networking.

I. INTRODUCTION

Recent scientific research increasingly necessitates high-performance computing (HPC). In fact, the advancement of measurement technology increases the size of scientific data obtained with scientific measurement devices. For example, radio telescopes coordinated in the Event Horizon Telescope as a virtual telescope generate 2PB of observation data every night [1]. As a result, scientists strongly require high performance to perform their own research with high efficiency. In addition, today's increasing expectations and concerns with artificial intelligence (AI) and machine learning (ML) have furthered the computing needs of scientists and researchers. In this way, HPC systems are now taking on greater importance in the advancement of scientific research.

Today's HPC systems are mostly built as cluster systems composed of multiple computing nodes, each of which is connected on a low latency and high-performance interconnect [2]. Furthermore, HPC cluster systems such as the K-computer [3] adopt a two-layered file system composed of a local file system on each computing node and a global file system shared by multiple computing nodes to achieve both capacity and performance in storage. To obtain the higher performance of applications on such cluster systems,

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Nitin.

the communication performance on the interconnect is as important as the computing performance on the computing nodes. In particular, in the case of the cluster system with an oversubscribed interconnect shared by the storage and the computing nodes, the communication performance on the interconnect becomes a more important factor that affects the total performance of the applications. For example, the inter-node communication traffic and the traffic derived from the staging operation, or the data movement between the local and global file systems collide, which may result in the degradation of job throughput.

In this research, we focus on the traffic collision of the inter-node communication traffic and the staging traffic to improve job throughput, targeting the cluster system with an oversubscribed interconnect where these two types of traffic flows. In other words, whether the dynamic control of the traffic flow derived from the staging operation leads to improvement in job throughput or not is investigated. For the investigation, we present a traffic collision avoidance method to dynamically configure a set of data paths for each type of the traffic only while the staging operation is conducted.

The structure of this article is as follows: In Section II, we show the HPC cluster systems targeted in this research and explain the possibility of a traffic collision between two types of traffic. Also, we present our approach to solve the traffic collision problem in this article. In Section III, we verify that a traffic collision causes the degradation of job throughput through the preliminary investigation. In Section IV, we present the traffic collision avoidance method. Section V shows the evaluation experiment for our investigation on an actual HPC cluster system. In Section VI, we review related works. Finally, we conclude this article in Section VII.

II. PROBLEM CLARIFICATION AND APPROACH

In this section, we first clarify the HPC cluster system targeted in this research. Second, the traffic collision between two types of the traffic on the HPC cluster system is described. Finally, we briefly describe our approach to solve the traffic collision problem in this research.

A. HPC CLUSTER SYSTEMS TO BE TARGETED

In this research, the HPC cluster systems with the oversubscribed interconnect shared by computing nodes and storages are targeted. An example of such cluster systems is shown in Fig. 1. In the target cluster systems, we assume that a two-layered file system is adopted. The two-layered file system is composed of a parallel file system as a global file system accessible from every computing node and a local file system accessible on a computing node. It is also assumed that a parallel file system such as Lustre [4] and Gluster [5] is adopted for the global file system and a file system is constructed on a high performance storage such as SSD on a computing node as a local file system. As to the interconnect, we assume an oversubscribed fat-tree where the bisection bandwidth is smaller than half of injection bandwidth.

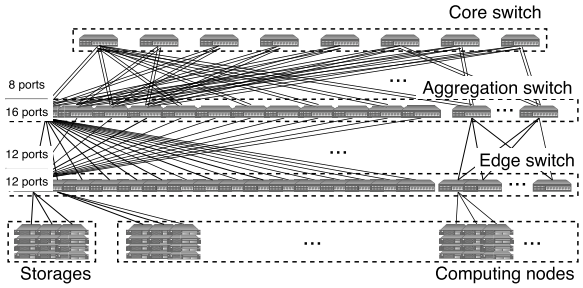


FIGURE 1. The HPC cluster system targeted in this research.

A fat-tree [6] is a major topology adopted for the interconnect of HPC cluster systems. In the fat-tree interconnect, the top-level switch is called the core switch. On the other hand, the lowermost switch is called the edge switch, and the remains are called the aggregation switch. In general, the fat-tree interconnect is classified into two types in terms of available bandwidth: full-bisection and oversubscription. In a full-bisection fat-tree, the bisection bandwidth is half of the injection bandwidth in the cluster system, meaning that half of computing nodes can communicate with the other half without network contention. On the other hand, in the oversubscribed fat-tree, the bisection bandwidth is smaller than half of the injection bandwidth. In today's HPC cluster systems, the latter fat-tree interconnect is used more frequently than the former although the former can provide higher communication performance on the interconnect. For example, AI Bridging Cloud Infrastructure (ABCI) at the National Institute of Advanced Industrial Science and Technology (AIST) adopts an oversubscribed fat-tree [7]. A reason for this can be explained from the fact that the latter costs more than the former, especially in the case of the larger HPC cluster systems.

B. NETWORK TRAFFIC COLLISION

Under target cluster systems like the one shown in Fig. 1, to achieve higher I/O performance from the application running on computing nodes, the input data of an application is staged from the global file system into the local file system before computation (stage-in) and the output data is staged from the local file system into the global file system after the computation (stage-out). The staging traffic, which is the traffic derived from this staging operation, flows on the interconnect of the HPC cluster systems. On the other hand, there are usually many parallel jobs running on HPC cluster systems. Each job is composed of multiple parallel processes that exchange data and messages with each other. Therefore, the inter-node communication traffic exists on the interconnect of the HPC cluster system. In other words, the interconnect is shared by these two types of the traffic. In the case of a full-bisection fat-tree interconnect, these two types of traffic flow without blocking each other. However, in the case of the oversubscribed fat-tree topology adopted in the modern HPC cluster systems, these two types of traffic may block each other due to the bottleneck of the bandwidth on

the interconnect. As a result, the staging operation may take a longer time and the inter-node communication also may take a longer time, which causes the degradation of the job throughput. In this research, we focus on this traffic collision to improve the job throughput.

C. APPROACH

If performance degradation by traffic collision happens, avoiding the traffic collision may lead to the improvement of the job throughput. As described in Section II-B, this staging operation is executed before and after job execution. This means that the longer stage-in results in the delay of job execution and also that the longer stage-out prevents other job to be executed. Therefore, this staging operation must be quickly finished. Likewise, the inter-node communication among computing nodes also must be quickly and efficiently performed for higher job throughput. The longer inter-node communication causes the increase in total job execution time.

For this reason, it becomes important to control these two types of traffic on the interconnect so that these two types of traffic do not interfere with each other. For example, it might be effective to configure a dedicated path on the interconnect to the staging traffic only when the staging operation happens. Based on the above idea, we take the strategy of applying a dynamic traffic control method for the two types of traffic on the interconnect in response to computation context [8].

III. PRELIMINARY INVESTIGATION

A. OVERVIEW OF THE PRELIMINARY INVESTIGATION

As a preliminary investigation, how the two types of traffic flow on the interconnect and how the job throughput is affected was observed. For this observation, we built a HPC cluster system with the oversubscribed fat-tree interconnect and then performed a job submission experiment on the cluster system to reproduce the situation that the two types of the traffic co-exist on the interconnect where the representative static routing algorithm often used in the fat-tree interconnect is applied. To investigate how the two types of traffic flow, we look into the bandwidth consumed by each type of the traffic in the core switches. To investigate how the job throughput is affected due to the traffic collision, we measure the total execution time of jobs when the traffic collision is configured to occur and the execution time when a traffic collision is configured not to occur.

B. EXPERIMENTAL ENVIRONMENT

Fig. 2 shows the HPC cluster system which we built for this preliminary investigation. The HPC cluster system is composed of six computing nodes, one storage node, four switches, one management node and four monitoring nodes (*mnt1* to *mnt4*). The specification of the nodes used for this preliminary investigation is shown in Table 1. The information on switches used for this investigation is summarized in Table 2. The storage node stores data to be moved in staging operations as the global file system of this HPC cluster system. The management node manages the computing nodes with Slurm [9], which is a job scheduler widely used in the modern HPC cluster systems.

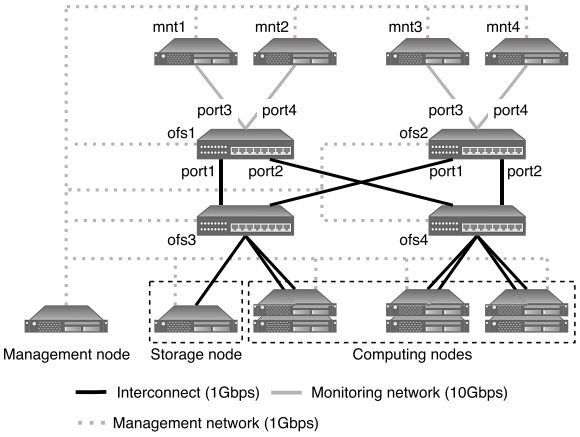


FIGURE 2. The experimental environment.

TABLE 1. The spec of each node.

OS	Fedora release 27
CPU	Intel® Xeon® E5520 (2.27 GHz)

TABLE 2. The spec of each OpenFlow switch.

Switch name	Product name
ofs1, ofs2	Nexus9372PX
ofs3	UNIVERGE PF5240
ofs4	Arista 7050T-36

Two core switches (*ofs1*, *ofs2*) and two edge switches (*ofs3*, *ofs4*) forms a two-level oversubscribed fat-tree (2:3 blocking on ofs3 and ofs4) interconnect, on which the computing nodes and the storage node are connected. Equal-cost multipath (ECMP) [10], which is a static routing algorithm widely used for fat-tree interconnects, was adopted. It selects one of the candidate shortest paths searched by a routing protocol such as Open Shortest Path First (OSPF) for a pair of the nodes. In this investigation, ECMP-based traffic control was implemented with the Ryu [11] framework, which is a software framework to develop an OpenFlow controller. In addition, all of switches, the computing nodes, the storage node and the monitoring nodes were connected on a management network.

OpenFlow [12] is the *de facto* standard implementation of Software-Defined Networking (SDN), which is a network architecture to realize network programmability and a centralized control of switches. It allows us to apply different routing algorithms and methods on the interconnect in a software programming manner. In this preliminary investigation, an OpenFlow controller is deployed on the management node to control the OpenFlow switches via the management network. Table 3 summarizes software used for this preliminary investigation.

TABLE 3. Software used for this investigation.

Software	Version
Ryu	4.15
Slurm	17.02.10
GNU C Compiler	7.3.1
OpenMPI	3.0.0rc5

TABLE 4. The port mirroring configuration.

The target port	The destination port
port1 on ofs1	port3 on ofs1
port2 on ofs1	port4 on ofs1
port1 on ofs2	port3 on ofs2
port2 on ofs2	port4 on ofs2

Also, the four monitoring nodes were configured to monitor and capture all outgoing packets passing through on the core switches ofs1 and ofs2. For this configuration, the port mirroring function on the core switches was used. Table 4 shows the port mirroring configuration.

C. JOB SUBMISSION EXPERIMENT

In this investigation, we submitted a set of six jobs, each of which executes MPI_Alltoall collective communication repeatedly, fifty times. Each job performs the stage-in and the stage-out operation before and after computation. For the staging operation, we have set up a simple function that uses SCP to move 2GB data generated by the *dd* command between *tmpfs*, which is a memory-based file system, on the storage node and tmpfs on a computing node that executes a job. Each job is set to request three computing nodes so that two jobs are executed simultaneously on six computing nodes and then the traffic collision between the two types of the traffic is observed. The reason why we adopt MPI_Alltoall collective communication with which data is exchanged among all processes composing a job is explained from our intention that we want to investigate how the traffic collision causes degradation in the job throughput. Furthermore, to investigate how the traffic collision causes degradation in job throughput, we adjusted the job scheduler configuration so that the processes of a job are not allocated to a set of computing nodes linked to the same edge switch.

To observe how the two types of the traffic consume the bandwidth on the links from the core switches to the edge switches, *tcpdump* was deployed on the monitoring nodes to capture all packets on the core switches during the job submission experiment. After the job submission experiment, we calculated the bandwidth consumed by each type of the traffic using the captured packets. In the calculation, the packets were classified into the staging traffic and the inter-node communication traffic based on the source and destination MAC address recorded on their packet headers. After that, for each classified group of packets, we calculated the bandwidth used at time t on a port of the core switches by summing the packet lengths of the packets captured between time t and time $t + 1$.

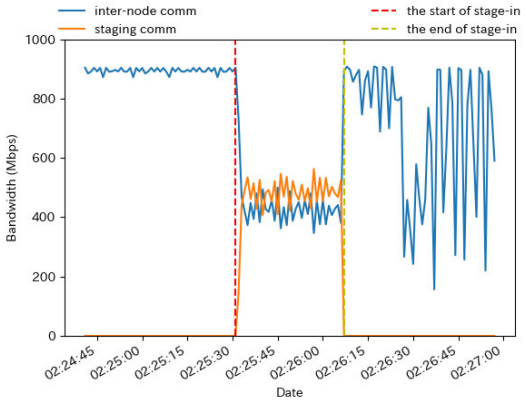


FIGURE 3. The bandwidth consumed by each traffic on port2 on ofs1.

To observe how the job execution time is changed with and without traffic collision on the interconnect, we have measured the job execution time with and without traffic collision in the job submission experiments. For the measurement of the job execution time without the traffic collision, we configured the staging traffic to pass through not the interconnect, but the management network.

D. EXPERIMENTAL RESULT

Fig. 3 shows how each type of the traffic consumed the bandwidth on port2 on ofs1 when a stage-in operation occurred under the situation where two jobs are executed as shown in Fig. 4. In Fig. 3, the Y axis indicates the bandwidth consumed by each type of the traffic, and the X axis indicates time. From this graph, it is supposed that the following situation happened. From 02:24:45 to 02:25:31, only the inter-node communication traffic flowed on port2 on ofs1. The stage-in operation started at 02:25:31 and finished at 02:26:07. During this stage-in operation, the staging traffic and the inter-node communication traffic co-existed on the link between ofs1 and ofs4 and the link between ofs3 and ofs1. Also, it was observed that the bandwidth used by the inter-node communication traffic was lowered to 424 Mbps from 894 Mbps. In addition, the bandwidth used by the staging traffic was 484 Mbps. After that, only inter-node communication traffic flowed on port2 on ofs1 again. Note that a similar traffic pattern was observed on the other ports (port1 on ofs1, port1 on ofs2 and port2 on ofs2) and during different time periods.

Fig. 5 shows the result of the job submission experiment. The leftmost bar shows the average execution time of 300 jobs (6 jobs $\times$ 50 job submissions). The result indicates that the average execution time under the situation where the traffic collision occurs was larger than the average execution time without the traffic collision. In detail, the job execution time was increased by 7.7%. The job execution time is composed of application execution time, stage-in operation time, and stage-out operation time. We also plot each component time in Fig. 5. From this result, it is obvious that the traffic collision causes the degradation in the job throughput.

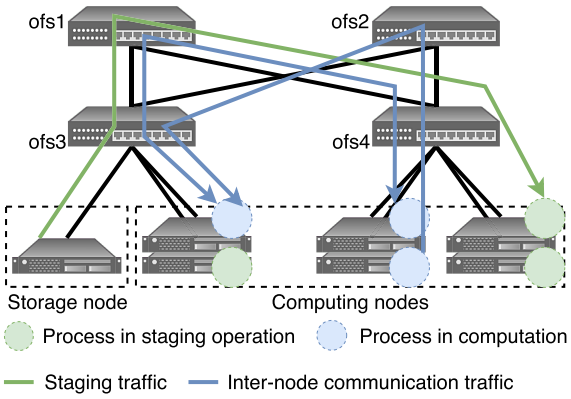


FIGURE 4. The process allocation during the stage-in operation.

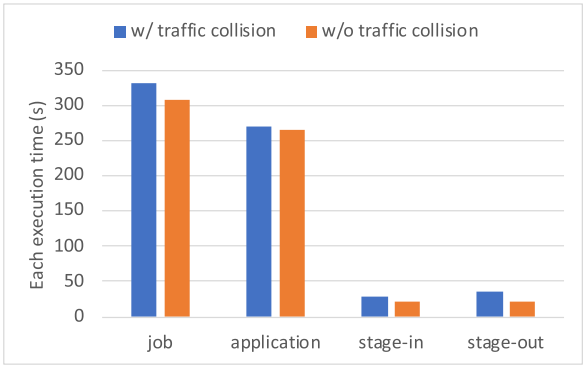


FIGURE 5. Average execution time with and without traffic collision.

IV. DYNAMIC TRAFFIC COLLISION AVOIDANCE

A. DESIGN

As shown in Section III, traffic collision causes degradation in job throughput. Based on the results of the preliminary investigation, we propose a method to avoid traffic collision to gain performance acceleration by applying dynamic control on the interconnect using SDN. The basic idea behind our proposed method is illustrated in Fig. 6. The proposed method consists of two functions: *dedicated path search* and *progressive path switching*. With the dedicated path search function, the proposed method searches a set of dedicated paths on the interconnect for each type of the inter-node communication traffic and the staging traffic. With the progressive path switching function, the proposed method configures the set of dedicated paths for each type of traffic while the staging traffic happens.

If the proposed method configures a set of dedicated paths for each type of traffic all the time, the utilization of the entire bandwidth available on the interconnect becomes low when any staging operation does not occur. Therefore, the proposed method takes a strategy to dynamically configure a new set of dedicated paths on the interconnect for each type of traffic and then diverts the two types of traffic to the corresponding set of dedicated paths in a progressive way without blocking any traffic. Note that we adopted ECMP as a simple load-balancing routing method for the situation when no staging operation happens.

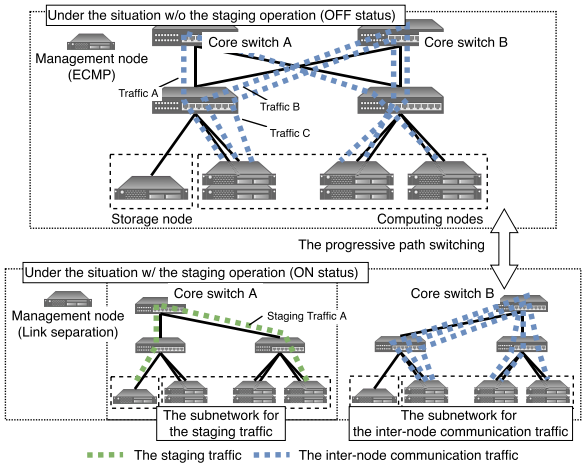


FIGURE 6. The basic idea behind the proposed method.

1) THE DEDICATED PATH SEARCH FUNCTION

The dedicated path search function searches a set of dedicated paths according to our proposed *link separation method*. The link separation method is composed of the following three steps: core switch allocation, subnetwork formation and path search. In the first core switch allocation step, the core switches composing the fat-tree interconnect are divided into the staging traffic group and the inter-node communication traffic group based on the amount of the inter-node communication traffic passing through each core switch. Although there is room for consideration about a better division algorithm of core switches, we have adopted a simple way to equally divide core switches into the two groups in this stage of the research. In the second subnetwork formation step, the dedicated path search function forms a subnetwork for each type of traffic so that the corresponding subnetwork does not contain any core switches allocated to the other type of traffic. In the third path search step, the dedicated path search function searches the shortest path for any pair of the nodes in the subnetwork. In the proposed method, the Dijkstra is used as an algorithm for searching the shortest path.

The dedicated path search function works as follows for the example in Fig. 6. *Core switch A* and *Core switch B* are allocated to two groups. Core switch A is allocated to the staging traffic group and Core switch B is allocated to the inter-node communication traffic group because more traffic flows on Core switch B than Core switch A. Next, the subnetwork for the staging traffic is formed so that it contains the Core switch A, which belongs to the staging traffic group. The subnetwork for the inter-node communication traffic is formed so that it contains the Core switch B, which belongs to the inter-node communication group. On the subnetwork for the staging traffic, the dedicated path search function searches the shortest path for *staging traffic A*, which newly appeared between the computing node and the storage node. On the other hand, on the subnetwork for the inter-node communication traffic, the function searches the shortest path for each *traffic A*, *traffic B* and *traffic C*. The proposed method

forces each type of traffic to flow on its shortest paths and consequently each type of traffic passes through a different core switch.

2) THE PROGRESSIVE PATH SWITCHING FUNCTION

The progressive path switching function is designed to interact with the job scheduler. We assume that ECMP works under the situation where no staging operation happens (OFF status), while on the other hand, the link separation method works under a situation where any staging operation happens (ON status). The progressive path switching function starts when this function is notified by the job scheduler that the staging operation happens. In detail, this function switches the path determined by ECMP to the dedicated path determined by the link separation method for each type of traffic, immediately after being notified of the beginning of the staging operation, or after this function recognizes the situation is under ON status. Then it switches the dedicated path by the link separation method back to the path by ECMP right after being notified of the end of the staging operation, or after this function recognizes the situation is under OFF status.

B. OpenFlow

An OpenFlow network is composed of an OpenFlow controller and OpenFlow switches. In the OpenFlow network, the OpenFlow controller controls how the OpenFlow switches forward packets centrally. The OpenFlow switches only forward packets, while the control and the forwarding are performed on each switch independently in a conventional network. Additionally, the OpenFlow controller is developed by a network administrator with a development framework such as Ryu and Trema [13]. Therefore, OpenFlow provides programmable traffic control and the OpenFlow controller can control the traffic according to an off-network event, which occurs outside the network, such as the beginning of a staging operation.

The following describes how the OpenFlow controller controls the OpenFlow switches. Each OpenFlow switch has a flow table, which is a set of flow entries. A flow entry is a rule to decide how to forward packets and it is comprised of an action, a match field and a priority. The action defines how to handle a packet and it generally decides which port to forward the packet from. The match field defines what kind of packets the action of the flow entry holding the match field is applied to and the action is applied to the packets whose packet header has the same information (MAC address, IP address, port number, and so on) described on the match field. The priority is for when multiple flow entries are matched to a packet, and the flow entry with the higher priority is applied to the packet. When an OpenFlow switch receives a packet, it finds the flow entry matched to the packet from its flow table and it forwards the packet according to the action of the found flow entry. The OpenFlow controller controls the OpenFlow switches by installing the flow entries to each OpenFlow switch.

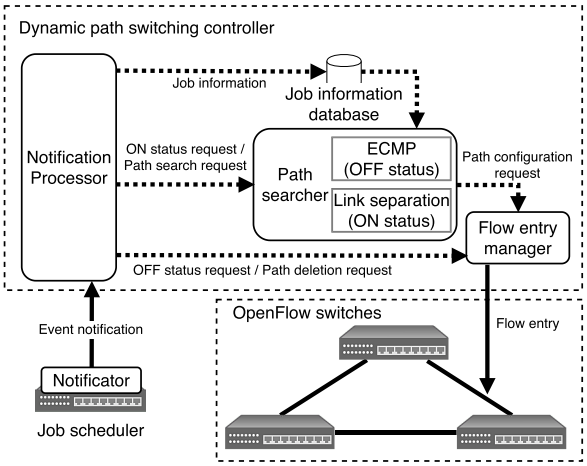


FIGURE 7. The architecture of the proposed method.

C. IMPLEMENTATION

Fig. 7 shows the implementation detail of the proposed method. The proposed method was implemented using the OpenFlow framework described in Section IV-B. The implementation is composed of a *notificator* module and *dynamic path switching controller* comprised of *notification processor*, *job information database*, *path searcher* and *flow entry manager* modules. The dynamic path switching controller receives event notifications regarding job and staging operation sent from the notificator module and then configures a set of dedicated paths for each type of traffic on the interconnect based on the event notifications. The notificator module, notification processor module and flow entry manager module offer the progress path switching function, and the path searcher module provides the dedicated path search function. The job information database module is used by the notification processor module and the path searcher module.

1) NOTIFICATOR MODULE

The notificator module notifies events regarding jobs and staging operations to the notification processor. For example, when a job starts, the notificator module delivers a job event notification that contains the job ID and the flag information identifying the job being started as well as the information on the computing nodes on which jobs are allocated. Also, when a job ends, it delivers a job event notification that contains the job ID and the flag information identifying the job being finished. On the other hand, when a staging operation starts or ends, the notificator module sends a staging event notification that contains the job ID and the flag information identifying the staging operation being started or finished.

2) JOB INFORMATION DATABASE MODULE

The job information database module holds *job information*. It is composed of the ID of a running job, the ID of the computing nodes on which the job is allocated and the flag information on whether the staging operation of the job happens. This job information is used by the notification

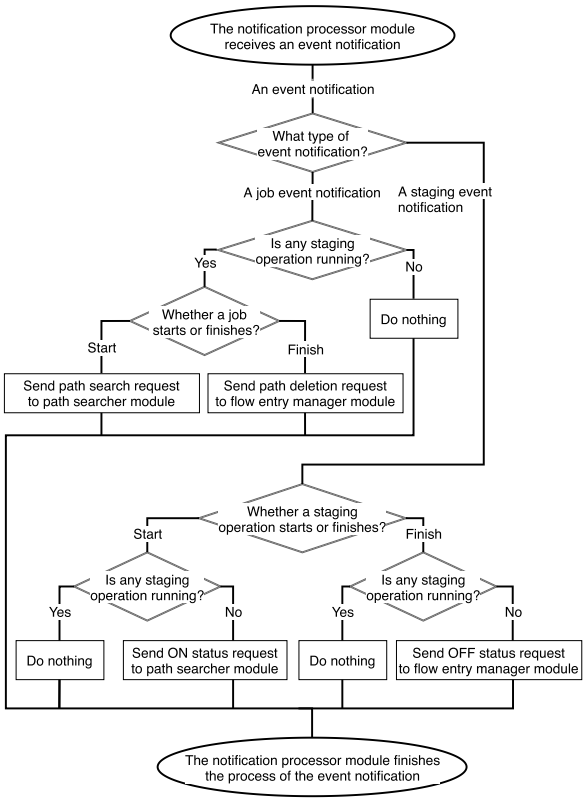


FIGURE 8. The flowchart of the path switching on the notification processor module.

processor module when the notification processor module performs the switching of the routing algorithm. Furthermore, the path searcher module searches a set of dedicated paths for each type of traffic based on the job information.

3) NOTIFICATION PROCESSOR MODULE

The notification processor module controls the path switching by instructing the path searcher module and the flow entry manager module to configure and delete the dedicated paths on the interconnect. When the notification processor module receives an event notification, it parses the event notification and then updates the job information held by the job information database module using the parsed information. After the job information is updated, the notification processor module instructs the path searcher module to configure a set of dedicated paths or instructs the flow entry manager module to delete a set of dedicated paths. For the instruction, the notification processor module sends a request regarding the switching of the routing algorithm or the control of the path search under the link separation method according to the flowchart shown in Fig. 8. The control of the path search under the link separation is mentioned in Section IV-C4 in detail.

Four kinds of requests sent by the notification processor module to the path searcher module or the flow entry manager module exist: *ON status request*, *OFF status request*, *path search request* and *path deletion request*. ON status request and OFF status request are used for the switching of the routing algorithm, and path search request and path deletion request are used for control of the path search under the link separation method. The notification processor module sends each type of the request as follows.

ON status request
: When a staging operation happens under OFF status, the notification processor module sends an ON status request to the path searcher module.

OFF status request
: When all the staging operations are finished, the notification processor module sends an OFF status request to the flow entry manager module.

Path search request
: When a new job starts under ON status, the notification processor module sends the path search request to the path searcher module for configuration of paths for the traffic caused by the started job.

Path deletion request
: When a job is finished under ON status, the notification processor module sends the path deletion request to the flow entry manager module.

4) PATH SEARCHER MODULE

The path searcher module performs the path search for the paths determined by ECMP under OFF status and for the set of dedicated paths by the link separation method under ON status. The searched path information, which is expressed as the array of node and switch IDs such as [*computing_node*1, *edge_switch*1, *core_switch*1, *edge_switch*2, *computing_node*2], is sent to the flow entry manager module so that the flow entry manager module configures the switches composing the interconnect.

In the case of ECMP (OFF status), the path search is performed only when the dynamic path switching controller is initialized and launched. The information on the searched path is sent to the flow entry manager module. On the other hand, the path search is performed with the link separation method whenever the path searcher module receives any ON status request or path search request from the notification processor module. After that, the information on the searched path is sent to the flow entry manager module.

Implementation of the link separation method is as follows. This method is composed of three steps as mentioned in Section IV-A: the core switch allocation step, the subnetwork formation step and the path search step. The path searcher module performs these three steps in order when receiving an ON status request. On the other hand, it performs only the path search step when receiving a path search request. The ON status request case is for searching a set of dedicated paths for the traffic caused by running jobs and the path search request case is for searching a set of dedicated paths for the traffic caused by a started job.

The core switch allocation step was implemented so that the path switching does not affect any existent inter-node

TABLE 5. The definition of the variables.

Variable	Definition
$jobs$	A set of job IDs of running jobs
$alloc\_com\_nodes$	The computing nodes allocated to the job specified by a job ID
$st\_subnetwork$	Subnetwork for the staging traffic
$in\_subnetwork$	Subnetwork for the inter-node communication traffic

communication traffic. Specifically, division of the core switches are conducted based on the number of flow entries to force more paths of the inter-node communication traffic to pass through the same core switches before and after the dedicated paths are configured on the interconnect. The path searcher module allocates half of the core switches with more flow entries than those of the other core switches to the inter-node communication traffic group because the more paths pass through a core switch, the more flow entries are installed to the core switch. On the other hand, it allocates the other core switches to the staging traffic group.

Next, the path search step was implemented based on the path search algorithm shown in Algorithm 1. Table 5 shows the variables used in this algorithm. Two variables, *jobs*, *alloc_com_nodes* hold the job information recorded by the notification processor module, and two variables, *st_subnetwork*, *in_subnetwork* are the subnetworks calculated in the subnetwork formation step of the dedicated path search function. This algorithm is composed of the staging traffic path search (line 2–10), the inter-node communication traffic path search (line 11–20) and the path configuration request step (line 21–24). In the staging traffic path search step, the path searcher module searches a path for any pair of the storage nodes and the computing nodes allocated to a job on *st_subnetwork*. In the inter-node communication traffic path search step, the path searcher module searches a path for any pair of the computing nodes allocated to a job on *in_subnetwork*. In the path configuration request step, the path searcher module sends a path configuration request, which contains the searched path information, for each path searched in two above steps to the flow entry manager module. In this way, the number of dedicated paths for each type of traffic is reduced so that the flow entry manager module takes less time for the configuration of the dedicated paths.

Again, for the example in Fig. 9, the path search based on this path searcher module works as follows. First, the path between the computing nodes *c1* and *c3* allocated to the job *J1* is configured according to ECMP under OFF status. When the job *J2* starts a stage-in operation after J2 starts on the computing nodes *c5* and *c6*, the notification processor module sends an ON status request to the path searcher module. On receiving the ON status request, the path searcher module searches a set of dedicated paths for the traffic caused by J1 and J2, and then it sends the path configuration requests to configure the dedicated path for the inter-node communication traffic caused by J1 and the requests for the staging traffic caused by J2 on the interconnect. The path configuration request to configure the former dedicated path contains the searched path information [$c1$, $sw3$, $sw1$, $sw4$, $c3$] and the requests to configure the latter dedicated path contains the searched path information [$storage$, $sw3$, $sw2$, $sw4$, $c5$]. When job *J3* starts its computation under this situation after J3 starts on the computing nodes *c2* and *c4*, the notification processor module sends a path search request to the path searcher module. On receiving the path search request, the path searcher module searches a set of dedicated paths for the traffic caused by the started job J3, and then it sends the path configuration request to configure the dedicated paths for the inter-node communication traffic caused by J3 using the same core switch sw1 used by the request for the inter-node communication traffic caused by J1. The path configuration request to configure this dedicated path contains the searched path information [$c2$, $sw3$, $sw1$, $sw4$, $c4$].

Algorithm 1 The Path Search of the Link Separation Method

```
1:  for job in jobs do
2:      ▷ The staging traffic path search step
3:      for src in storage_nodes do
4:          for dst in alloc_com_nodes[job] do
5:              path = Dijkstra(src, dst, st_subnetwork)
6:              path_list.add(path)
7:              path = Dijkstra(dst, src, st_subnetwork)
8:              path_list.add(path)
9:          end for
10:     end for
11:     ▷ The inter-node communication traffic path search step
12:     for src in alloc_com_nodes[job] do
13:         for dst in alloc_com_nodes[job] do
14:             if src == dst then
15:                 continue
16:             end if
17:             path = Dijkstra(src, dst, in_subnetwork)
18:             path_list.add(path)
19:         end for
20:     end for
21:     ▷ The path configuration request step
22:     for path in path_list do
23:         send_path(path)      ▷ To the flow entry manager module
24:     end for
25: end for
```

5) FLOW ENTRY MANAGER MODULE

The flow entry manager module configures paths when receiving the path configuration request sent from the path searcher module, and removes paths when receiving an OFF status request or a path deletion request sent from the notification processor module. When the flow entry manager module receives a path configuration request, it makes flow entries to configure the requested path and then installs them to the

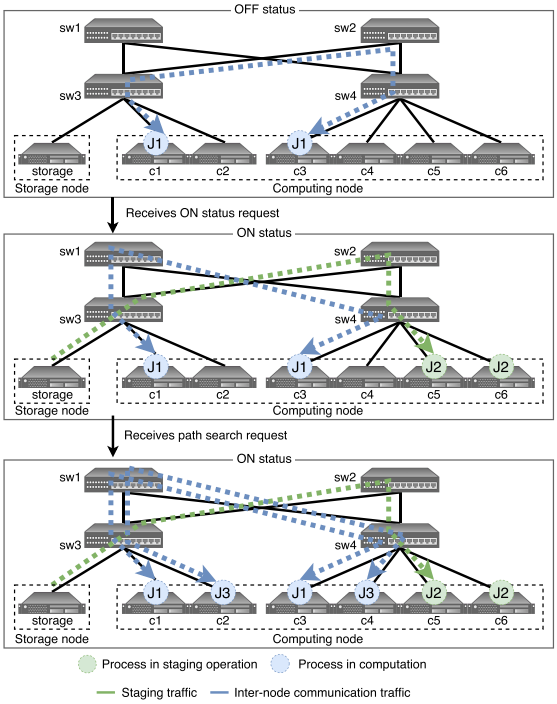


FIGURE 9. The path search example based on the path searcher module.

switches composing the interconnect. When the flow entry manager module receives an OFF status request, it removes all of flow entries for dedicated paths from the switches. When the flow entry manager module receives a path deletion request, it removes the flow entries for the dedicated paths for the traffic caused by a finished job.

To perform the switching of ON and OFF status seamlessly without blocking both the two types of traffic, the flow entry manager module manages the flow entries to configure the path by each routing algorithm so that the path by ECMP can be available to any traffic until the flow entries for the dedicated paths for each type of traffic are installed on each switch. Fig. 10 shows the switching mechanism based on this idea. In this switching mechanism, the flow entries for path configuration under OFF status are always on switches, while the flow entries for path configuration under ON status are always regenerated and installed with higher priority than the ones under OFF status.

V. EVALUATION

A. EVALUATION APPROACH

To investigate whether the proposed method can improve performance in job throughput or not, we perform the same job submission experiment on the same HPC cluster system used for the preliminary investigation summarized in Section III. For the routing algorithm on the interconnect, we used the proposed method instead of ECMP. We measure the total execution time of a job. Also, we observe how the bandwidth consumed by each type of traffic is changed over time.

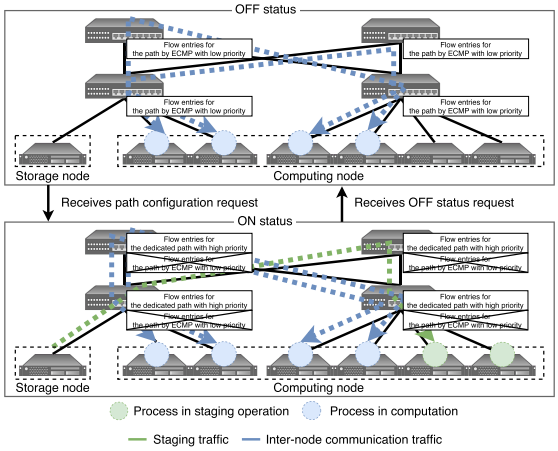


FIGURE 10. The switching mechanism of ON and OFF status.

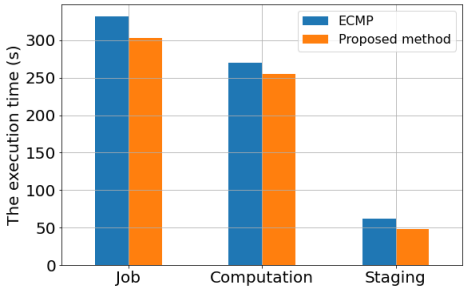


FIGURE 11. The average execution time of the jobs in the job sets.

B. EVALUATION RESULT

We show three types of evaluation results: job throughput, operation verification of the proposed method and progressive path switching. In the job throughput, we investigate whether the proposed method improves the job throughput in the job submission experiment. In the operation verification of the proposed method, we investigate whether our implemented dynamic path switching controller works properly. In the progressive path switching, we investigate how the progressive path switching function of the proposed method contributes to improvement in the job throughput.

1) JOB THROUGHPUT

Fig. 11 shows the result of the job submission experiment. This result is the average execution time of the jobs in the fifty job sets. In this result, we plot three component times: the job execution time, the computation time and the staging operation time. To investigate how our proposed dynamic traffic collision avoidance improves job throughput, we compare these times between the proposed method and ECMP. From this result, the job execution time of the proposed method is shorter than ECMP and its reduction rate is 8.7%. The computation time and the staging operation time of the proposed method are also shorter than ECMP and this indicates that the performance improvement of the inter-node communication and the staging operation contributes to improvement of the job throughput. The reduction rate of the computation time is 5.6% and the rate of the staging operation time is 22.0%.

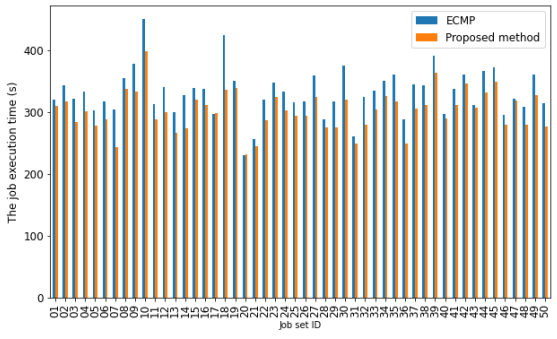


FIGURE 12. The average job execution time for each job set.

Fig. 12 shows the result for each job set submitted in the job submission experiment. Each bar in this result is the average job execution time of the six jobs in a job set. We observed that the job execution time under the proposed method is shorter than that under the ECMP method in 45 job sets. This fact indicates that the proposed method stably improves the job throughput.

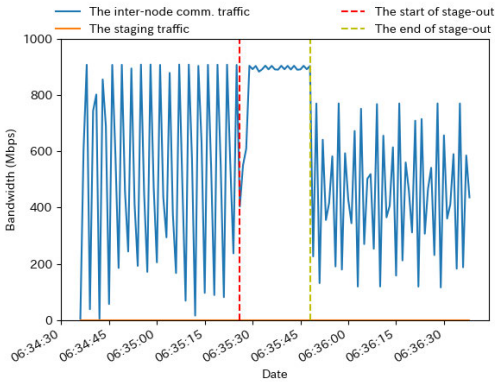
2) OPERATION VERIFICATION OF THE PROPOSED METHOD

Fig. 13 shows the bandwidth consumed on each port of the switches during the stage-out operation of a job in a job set when the proposed method is adopted for the routing algorithm on the interconnect. In Fig. 13, the Y axis indicates the bandwidth consumed by each type of the traffic, and the X axis indicates time. These graphs indicate that our implemented dynamic path switching controller configures a set of dedicated paths for each type of traffic properly while the stage-out operation occurs as follows. Until 06:35:26, only the inter-node communication traffic flowed on all the ports. Under this situation, the stage-out operation started at 06:35:26 and finished at 06:35:48. The change in the bandwidth from before the start of this stage-out operation to during this stage-out operation is illustrated as follows. First, Fig. 13(a) shows that the bandwidth used by the inter-node communication traffic was raised from 519 Mbps to 845 Mbps on port1 on ofs1. Second, Fig. 13(b) shows that the bandwidth used by the inter-node communication traffic was raised from 784 Mbps to 849 Mbps on port2 on ofs1 during this stage-out operation. Third, Fig. 13(c) shows that the bandwidth used by the inter-node communication traffic on port1 on ofs2 was lowered to 42 Mbps from 522 Mbps, and the bandwidth used by the staging traffic on port1 on ofs2 was raised from 0 Mbps to 721 Mbps. Finally, Fig. 13(d) shows that the bandwidth used by the inter-node communication traffic on port2 on ofs2 was lowered to 30 Mbps from 260 Mbps. These observation results indicate that ofs2 was used for the dedicated paths of the staging traffic and ofs1 was used for the dedicated paths of the inter-node communication traffic during this stage-out operation. The reason why the bandwidth used by the inter-node communication traffic on port1 and port2 on ofs2 was not 0 Mbps even though ofs2 was used for the dedicated paths of the staging traffic is that the inter-node communication traffic flowed there for approximately 2 seconds before the dynamic path switching controller allocated the dedicated paths for each type of traffic. After the stage-out operation finished, only the inter-node communication traffic flowed on all of ports again. In this way, the proposed method avoids the traffic collision between each type of traffic dynamically. This path switching is observed in the staging operations of the other jobs in the job set.

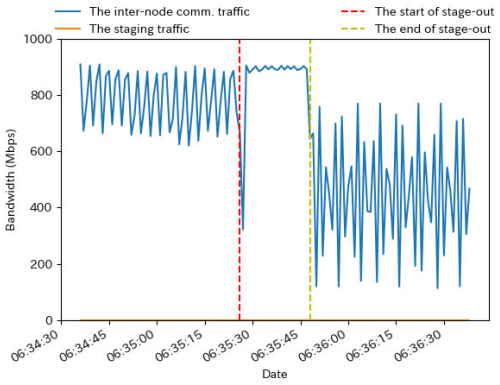
3) PROGRESSIVE PATH SWITCHING

Fig. 14 shows the average times of the progressive path switching between ON and OFF status. The path switching time from OFF status to ON status is from when the notificator module informs the dynamic path switching controller that a staging operation starts until when the controller completes the flow entry installs, and the path switching time from ON status to OFF status is from when the notificator module informs the controller that a staging operation finishes until when the controller completes the flow entry deletions. In this graph, we plot the path switching time from OFF status to ON status in the stacked bar graph composed of the processing times in the notification processor module, the path searcher module and the flow entry manager module. From this graph, there is a 2.20 seconds delay to complete the path switching from the paths determined by ECMP to the paths determined by the link separation method, while there is a 0.31 seconds delay to complete the path switching from the paths by the link separation method to the paths by ECMP. Moreover, the overhead time to complete the path switching from OFF status to ON status is 0.0085 seconds for the notification processor module, 0.17 seconds for the path searcher module and 2.01 seconds for the flow entry manager module. The breakdown of the overhead time shows that the flow entry installs on the flow entry manager module incurs the longest delay in the path switching from OFF status to ON status. On the other hand, the path switching time from ON status to OFF status is mostly occupied by the flow entry deletions on the flow entry manager module. The reason can be explained from the fact that the path searcher module is not performed in this path switching and the notification processor module takes a few milliseconds as shown above.

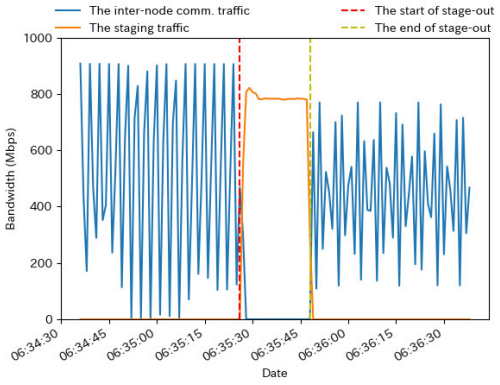
However, each type of traffic can continue to flow on the interconnect until the path switching completes because the progressive path switching works as shown in Fig. 15. The stage-in operation observed in these graphs started at 06:25:46 and finished at 06:26:09. Fig. 15(a) shows the staging traffic flows on port2 of ofs1 at 202 Mbps from 06:25:47 to 06:25:48. After that, the staging traffic flows on port2 of ofs2 at 735 Mbps from 06:25:49 to 06:26:09 as shown in Fig. 15(b), and the inter-node communication traffic flows on port2 of ofs1 as shown in Fig. 15(a). In this way, the progressive path switching function enables each type of traffic to pass through the paths by ECMP in 2.20 seconds on average until the path switching completes and suppresses the effect of the delay in the path switching from OFF status to ON status to the job throughput.



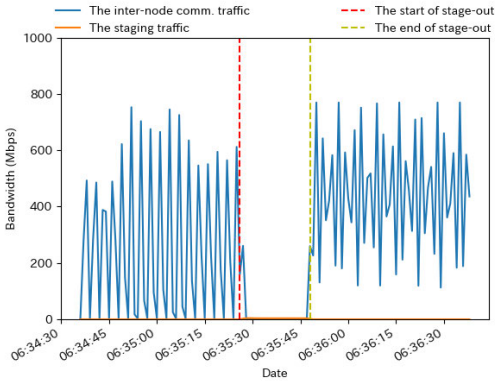
(a) The bandwidth consumed by each traffic on port1 on ofs1.



(b) The bandwidth consumed by each traffic on port2 on ofs1.



(c) The bandwidth consumed by each traffic on port1 on ofs2.



(d) The bandwidth consumed by each traffic on port2 on ofs2.

FIGURE 13. The bandwidth consumed on each port of the switches during a stage-out operation under ON status.

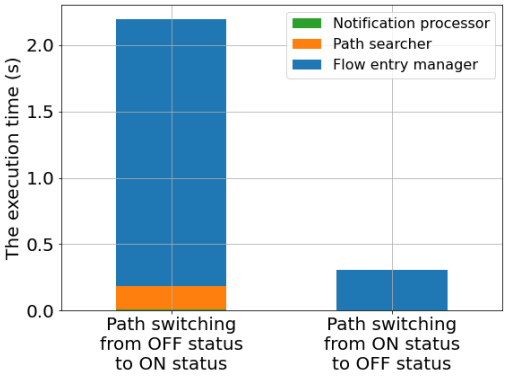
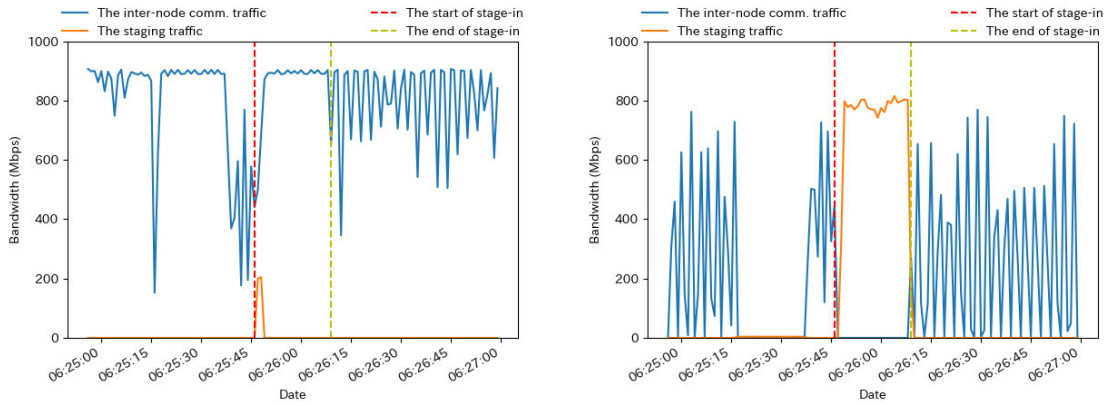


FIGURE 14. The average execution time of the progressive path switching between ON and OFF status.

VI. RELATED WORKS

An investigation of the interference between I/O traffic and inter-node communication traffic on a full-bisection fat-tree interconnect exists [14]. This investigation shows the interference degrades inter-node communication performance more than I/O performance. It has been also shown that several strategies, where the interference is reduced by configuring the traffic pattern of how traffic flows on the interconnect such as I/O sever placement, can improve the inter-node communication and the I/O performance. On the contrary, our proposed method focuses on the traffic control on the interconnect, which has not been considered in this investigation.

On the interconnect constructed with InfiniBand [15], several research studies [16], [17] have improved communication performance. In vFtree [16], each traffic identified by a pair of source and destination nodes is allocated to a different virtual lane to alleviate the performance degradation caused by Head-of-Line blocking. However, this method does not solve the problem that each type of traffic blocks each other due to the bottleneck of the bandwidth on the interconnect. A QoS-aware data-staging framework [17] targets the interference between inter-node communication traffic and I/O traffic, which is caused by processes to access the data on a global file system during computation. This framework allocates I/O traffic to different virtual lanes from the lanes used by inter-node communication to transfer the inter-node communication traffic with priority over I/O traffic so that I/O traffic does not affect the performance of the inter-node communication. Although this framework controls multiple types of traffic to improve the performance of the inter-node communication, this way to sacrifice the performance of I/O is contrary to our research issue to improve both



(a) The bandwidth consumed by each traffic on port2 on ofs1. (b) The bandwidth consumed by each traffic on port2 on ofs2.

FIGURE 15. The bandwidth consumed on port2 of each switch during a stage-in operation under ON status.

the inter-node communication performance and the staging operation performance.

Traffic load-balancing methods using SDN have been presented in [18]–[20]. Hedera [18] estimates the bandwidth required by traffic and allocates paths to the traffic based on the estimated bandwidth so that the traffic is distributed on the interconnect without the traffic collision. Dynamic load-balancing (DLB) [19] and SDN-based ECMP [20] monitor the bandwidth consumed by traffic, and allocate the path with the most available bandwidth to new traffic when the new traffic starts to flow on the interconnect. These traffic load-balancing methods fairly distribute traffic on the interconnect without looking into the characteristics of each type of traffic for the optimal bandwidth allocation to prevent traffic collision from occurring. However, based on the characteristic that inter-node communication traffic always flows on the interconnect while the staging traffic does not always flow on the interconnect, our proposed method switches the traffic control applied on the interconnect between ECMP and the link separation method when the staging operation starts or finishes.

The idea of using dynamic traffic control on the interconnect of HPC cluster systems is shared in [21]–[25]. In [21], an SDN-accelerated HPC cluster system has been proposed as a concept to integrate SDN into HPC cluster systems. Based on this concept, an SDN-enhanced job management system (JMS) [22], [23], which is a job scheduler to manage not only the computing nodes but also the interconnect to improve the job throughput, and SDN-enhanced MPI [24], [25], which enhances MPI protocol using SDN to improve the performance of inter-node communication, have been proposed. In SDN-enhanced JMS [22], [23], the job scheduler allocates computing nodes to a job based on the topology of the interconnect so that as many of the computing nodes as possible are connected in one hop, and then allocates the path with the most available bandwidth to a pair of each computing node allocated to the job for improving the inter-node communication performance. In SDN-enhanced MPI_Bcast [24], MPI_Bcast, which is one of the MPI functions, has been enhanced using SDN so that the amount of the traffic caused by MPI_Bcast is reduced. In [25], the coordination mechanism, which applies the traffic control based on the SDN-enhanced MPI in synchronization with the execution of the MPI functions, has been proposed to apply the SDN-enhanced MPI to real-world MPI applications. Although these research studies improve the performance of the inter-node communication, they do not assume that other types of traffic such as the staging traffic flow on the interconnect.

VII. CONCLUSION

In this article, we proposed a traffic collision avoidance method between staging traffic and inter-node communication traffic on the interconnect, targeting the cluster system with an oversubscribed interconnect shared by the storage and the computing nodes. Our proposed method dynamically switches routing algorithms based on whether the staging operation happens. Our proposed method allocates a set of dedicated paths determined by our proposed link separation method for each type of traffic while any staging operation happens (ON status). On the other hand, it allocates a set of paths determined by ECMP for the inter-node communication traffic while no staging operation happens (OFF status). The evaluation in this article indicates the proposed method is capable of avoiding traffic collision and can then accelerate the staging operation by 22.0% on our cluster system. Also, the evaluation indicates the overhead of the application incurred by the proposed method is negligible. Furthermore, 8.7% of the job execution time is reduced by the proposed method.

However, several issues remain to be tackled in the future. The first issue is to investigate how many core switches should be allocated to the staging traffic group and the inter-node communication traffic group in the dedicated path search function to further improve the communication performance of each type of traffic. This issue is difficult to achieve for the scale of the interconnect and the bisection bandwidth of the interconnect. The second issue is the

overhead reduction of traffic control in our proposed method to improve scalability for the scale of the cluster system. In our proposed method, a set of flow entries is generated per a pair of nodes, and consequently the traffic control time to install them on each switch is increased on a larger scale of the cluster system. The final issue is to control I/O traffic in addition to the staging traffic and the inter-node communication traffic. Although we proposed our traffic collision avoidance method assuming that the staging traffic and the inter-node communication flow on the interconnect in this stage of our research, it is necessary to investigate how to avoid traffic collision between three types of traffic because the computing nodes are allowed to read and write data on a global file system during computation in the real-world cluster system.

ACKNOWLEDGMENT

Chunghan Lee was with Fujitsu Laboratories Ltd., Kawasaki, Kanagawa 211-8588, Japan.

REFERENCES

[1] D. Castelvecchi, "How to hunt for a black hole with a telescope the size of earth," *Nature News*, vol. 543, no. 7646, p. 478, 2017.
[2] H. W. Meuer, E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. *TOP500: TOP500 Supercomputer Sites*. Accessed: May 22, 2020. [Online]. Available: https://www.top500.org
[3] M. Yokokawa, F. Shoji, A. Uno, M. Kurokawa, and T. Watanabe, "The K computer: Japanese next-generation supercomputer development project," in *Proc. 17th IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 371–372.
[4] P. J. Braam and P. Schwan, "Lustre: The intergalactic file system," in *Proc. 4th Ottawa Linux Symp.*, Jun. 2002, pp. 50–54.
[5] *Gluster*. Accessed: May 27, 2020. [Online]. Available: http://gluster.org
[6] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. 8th ACM SIGCOMM Conf. Data Commun.*, Oct. 2008, pp. 63–74.
[7] National Institute of Advanced Industrial Science and Technology. *ABCI: AI Bridging Cloud Infrastructure*. Accessed: May 25, 2020. [Online]. Available: https://abci.ai.
[8] A. Endo, R. Jingai, S. Date, Y. Kido, and S. Shimojo, "Evaluation of SDN-based conflict avoidance between data staging and inter-process communication," in *Proc. 15th Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2017, pp. 267–273.
[9] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple linux utility for resource management," in *Proc. 9th Job Scheduling Strategies Parallel Process.*, Jun. 2003, pp. 44–60.
[10] D. Thaler and C. E. Hopps, *Multipath Issues in Unicast and Multicast Next-Hop Selection*, document RFC 2991, Nov. 2000.
[11] T. Fujita. *Ryu SDN Framework*. Accessed: Jan. 22, 2020. [Online]. Available: https://github.com/osrg/ryu
[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[13] *Trema: Full-Stack OpenFlow Framework in Ruby/C*. Accessed: Apr. 2, 2020. [Online]. Available: https://trema.github.io/trema/
[14] K. A. Brown, N. Jain, S. Matsuoka, M. Schulz, and A. Bhatele, "Interference between I/O and MPI traffic on fat-tree networks," in *Proc. 47th Int. Conf. Parallel Process.*, Aug. 2018, pp. 1–10.
[15] T. Shanley and J. Winkles, *InfiniBand Network Architecture*. Boston, MA, USA: Addison-Wesley, Oct. 2002.
[16] W. L. Guay, B. Bogdanski, S.-A. Reinemo, O. Lysne, and T. Skeie, "vFtree—A fat-tree routing algorithm using virtual lanes to alleviate congestion," in *Proc. 25th IEEE Int. Parallel Distrib. Process. Symp.*, May 2011, pp. 197–208.
[17] R. Rajachandrasekar, J. Jaswani, H. Subramoni, and D. K. Panda, "Minimizing network contention in InfiniBand clusters with a QoS-aware data-staging framework," in *Proc. 14th IEEE Int. Conf. Cluster Comput.*, Sep. 2012, pp. 329–336.
[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, vol. 10, Apr. 2010, p. 19.
[19] Y. Li and D. Pan, "OpenFlow based load balancing for fat-tree networks with multipath support," in *Proc. 12th IEEE Int. Conf. Commun.*, Jun. 2013, pp. 1–5.
[20] R. Dewanto, R. Munadi, and R. M. Negara, "Improved load balancing on software defined network-based equal cost multipath routing in data center network," *Jurnal Infotel*, vol. 10, no. 3, pp. 157–162, Aug. 2018.
[21] S. Date, H. Abe, D. Khureltulga, K. Takahashi, Y. Kido, Y. Watashiba, P. U-Chupala, K. Ichikawa, H. Yamanaka, E. Kawai, and S. Shimojo, "SDN-accelerated HPC infrastructure for scientific research," *Int. J. Inf. Technol.*, vol. 22, no. 1, pp. 1–30, Jun. 2016.
[22] Y. Watashiba, S. Date, H. Abe, Y. Kido, K. Ichikawa, H. Yamanaka, E. Kawai, S. Shimojo, and H. Takemura, "Efficacy analysis of a SDN-enhanced resource management system through NAS parallel benchmarks," *Rev. Socionetwork Strategies*, vol. 8, no. 2, pp. 69–84, Dec. 2014.
[23] M. Shimizu, Y. Watashiba, S. Date, and S. Shimojo, "Adaptive network resource reallocation for hot-spot avoidance on SDN-based cluster system," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2016, pp. 608–613.
[24] H. Morimoto, K. Dashdavaa, K. Takahashi, Y. Kido, S. Date, and S. Shimojo, "Design and implementation of SDN-enhanced MPI broadcast targeting a fat-tree interconnect," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2017, pp. 252–258.
[25] K. Takahashi, S. Date, D. Khureltulga, Y. Kido, H. Yamanaka, E. Kawai, and S. Shimojo, "UnisonFlow: A software-defined coordination mechanism for message-passing communication and computation," *IEEE Access*, vol. 6, pp. 23372–23382, 2018.

ARATA ENDO (Member, IEEE) received the B.E. and M.E. degrees from Osaka University, Osaka, Japan, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the Graduate School of Information Science and Technology. Since 2020, he has been a Specially-Appointed Researcher with the Cybermedia Center, Osaka University. His research field is computer science. His research interests include high-performance computing, visualization, and networking. He is a member of IPSJ.

HIROKI OHTSUJI received the B.E., M.E., and Ph.D. degrees from the University of Tsukuba, in 2011, 2013, and 2016, respectively. He has worked as a Research Aide with the Argonne National Laboratory, from May 2013 to August 2013. From 2014 to 2016, he was a Research Fellow with JSPS. Afterwards, he joined Fujitsu Laboratories Ltd. His research interests include utilizing high-speed networks and memory devices to improve the performance of storage systems for high-performance computing and machine learning systems.

ERIKA HAYASHI received the B.S. and M.S. degrees in science from Nara Women's University, in 2017 and 2019, respectively. Afterwards, she joined Fujitsu Laboratories Ltd. She is currently engaged in research and development of high-performance computing file systems. Her research interests include persistent memory systems, storage systems, and machine learning.

EIJI YOSHIDA received the B.S. and M.S. degrees from Hiroshima University, in 1998 and 2000, respectively, and the Ph.D. degree in electrical engineering from Tohoku University, in 2015. He joined Fujitsu Laboratories Ltd., in 2000. He was a Visiting Scholar with Stanford University, from 2007 to 2008. His research interests include persistent memory systems, distributed storage systems, and cloud computing.

CHUNGHAN LEE received the Ph.D. degree from the Department of Electronic and Information Engineering, Toyohashi University of Technology, Japan, in 2013. From 2013 to 2019, he has worked with Fujitsu Laboratories Ltd., Japan. He is currently a Senior Researcher with Toyota Motor Corporation, Japan. His research interests include SDN/NFV/cloud and network measurement/analysis.

SUSUMU DATE (Member, IEEE) received the B.E., M.E., and Ph.D. degrees from Osaka University, in 1997, 2000, and 2002, respectively. He was an Assistant Professor with the Graduate School of Information Science and Technology, Osaka University, from 2002 to 2005. He has worked as a Visiting Scholar with the University of California at San Diego, in 2005. From 2005 to 2008, he has worked as a Specially-Appointed Associate Professor on the internationalization of education with the Graduate School of Information Science and Technology, Osaka University. Since 2008, he has been an Associate Professor with the Cybermedia Center, Osaka University. His research field is computer science. His current research interests include cloud, cluster, grid, high-performance computing, and their applications. He is a member of IPSJ.

SHINJI SHIMOJO (Member, IEEE) received the M.E. and Ph.D. degrees from Osaka University, Japan, in 1983 and 1986, respectively. He was an Assistant Professor with the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University, in 1986, and an Associate Professor with the Computation Center, from 1991 to 1998. During this period, he also worked for a year as a Visiting Researcher with the University of California at Irvine. He has been a Professor with the Cybermedia Center (then the Computation Center) at Osaka University, since 1998, and from 2005 to 2008, he was the Director of the Center. He was an Executive Researcher and the Director of the Network Testbed Research and Development Promotion Center, National Institute of Information and Communications Technology, from 2008 to 2011. He is currently the Director of the Cybermedia Center. His current research interests include wide variety of multimedia applications, peer-to-peer communication networks, ubiquitous network systems, and the IoT systems. He is a member of IEICE and a Fellow of IPSJ. He is a Founding Member of PRAGMA and CENTRA. He was awarded the Osaka Science Prize in 2005. He was awarded by the Minister of Internal Affairs and Communications in 2017.