2016 IEEE 8th International Conference on Cloud Computing Technology and Science

Adaptive Network Resource Reallocation for Hot-spot Avoidance on SDN-based Cluster System

Masaharu Shimizu*, Yasuhiro Watashiba†, Susumu Date*, Shinji Shimojo*
* Osaka University, Japan
Email: {masaharu.shimizu@ais., date@, shimojo@}cmc.osaka-u.ac.jp
† Nara Institute of Science and Technology, Japan
Email: watashiba@is.naist.jp

***Abstract*—On a cluster system running behind the Cloud computing, multiple processes are generated from most applications and then executed on multiple computing nodes. Their processes communicate with each other during their execution. The communication performance among multiple processes plays an important role in the total execution performance of an application. The SDN-enhanced JMS, which we have developed in the previous work, offers the framework that allows administrators to prescribe resource provisioning way using the information on both computing and network resources on a cluster system. However, an advanced and effective method for providing an appropriate set of computing and network resources to jobs is still an issue to be tackled. In this paper, we aim to design and implement an adaptive network resource allocation method with which network resources can be reallocated to jobs in execution, in response to the change of process placement occurred at job dispatch and termination events. Our evaluation shows that the proposed method can reallocate network resources of jobs in execution, and suppress the degradation of network throughput of jobs.**

***Keywords*-Software-Defined Networking, OpenFlow, Network Resource Management, Resource Reallocation, Cluster System**

I. Introduction

Most of recent scientific researchers have necessitated high-performance computing environment for performing large-scale simulations and data analysis. Accompanied with the rising computing demands and needs from various scientific fields, a diversity of computing patterns have emerged. For the reason, the Cloud computing has gathered much attention and concern as a computing technology that meets the diverse computing needs. In general, the Cloud computing environment is a cluster system of computing resources. The computing resources are connected with networks called interconnects.

On an ordinary cluster system, most scientific applications are executed in a distributed and parallel computing fashion for gaining high-performance. For the distributed and parallel computing on the cluster system, multiple processes are generated and then executed on multiple computing nodes. Their processes communicate with each other for data exchange and synchronization. Therefore, the communication performance among multiple processes heavily affects on the total execution performance of an application.

Generally, a cluster system has a resource management system (RMS) deployed for the purpose of job scheduling, system load balance, or high-throughput computation. In reality, however, most traditional RMSs available today focus on and handle only computing resources as the target of resource management, despite the great importance of communication performance. From the observation above, we have developed SDN-enhanced Job Management System (JMS) [1] as a network-aware RMS that offers the framework that allows administrators to prescribe their own resource provisioning way using the information on both computing and network resources on a cluster system. However, an advanced and effective method for providing an appropriate set of computing and network resources users' job requests is still an issue to be explored, although we have already developed a simple method on the top of SDN-enhanced JMS for allocating an appropriate set of both resources to a job based on process placement and network utilization. In this paper, we explore an adaptive network resource allocation method with which network resources can be reallocated to jobs in execution, in response to the change of process placement occurred at job dispatch and termination events. With the adaptive network resource allocation, the inefficient network allocation situation occurred by the simple method is avoided.

This paper is structured as follows. Section II briefly reviews traditional resource management technologies and SDN-enhanced JMS, our previous work, for further discussion in the proceeding sections. In section III, we explain a case where an inefficient network allocation takes place under the simple method without reallocation of network paths to jobs in execution. Subsequently, technical issues to be tackled are derived and clarified. Section IV proposes the adaptive network allocation method. In section V, we conduct experiments for evaluating the validity and effectiveness of the proposed method on a cluster system with a fat-tree interconnect. In section VI, we describe related works to our research. In section VII, we conclude this paper with our future direction.

II. Resource Management on Cluster System

A. Traditional Resource Management System

High-performance computing (HPC) harnesses computing power from multiple high-performance computers. On the HPC environment, Job Management System (JMS), exemplified by PBS [2], NQS [3] and Open Grid Scheduler/Grid

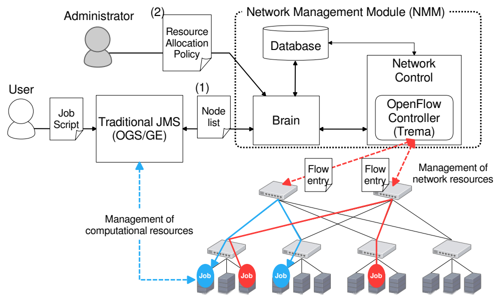

Administrator
(2)
Resource Allocation Policy
Network Management Module (NMM)
Database
Network Control
OpenFlow Controller (Trema)
User
Job Script
Traditional JMS (OGS/GE)
(1)
Node list
Brain
Flow entry
Flow entry
Management of network resources
Management of computational resources
Job

Fig. 1: Architecture of SDN-enhanced JMS

Engine (OGS/GE) [4], has been widely adopted for load-balancing computation workload.

The JMS receives computing requests from users and manages each computing request as a job. A set of computing resources are allocated to each job based on the computing request and the computing resource utilization, such as CPU load and memory usage. Resources allocated to a job are not released until its computation finishes. These traditional JMSs commonly handle only computing resources, in spite that network resources have a great influence on a job performance.

B. *SDN-enhanced JMS*

SDN-enhanced JMS [1] is our developed system that handles both computing and network resources as the targets of resource management for better provision of both resources. The SDN-enhanced JMS integrates OpenFlow [5], an implementation of Software-Defined Networking (SDN) [6], into a traditional JMS so that it takes advantage of network programmability brought by SDN for handling the interconnect as network resources.

An OpenFlow network is composed of OpenFlow Controller and OpenFlow switches. OpenFlow Controller manages packet flows in the network by sending instructions on how to deal with packets onto OpenFlow switches. In an OpenFlow network, the functionalities of network control and packet forwarding, which are built into a switch in a traditional network, are decoupled and offloaded to OpenFlow Controller and OpenFlow switches, respectively. OpenFlow Controller takes a role of deciding how to forward packets in a centralized manner. OpenFlow switches just have the responsibility of forwarding packets according to the instructions from the OpenFlow Controller.

Figure 1 shows the architecture of the SDN-enhanced JMS. Network Management Module (NMM) has been implemented as an extension module of the traditional JMS for the purpose of managing network resources on a cluster system. The NMM has three components: *brain*, *network control* and *database*. The OGS/GE has been utilized as the traditional JMS and OpenFlow controller has been implemented into the *network control* component using Trema [7], a development framework of the OpenFlow Controller.

The *brain* component is in charge of determining the allocation of computing and network resources based on their utilization information and resource allocation policy set in advance by the administrator. The *brain* component receives a *node list* from the traditional JMS (1). The *node list* is a candidate list of computing nodes determined by the traditional JMS. The order of computing nodes in the *node list* is basically determined according to computing resource utilization, that is, CPU load. After that, the *brain* component rearranges the order of computing nodes in the *node list*, by leveraging the available capacity information of network resources obtained from the *network control* component and the resource allocation policy set in advance by the administrator (2). The policy file describes the criteria for re-sorting computing nodes in the *node list* based on the above information and the resource allocation policy. After receiving the rearranged *node list* from the *brain* component, the traditional JMS finalize computing nodes to execute the job according to the order of the rearranged *node list*. After that, the traditional JMS notifies the *brain* component of the selected computing nodes. The *brain* component instructs the *network control* component to configure the network paths among the selected nodes.

The administrator can decide the criteria for choosing computing nodes as the resource allocation policy. In our previous work, we have developed a simple policy for the SDN-enhanced JMS. The simple policy seeks a set of computing nodes, which are considered to offer the maximum available bandwidth between computing nodes, from available computing nodes in the cluster system. While the policy selects the computing nodes and network paths for the job, the allocation of network resources for existing jobs is held.

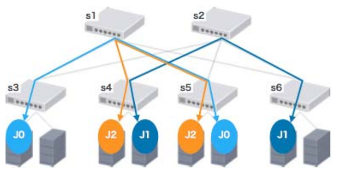
III. Problem and Issue

In our previous work, how to allocate network paths as network resources used for inter-process communication for a job has not been discussed. In this section, we first present a case where network resources are not efficiently utilized under our current simple method without reallocating network paths for jobs in execution. After that, we describe technical issues for suppressing inefficient network resource allocation.

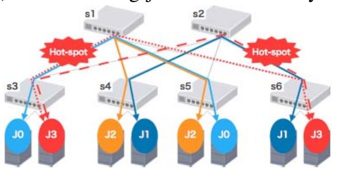
A. *Case: Inefficient Network Resource Allocation*

For describing a case, a cluster system with 2-level fat-tree interconnect is assumed as a high-performance computing environment behind a Cloud environment (Fig. 2). Bandwidth capacity of links is set 1.0 Gbps. Figure 2 illustrates a situation where sufficient network resources are not allocated to a job. Now we suppose that three jobs are running at first. For simplicity, it is assumed that each job (J0, J1, J2) requires two processes and each pair of processes communicate with each other in 0.5 Gbps.

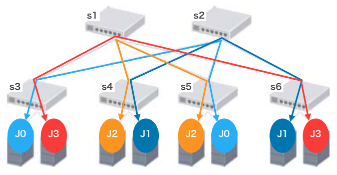
Under this assumed situation, a hot-spot on the interconnect of the cluster system may take place when new job J3 whose processes communicate with each other in 1.0 Gbps arrives. With the existent simple method that does not perform reallocation of network paths for jobs in execution, two nodes available at the moment are allocated to job J3 as shown in Fig. 2(b). In this case, there are two possible network paths to be allocated to J3: from s3 to s6 via s1, and from s3 to s6



(a) Three running jobs on a cluster system



(b) Network resource allocation with hot-spot



(c) Possible network resource allocation without hot-spot

Fig. 2: Example of the problem caused by static network resource allocation

via s2. However, as a network link between s1 and s3 is being utilized by J0, and a link between s2 and s6 is also being used by J1, the two possible network paths allocated to J3 inevitably overlap with the network path of J0 or J1. Also, the total bandwidth requirements from J0 and J3 (or j1 and j3) becomes 1.5 Gbps, which is more than the bandwidth capacity of each link. This situation causes two hot-spots on the interconnect of the cluster system, where the bandwidth capacity of network path does not satisfy jobs' network resource requirement (Fig. 2(b)). Such hot-spots degrade the available bandwidth of the network path. The degradation of the available bandwidth makes the execution time of the jobs longer.

B. Issues

Figure 2(c) indicates that there is a possible solution where network resources can be allocated to each job without hot-spots. For shifting from the situation in Fig. 2(a) to the situation in Fig. 2(c), it is necessary to reallocate network paths as network resources to be used by existing jobs in execution. In this example case, altering the network path being used by J0 is needed. In other words, the reason for causing hot-spots is that the network paths assigned to existing jobs in execution cannot be changed against the change of computation over time. From above-mentioned reason, the following two technical issues are achieved to avoid hot-spots.

1. *Hot-spot prediction*:
   The current SDN-enhanced JMS does not have any mechanism for detecting the evitable situation described in section III-A. How detection method or mechanism can be integrated is a technical issue.

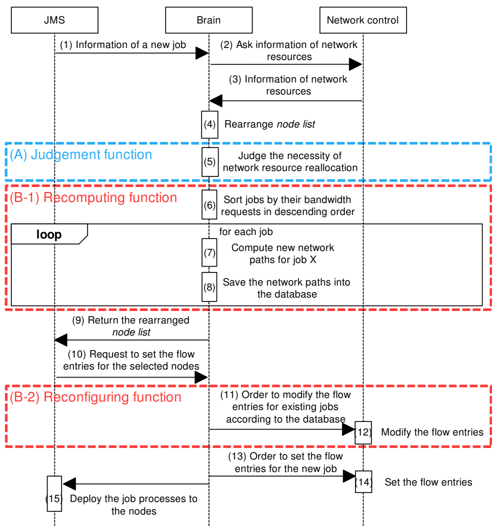

JMS
Brain
Network control
(1) Information of a new job
(2) Ask information of network resources
(3) Information of network resources
(4) Rearrange node list
(A) Judgement function
(5) Judge the necessity of network resource reallocation
(B-1) Recomputing function
(6) Sort jobs by their bandwidth requests in descending order
loop
for each job
(7) Compute new network paths for job X
(8) Save the network paths into the database
(9) Return the rearranged node list
(10) Request to set the flow entries for the selected nodes
(B-2) Reconfiguring function
(11) Order to modify the flow entries for existing jobs according to the database
12 Modify the flow entries
(13) Order to set the flow entries for the new job
14 Set the flow entries
15 Deploy the job processes to the nodes

Fig. 3: Procedure sequence of the proposed method

2. *Network path switching during job execution*:
   To avoid the competing overlap and sharing of network paths between user jobs, reallocation of network paths as network resources must be performed even during job execution adaptively to the change of computation over time. How this switching mechanism should be integrated into SDN-enhanced JMS is a technical issue.

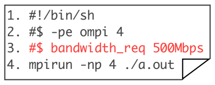
IV. Adaptive Network Resource Reallocation

1) Overview of the proposed method:

We propose an adaptive network resource reallocation method for achieving the issues described in section III-B. Figure 3 overviews the work sequence of the adaptive network resource reallocation method which we have designed and implemented. The dotted squares illustrated in Fig.3 are the procedures extended to the SDN-enhanced JMS for performing network resource reallocation.

Procedure (5) is executed for detecting any hot-spot in the case of rearranging the *node list* in the same way as the SDN-enhanced JMS ((1)-(4)). This procedure is added for judging the necessity of reallocating network resources of existing jobs. If the *brain* component recognizes that the network resource reallocation is necessary by the procedure, it determines new sets of network resources for existing jobs ((6)-(8)). Subsequently, in the same way as the SDN-enhanced JMS (9), the *brain* component returns the rearranged *node list* to the JMS. Then, the JMS requests the *brain* component to allocate network resources to the new job (10). On receiving the request, the *brain* component instructs the *network control* component to reallocate a new set of network resources to each existing job in execution for performing network resource reallocation ((11), (12)). After that, both computing and network resources was allocated to the new job ((13)-(15)).

2) Hot-spot detection (prediction):



```
#!/bin/sh
#$ -pe ompi 4
#$ bandwidth_req 500Mbps
mpirun -np 4 ./a.out
```

Fig. 4: Example of job script with network resource requirements

Hot-spot detection is completed by the *judgement function* ((A) in Fig. 3). The *judgement function* is designed to understand whether hot-spots are caused by allocating network resources to a new job or not, and then trigger the network resource reallocation for avoiding hot-spots if it judges that a hot-spot could occur. To judge whether a hot-spot occurs or not, the following information is necessary: bandwidth requirement of each job, status of computing resource allocation to jobs in execution, and network paths being allocated to them. The *judgement function* is implemented in the *brain* component so that the brain component can gather these information.

More specifically, this *judgement function* first attempts to make a plan to allocate a set of network resources to a new job without reallocating network resources for existing jobs in execution. The *judgement function* analyzes the planned resource allocation result. In the case that any hot-spot is predicted in the case of applying the plan, the *judgement function* triggers the network resource reallocation.

For judging whether any hot-spot occurs or not, the *judgement function* has to understand how much network resources all of jobs use. For understanding the bandwidth requirement of the new job, we add an interface for requesting a bandwidth requirement of a job in the job script as shown in Fig. 4. The *judgement function* utilizes the bandwidth requirement of the job from this user-written description.

3) Network path switching during job execution:

Network path switching during job execution is accomplished with *recomputing function* ((B-1) in Fig. 3) and *reconfiguring function* ((B-2) in Fig. 3). The *recomputing function* determines the allocation of network resources for not only a new job, but also existing jobs in execution. To allocate network paths as network resources to jobs, the following information is necessary: bandwidth requirement of each job and computing resource allocation status for jobs. For obtaining the information, the *recomputing function* is added into the *brain* component. The *brain* component can get the bandwidth requirements of jobs through the interface described in section IV-1. The *brain* component can also know where processes of jobs are running.

On the other hand, the *reconfiguring function* reflects the decision of network resource allocation for jobs in execution to the underlying interconnect. The conventional *network control* component can add only the configuration of network paths for a new job. For enabling the modification of the existing configuration of network paths, the *reconfiguring function* is added into the *network control* component.

In detail, the *recomputing function* first understands the bandwidth requirement of each job. Next, this function assigns network paths to a set of simultaneously executed jobs from a scratch in the order from the jobs with the larger bandwidth requirements to the smaller bandwidth. This is because network throughput of a job with large bandwidth requirement tends to be more degraded than one requiring small bandwidth when a hot-spot is caused. On this assignment, the *recomputing function* utilizes Dijkstra algorithm, whose cost of each link is total bandwidth requirements of jobs using the link. After determining the network resource allocation for all jobs, the *reconfiguring function* is triggered. In this function, the *brain* component instructs the *network control* component to allocate a new set of network resources computed by the *recomputing function*.

TABLE I: Job specifications

Category	Bandwidth requirement	Total data
H	1000Mbps	40GBytes
L	500Mbps	20GBytes

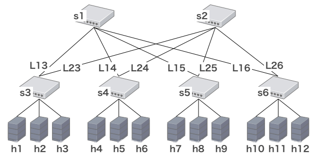


Fig. 5: Evaluation environment

V. Evaluation

We conducted two experiments to evaluate the validity and effectiveness of the proposed network resource reallocation method.

A. Evaluation Plan

In the experiments we conducted, we submit a set of jobs, each of which generated two processes to the evaluation environment described later. In the first experiment, we observe which network paths are used by jobs for confirming that the proposed method could avoid hot-spots (*exp-1*). In the second experiment, the network throughput of each job is measured to observe how much bandwidth the jobs in execution succeeded to obtain (*exp-2*). This second experiment is conducted to observe the proposed method could suppress the degradation of network throughput of jobs caused from hot-spot.

For focusing only on the communication among computing nodes and making it easy to understand how network paths are allocated to each job, we impose a restriction that a computing node could accommodate only one job process. The submitted job set is comprised of 6 jobs so that 12 processes simultaneously run on the same number of computing nodes on the cluster system explained later (Fig. 5). The job set is composed of two categories shown in Table I. A job belonging to category H requires larger bandwidth than one belonging to category L. Job1, job2, job3 and job4 are set category H, job5 and job6 are set category L. Each job was designed to utilize Iperf [8], which generates packet stream in the specified bandwidth.

B. Evaluation Environment

Figure 5 shows the cluster system as the evaluation environment used in the experiments. The cluster system is composed

TABLE II: Computing node specifications

Name	Spec
CPU	Intel Xeon CPU E5520 (2.27GHz) $\times$ 2
Memory	DDR3 (1333MHz) 12GB
NIC	on board Intel 82576 GbE
OS	Fedora 20

TABLE III: Combination of job and computing nodes without the proposed method

Job	Computing nodes
job1	h2, h6
job2	h1, h11
job3	h7, h8
job4	h5, h9
job5	h10, h12
job6	h3, h4

of 12 computing nodes, each of which has the specification as shown in Table II. The topology of the interconnect is a 2-level fat-tree composed of 6 OpenFlow switches, each of which is a logically divided switch from one OpenFlow switch (NEC UNIVERGE PF5240). All network links are connected via a GbE network.

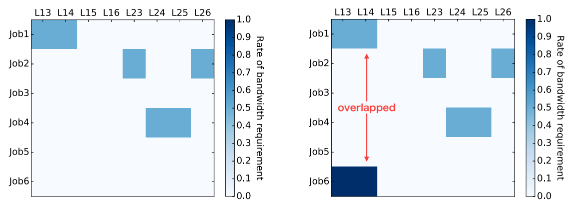
C. Evaluation Result

The results of *exp-1* when the proposed method was not adopted is shown in Fig. 6. The heatmap (Fig. 6) shows which links each job used. Its vertical axis represents job IDs and its horizontal axis represents link IDs between switches. The map means that a link was used if a cell corresponding the link is colored. Also, the color of cells illustrates how much the job required the bandwidth. Table III shows computing nodes on which processes of jobs ran.

Figure 6(a) shows the snapshot of showing which network links were used by jobs before job6 starts, and Fig. 6(b) when job6 starts. Before job6 started, network links were used by jobs without hot-spot. When job6 arrived, h3 and h4 were selected to accommodate two processes of job6. There had been two possible network paths for job6: path1 (using L13 and L14), and path2 (using L23 and L24). In this experiment, the path1 was selected for job6 (Fig. 6(b)). L13 and L14 were already utilized by job1. Furthermore, the total bandwidth requirements of job1 and job6 was 1500 Mbps. In this situation, hot-spots were caused on L13 and L14. If the path2 was selected for job6, hot-spots assumed to be caused on L23 and L24 because L23 was used by job2 and L24 was used by job4.

On the other hand, the results of *exp-1* when the proposed method was adopted is shown in Fig. 7. The heatmap describes which links each job used. Table IV shows computing nodes on which each job was executed.

In this experiment, it was observed that the proposed method succeeded to avoid hot-spots two times at the timing of job5 and job6 starting (Fig. 7). When job5 started, the proposed method succeeded to avoid hot-spots by reallocating network resources of job1 (*situation-1*). The proposed method also succeeded to avoid hot-spots by reallocating network paths of job3 (*situation-2*). In *situation-1*, network links were used by each job (Fig. 7(a)). After that, job5 arrived and then h8 and h12 were selected to place processes of job5, there had



(a) Before job6 starts **(b)** After job6 starts

Fig. 6: Network links usage without the proposed method

TABLE IV: Combination of job and computing nodes with the proposed method

Job	Computing nodes
job1	h7, h10
job2	h3, h9
job3	h2, h11
job4	h4, h5
job5	h8, h12
job6	h1, h6

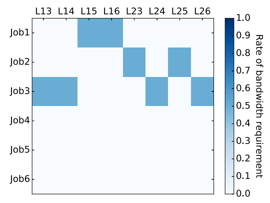
been two possible network paths for job5: path1 (using L15 and L16), and path2 (using L25 and L26). In this experiment, the path1 was allocated to job5 (Fig. 7(b)). In this situation, because L15 and L16 were already used by job1, and the total bandwidth requirements of job1 and job5 was 1500 Mbps, hot-spots might have occurred on L15 and L16. However, the proposed method succeeded to predict that hot-spots would occur in the case that reallocation of network paths for jobs in execution was not performed, and then avoid hot-spots by migrating network paths for job1 as shown in Fig. 7(b).

In *situation-2*, the network path using L13 and L14 was selected for job6. It is observed that the proposed method succeeded to avoid hot-spots on L13 and L14 by reallocating network paths for job3 (Fig. 7(c)). From the above-described evaluation results, we consider that the proposed method succeeded to avoid hot-spots by reallocating network resources of existing jobs in execution.

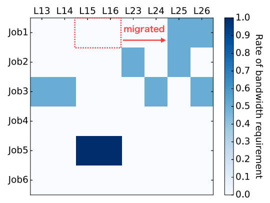
Figure 8 illustrates the result of *exp-2*. The red line represents the network throughput of job6 when the proposed method was adopted. The blue line stands for the network throughput of job6 when the proposed method was not applied. The result clearly shows the difference of these two lines. As described in the evaluation results of *exp-1*, the proposed method succeeded to avoid hot-spots. With the hot-spot avoidances, the proposed method succeeded to keep the network throughput of job6 940 Mbps as the red line. On the other hand, without the proposed method, the SDN-enhanced JMS with the simple method so far failed to avoid hot-spots on L13 and L14 (Fig. 6(b)). Thus the network throughput of job6 was degraded, approximately 450 Mbps for 200 seconds. This degradation of the network throughput delayed the execution time of job6 for about 100 seconds. From these result, we considered the proposed method succeeded to suppress the degradation of job performance.

VI. Related Work

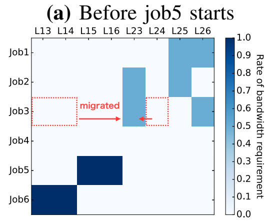
We have focused on the efficient use of the interconnect. Many researches have worked on this research topic. Yu and



(a) Before job5 starts



(b) After job5 starts



(c) After job6 starts

Fig. 7: Network links usage with the proposed method

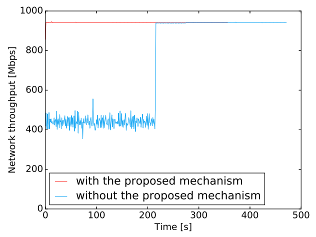


Fig. 8: Network throughput of job6

Deng proposed an OpenFlow-based routing technology on a cluster system with fat-tree interconnect [9]. The purpose of their routing algorithm is to efficiently distribute the traffic of an upcoming network flow, while our research targets resource management including the allocation of source and destination computing nodes.

Cheng and Wang proposed the communication optimization method on a hadoop cluster [10]. Hadoop application has a shuffle phase, on which computing nodes to execute map tasks delivery data to ones to run reduce tasks. The OpenFlow controller on their proposed mechanism gets the amount of the data generated on the phase beforehand. Based on the information of the amount of the data, the controller computes optimized communication routes and configures them to OpenFlow switches. Their mechanism cooperates the computational resource management and the network resource management like our system. Their mechanism only focuses the hadoop system. In contrast, ours aims to control diversified computing requests and applications.

LIGHTNESS [11] is a SDN-based approach for enabling dynamic set up network connectivity of intra/inter cluster systems for Cloud environment. A data center with LIGHTNESS architecture contains a SDN-based control plane, which provides application programing interfaces (APIs) for controlling the network among Top-of-Rack (ToR) switches on the cluster system, or one among cluster systems. Resource management systems can manage these networks by making use of the APIs. LIGHTNESS focuses on the base system for managing the computing and network resources on the cluster system like our research. However, our research also pursues the algorithm to realize the network resource reallocation for efficiently use of the network resources on the cluster system.

VII. Conclusion

In this paper, we designed and implemented an adaptive network resource allocation method with which network resources can be reallocated to jobs in execution, in response to the change of process placement occurred. Our evaluation showed that the proposed method can reallocate network resources of jobs in execution, and suppress the degradation of network throughput of jobs. In the future work, we will assess the effectiveness of the mechanism using a real application on a larger scale cluster system.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP16H02802.

References

[1] W. Yasuhiro *et al.*, “A Job Management Framework Enabling Deployment of Allocation Policy Regarding Computational and Network Resources,” *IEICE TRANSACTIONS on Information and Systems*, vol. J97-D, no. 6, pp. 1082–1093, 2014.

[2] R. Henderson, “Job Scheduling under the Portable Batch System,” *Job Scheduling Strategies for Parallel Processing*, vol. 949, pp. 279–294, 1995.

[3] B. A. Kingsbury, “The Network Queuing System,” Sterling Software, Palo Alto, Tech. Rep., 1986.

[4] *Open Grid Scheduler: The official Open Source Grid Engine*. [Online]. Available: http://gridscheduler.sourceforge.net/.

[5] N. McKeown *et al.*, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks,” [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[7] *Trema: Full-Stack OpenFlow Framework in Ruby and C*. [Online]. Available: http://trema.github.io/trema/.

[8] *Iperf - The TCP/UDP Bandwidth Measurement Tool*. [Online]. Available: https://iperf.fr/.

[9] Y. Li and D. Pan, “OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support,” *IEEE International Conference on Communications 2013*, pp. 2–6, 2013.

[10] L. Cheng and S. Wang, “Application-Aware SDN Routing for Big Data Networking,” pp. 1–6, 2015.

[11] S. Peng *et al.*, “A Novel SDN enabled Hybrid Optical Packet/Circuit Switched Data Centre Network: the LIGHTNESS approach,” *EuCNC 2014 - European Conference on Networks and Communications*, pp. 1–5, 2014.