Highly Reconfigurable Computing Platform for High Performance Computing Infrastructure as a Service: Hi-IaaS

Akihiro Misawa[1], Susumu Date[1], Keichi Takahashi[1], Takashi Yoshikawa[1,2], Masahiko Takahashi[2], Masaki Kan[2], Yasuhiro Watashiba[3,1], Yoshiyuki Kido[1], Chonho Lee[1] and Shinji Shimojo[1]
[1]*Cybermedia Center, Osaka University, 5-1 Mihogaoka, 567-0047, Ibaraki, Osaka, Japan*
[2]*System Platform Research Laboratories, NEC, 1753 Shimonumabe, Nakahara, 211-8666, Kawasaki, Kanagawa, Japan*
[3]*Information of Science, Nara Institute of Science and Technology, 8916-5, Takayama, 630-0192, Ikoma, Nara, Japan*

Keywords: Cloud Computing, Disaggregation, Resource Pool, GPU/FPGA Accelerator, Hetero Computer, Distributed Storage, Job Scheduling, Resource Management, PCI Express, Openstack, Software Defined System.

Abstract: It has become increasingly difficult for high performance computing (HPC) users to own a HPC platform for themselves. As user needs and requirements for HPC have diversified, the HPC systems have the capacity and ability to execute diverse applications. In this paper, we present computer architecture for dynamically and promptly delivering high performance computing infrastructure as a cloud computing service in response to users' requests for the underlying computational resources of the cloud. To obtain the flexibility to accommodate a variety of HPC jobs, each of which may require a unique computing platform, the proposed system reconfigures software and hardware platforms, taking advantage of the synergy of Open Grid Scheduler/Grid Engine and OpenStack. An experimental system developed in this research shows a high degree of flexibility in hardware reconfigurability as well as high performance for a benchmark application of Spark. Also, our evaluation shows that the experimental system can execute twice as many as jobs that need a graphics processing unit (GPU), in addition to eliminating the worst case of resource congestion in the real-world operational record of our university's computer center in the previous half a year.

1 INTRODUCTION

High performance computing (HPC) systems have increasingly been used for IoT/BigData and Artificial Intelligence (AI) besides being used for conventional scientific simulations. Future HPC systems will be required to perform a wider variety of functions and deliver higher performance than today. However, it is becoming more difficult for HPC users to own such a HPC platform for themselves because HPC systems tend to be application-specific, unique, and peculiar, and as a result, come in a wide variety. For example, a server of a HPC system for graphics processing unit (GPU) computing must have a high electrical power supply, big slot space, and a cooling mechanism to install a high power GPU accelerator. Taking a cluster computing system using a message passing interface (MPI) into consideration, a high-speed and expensive host bus adapter (HBA) of Infiniband may be installed. As a result, these HPC systems are far more expensive and also consume a lot more electrical power than conventional computing systems. Thus, HPC users usually cannot afford to own those varieties of HPC systems for themselves.

For such HPC users, advanced cloud services have recently become a way of obtaining HPC resources, such as a GPU computing platform and a distributed cluster system using an MPI (Docs.aws.amazon.com, 2017; GitHub, 2017; Sanders, 2017). However, each system configuration of these HPC systems is unique and rigid in terms of using specific devices, network topologies, and protocols as shown in Figure 1. For this reason, the computing resources for the HPC systems in a cloud must be reserved for only the usage for which they are configured and cannot be flexibly used for other usages. For example, a server node installed with two GPUs is always a two-GPU machine even if other users require only one GPU for each server node. A cluster system composed of multiple server nodes must be deployed on network carefully designed in advance to provide the highest performance per

Misawa, A., Date, S., Takahashi, K., Yoshikawa, T., Takahashi, M., Kan, M., Watashiba, Y., Kido, Y., Lee, C. and Shimojo, S.
Highly Reconfigurable Computing Platform for High Performance Computing Infrastructure as a Service: Hi-IaaS.
DOI: 10.5220/0006302501630174
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 135-146
ISBN: 978-989-758-243-1

system. For example, an MPI application may necessitate a mesh or fat tree topology while a Hadoop/Spark application may necessitate a map and reduce topology.

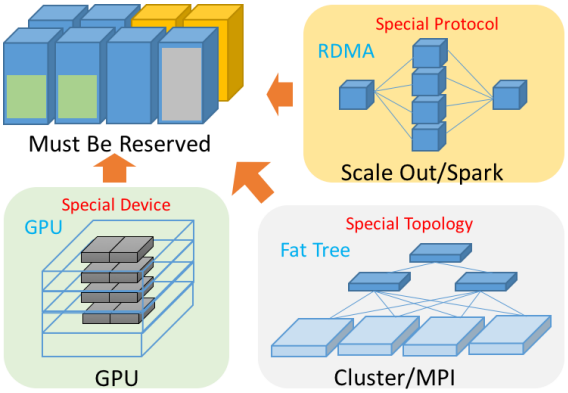

Figure 1: HPC needs special configuration.

This rigidness causes problems for both the user and cloud service provider. From the users' point of view, users cannot obtain the most appropriate underlying computer resources of the cloud for their applications because the computer system must be chosen from a fixed configuration menu. This menu may not offer the optimal configuration for their use case because the menu items may be under or over the expected or predicted specifications. In addition, the resources a user wants to use might be being used for another user's job. This often happens because clouds contain fewer HPC resources than they do conventional servers for general cloud services.

From the cloud service provider's point of view, the provider must prepare and reserve many different HPC platforms to satisfy diversified HPC user needs. This leads the job duration for each platform to become short. As a result, the utility rate of the HPC system may be lower than 30%, even though HPC resources are expensive to deploy.

Our goal is to provide HPC resources from a cloud for various user applications on diverse domain specific computing platforms. We call the concept High performance computing Infrastructure as a Service (Hi-IaaS). Providing HPC resources as a cloud service encounters the abovementioned problems caused by the rigidness of the HPC system. Therefore, a system for Hi-IaaS needs the flexibility to cover diversified HPC applications and different platforms for each application.

From the system point of view, Hi-IaaS must cover recently emerged applications, such as IoT/BigData and AI in addition to scientific simulations that have been major applications of HPC. In other words, Hi-IaaS must offer both high performance data analytics (HPDA) and HPC in a cloud. The key to covering many applications and their application specific computing platforms is reconfigurability in both software and hardware.

In this paper, we present the architecture of Hi-IaaS and describe key technologies mainly focusing on the hardware reconfigurability. A part of the Hi-IaaS system including the essential function of the reconfigurable hardware platform and job management system has been developed to investigate the whole system operation of Hi-IaaS from a user's job submission, through hardware reconfiguration, to job execution.

2 CONCEPT AND ARCHITECTURE OF Hi-IaaS

Hi-IaaS provides computing platforms for diversified HPDA and HPC applications as a cloud service. This section describes the concept and platform architecture of Hi-IaaS.

2.1 Architecture

From the background mentioned in section 1, the requirements for Hi-IaaS system are summarized as follows.

A) Covering diversifying high performance applications.
B) Providing optimal hardware for each user's job.
C) Minimizing users' job waiting time.
D) Minimizing hardware and software cost.
E) Increasing resource utility rate.

To execute various applications while enabling each application to enjoy high performance, the system must be equipped with various hardware devices including graphics processing unit (GPU)/ field programmable gate array (FPGA) accelerators, high speed storage, and an interconnect network. In addition, from the cloud service provider's point of view, the computing resources must be provided quickly with less waiting time. The simplest solution is preparing a sufficient quantity of all the hardware that can afford to provide the necessary performance for each application to avoid resource conflicts among users. However, this solution leads to high initial cost and operation cost including electrical power consumption. In addition, the resource utility rate tends to still be low, and hence, the solution does not satisfy requirement D) or E).

2.2 Reconfigurable Hardware

The solution we present here is based on a highly reconfigurable hardware with a software defined management method. The whole system architecture is shown in Figure 2. The fineness of the reconfigurability is a hardware device level. This means that, for example, if a user wants to execute a type of deep learning job, the user can specify as many GPUs as necessary for the job. In addition, when the job is completed, the GPU must be detached and then allocated for other jobs. This hardware device level reconfigurability is the key to cover various applications with high performance by an optimal hardware reconfiguration for each application. In addition, a software defined management capability can make the reconfiguring process executed dynamically along with each user's job. It can save hardware resources for high performance applications by sharing them among different user jobs and thus can minimize hardware cost and increase the resource utility rate.

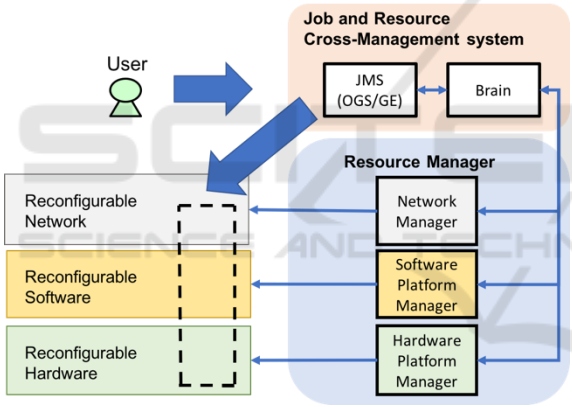

Figure 2: Proposed system architecture.

2.3 Reconfigurable Software

In addition to the highly reconfigurable hardware, a reconfigurable software platform is also necessary to provide a high performance computing platform from a cloud. This is because most recent high performance applications such as high performance data analytics are executed on a distributed system software framework such as Hadoop and Spark (Hadoop.apache.org, 2017; Spark.apache.org, 2017). By using these frameworks, users can utilize a scaling-out method to increase the throughput of the processing. In those frameworks, data storage should also be a reconfigurable distributed system. Thus, the computing platform needs to support such reconfigurable and distributed software frameworks.

When the user's job utilizes GPU/FPGA accelerators, software reconfigurability is also necessary. By using accelerators, the computing system becomes a hetero system equipped with a heterogeneous processing unit. Thus, the system needs a mechanism for assigning a set of processors and accelerators to execute the user's job from multiple central processing units (CPUs) and accelerators in the system. This function is important and essential for the reconfigurable hardware because the number of processors and accelerators might change after the reconfiguration sequence.

By utilizing the reconfigurable software platform, the proposed system needs to provide the optimal computing resources of not only hardware but also a software framework co-operating with the reconfigurable hardware.

2.4 Job and Resource Cross-Management

If the HPC resources are allocated in accordance with each user's job execution dynamically and then reassigned to another job after the job is completed, the number of hardware can be decreased and the resource utility rate can be increased. In turn, this will decrease conflicts between users demanding the same HPC hardware resources and thus decrease the job waiting time caused by resource congestion.

For this purpose, the job management system and the resource manager of the reconfigurable hardware work together as a job and resource cross-management system (JRMS). When a user's job is submitted, the JRMS enqueues the job and makes a list of necessary hardware and software frameworks as a resource recipe. Then the resource manager reconfigures hardware and software to make a computer platform for the job in accordance with the resource recipe. After that, the scheduler gives the dynamically configured computer platform to the job.

2.5 Features of Proposed System

The three functional blocks mentioned above (reconfigurable hardware, reconfigurable software, and a job resource cross-management system) make our proposed system exhibit three features.

A) A computing platform is reconfigured dynamically.
B) Application specific software framework can be used.
C) Hardware resources are shared among different jobs.

These three features can satisfy the five requirements in subsection 2.1.

Note that the platform must be implemented on the basis of open standard hardware and software because utilizing open source resources effectively reduces the capital cost and operational cost for a big system like a cloud.

3 DYNAMIC RECONFIGURABLE HARDWARE

The Hi-IaaS system proposed in section 2 must provide optimal HPC resources in accordance with the user request. For this purpose, the elemental hardware configuration of the proposed system such as the number of servers, network connection, and data storage must be able to be changed even after system deployment. In addition, hardware devices composing each platform (such as a GPU accelerator, solid state drive (SSD) storage, and HBA of InfiniBand) must be able to be easily attached/detached by the resource manager.

The following subsection describes the three key components for achieving such hardware-level high reconfigurability in detail.

3.1 Resource Pool of Disaggregated Computer Platform

The first key component is a reconfigurable hardware resource pool. One prospective way of making a reconfigurable hardware platform is a resource pool system (or disaggregated computer system) (Han et al., 2013; Open Compute 2017; Presentations.interop.com, 2017). It has pools of hardware devices such as a compute pool, a GPU/FPGA accelerator pool, a storage pool, and a network interface pool as shown in Figure 3.

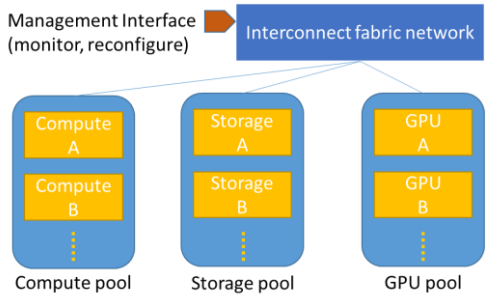

Figure 3: Concept of resource pool system.

For such a disaggregated system, interconnect technology is the most important because it is the very point of disaggregation. To attach hardware devices such as accelerators, storages, and network interfaces at the same point of computer architecture, Peripheral Component Interconnect (PCI) Express is the most appropriate interconnect because almost all the devices have the PCI Express interface. PCI Express is the most common open standard of an I/O interface today.

However, PCI Express is a bus technology with a single root tree topology that is implemented by a PCI Express switch chip. This means only a single compute can exist in a fabric network based on PCI Express. In addition, the number of devices that can be connected to a PCI Express bus is mostly limited to fewer than 10 because of the scale of the switch chip. The link distance is also limited because cascading the PCI Express switch makes a complex and deep forest of PCI bridges in which a conventional basic input/output system (BIOS) cannot complete all the discovery and enumeration processes of bus and endpoint devices.

To use PCI Express as an interconnect fabric of a Hi-IaaS system, Express Ethernet (ExpEther) technology has been adopted (Suzuki et al., 2006) in the proposed Hi-IaaS system. ExpEther is a PCI Express switch over an Ethernet as shown in Figure 4. It can make multiple single-hop PCI Express switches over an Ethernet network. ExpEther has five advantages in terms of the interconnect fabric of highly reconfigurable disaggregated computer hardware (Yoshikawa et al., 2014).

A) Almost no limit to the number of computes and devices.
B) Almost no limit to the connection distance.
C) Logically equivalent to a single-hop PCI Express switch.
D) PCI Express tree can be software defined (reconfigurable).
E) Comparable performance to local device.

Features A)-C) are achieved by the distributed PCI Express switch architecture of ExpEther (Suzuki et al., 2006).

ExpEther makes a single hop PCI Express switch even if the Ethernet network is composed of multiple switches. By using this architecture, we can put as many computes and devices as necessary in a single Ethernet network without limits to the connection distance.

In addition, an ExpEther chip has a group ID. PCI configuration is automatically executed among ExpEther chips that have the same group ID. That is,

ExpEther chips with the same group ID connected through the Ethernet are logically equivalent to a PCI Express switch. Note that the Ethernet is transparent to OS/software. From the OS/software, all the devices are recognized as local devices as if they were in a local chassis.

Therefore, by controlling group ID, the computer hardware can be reconfigured and can be software defined by using the resource manager as mentioned in feature D). All the functions of ExpEther are implemented in a hardware chip, and thus, the latency of the chip is less than 1 µs. The device can perform comparably to those installed in the local slot inside the chassis of a computer as mentioned in feature E).

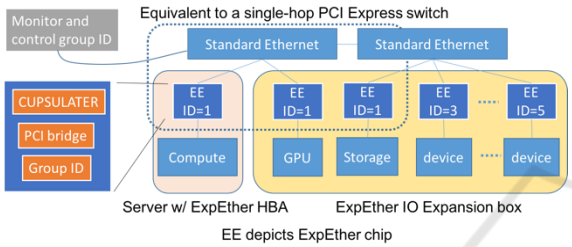

Figure 4: Dynamic attachment/detachment of GPU.

Considering these features, we adopt ExpEther as an interconnect fabric of a reconfigurable computer hardware platform. To make the resource pool, computes can be a conventional server, and devices can be conventional PCI Express devices without any modification. In addition, software including OS and device drivers does not need to be modified.

The simplest implementation is using a pair of an ExpEther HBA card and an IO Expansion box connected directly or via an Ethernet switch. The IO Expansion box contains multiple PCI Express slots with an ExpEther chip on the motherboard. Note that the ExpEther chip has congestion control and a retry mechanism. Therefore, the Ethernet switch can be a standard one without supporting Converged Ethernet specifications.

3.2 Resource Manager

The second key component is the resource manager. By using ExpEther as an interconnect fabric, we can put multiple roots (computes) and endpoint devices in a single network to make a resource pool system. ExpEther chips that have the same group ID make a PCI Express tree automatically. The group ID of an ExpEther chip can be set remotely by sending a control packet from the resource manager through the Ethernet that connects all the computes and devices. As described in the previous section, The Ethernet is transparent to the OS/software. From the OS/software, all the devices are recognized as local devices as if they were in the chassis. Therefore, changing group ID corresponds to inserting/removing PCI Express devices to/from a local PCI Express slot physically. By using a PCI Express device that supports a PCI compliant hot-plug process, the proposed system can become reconfigurable at the hardware device-level. That is, we can attach/detach any devices from a resource pool in accordance with the user's request.

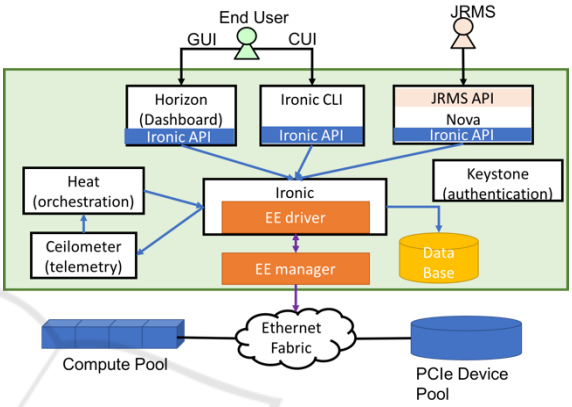

Figure 5: OpenStack-based resource manager.

The resource manager basically monitors all the computes, devices, and the PCI Express-level connections, that is, the PCI Express tree. To configure the hardware, the resource manager sets group IDs of ExpEther chips with computes and devices that should be connected in accordance with the hardware reconfiguration specified by the job resource cross-management system (JRMS) described below.

We have developed a resource manager on the basis of the OpenStack framework (OpenStack, 2017) as shown in Figure 5. The current version of OpenStack (MITAKA) does not have a management method for disaggregated computers and devices. To include such low layer level management in the OpenStack framework, we have modified Ironic because it offers a function for bare metal (physical) server control although a major control object of the OpenStack is virtual machines. In addition, Ironic has an advantage because it is already equipped with (re)boot control, which is required to attach/detach PCI Express devices.

The monitor and controller function for the ExpEther system is implemented as an EE manager. It is a simple module that monitors and sends a control packet of ExpEther. The monitored information is recorded in a database through Ironic.

When the system is reconfigured, a control packet including a new group ID for the ExpEther chip with a specified compute/device is sent through the Ethernet. The management methods specific to ExpEther are implemented as an EE driver in Ironic. This segregated implementation of the management methods enables the modified Ironic to cover other disaggregated computer systems that use interconnect fabric other than ExpEther.

The resource manager has a control application program interface (API) for the JRMS and a graphical user interface (GUI) and command-line interface (CLI) for a user's direct control. The Horizon GUI is expanded to include a device-level monitor and a control interface. Other mechanisms (such as Heat: Orchestration, Ceilometer: telemetry, Nova: scheduler, Keystone: authentication) have been implemented together without modification from an original OpenStack framework.

3.3 Job Resource Cross Management System

The third key component is job and resource cross-management system (JRMS). To allocate HPC resources in accordance with the requested specification dynamically with the job execution, a JRMS has been developed. It consists of two major functional blocks: a job management system (JMS) based on Open Grid Scheduler (Gridscheduler.sourceforge.net, 2017), and a policy based resource assignment controller (Brain) with an interface between the resource manager mentioned above and also a software-defined networking (SDN) controller as shown in Figure 6. The operational flow is described as follows.

1) User submits a job with resource request.
2) JMS makes a list of candidate resources.
3) Brain receives H/W resource status
4) Brain asks the resource manager to configure resources
5) Resource manager announces the completion of reconfiguration
6) Brain announces resource address for job scheduler
7) JMS executes the job in the queue

JRMS plays three major roles. It receives a user's job request and then schedules it in the queue. At the same time, it picks out some lists of resources that satisfy the user's request from a resource pool including a network route for an MPI when the job is specified to be executed on a cluster system. Brain receives the set of candidate resources and then chooses and modifies the received list by considering the current usage of the resources and link/node utilization. Then the job is set in the execution queue with the resource recipe. When the job's turn comes, JRMS tells the resource manager to configure the hardware to make a computer platform in accordance with the resource recipe. After a certain amount of the system waiting time, which can be set as a parameter, the job is executed. Although it is out of the scope of this paper, JRMS can control networks and software.

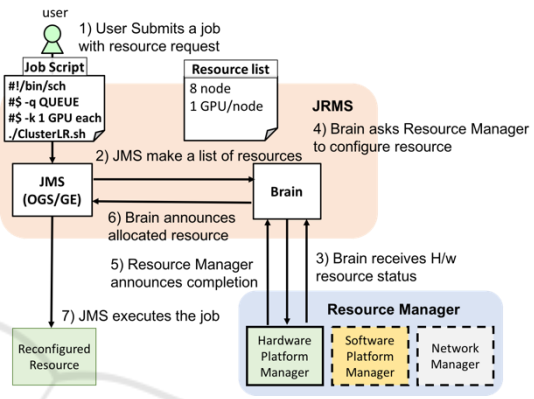

Figure 6: Job and Resource Cross Management System (JRMS).

4 EXPERIMENTAL SYSTEM

4.1 Implementation

We have built an experimental system to see whether the proposed architecture can be implemented into a real system, i.e., to see whether the computer hardware can be dynamically reconfigured with each job execution.

The experimental system is composed of the resource pool, resource manager, and job and resource cross-management system as shown in Figure 7.

The resource pool has two computes, both of which are commercially available compact workstations (NEC Express 5800 52Xa: CPU: E3-1200v3, 16GB DDR3-1600 SDRAM). The PCI Express slot of the ExpEther IO expansion unit contains IO devices, which are two commercially available GPUs (NVIDIA K-5000) and a non-volatile memory express standard (NVMe) card. The NVMe storage card is a laboratory-level prototype. It supports NVMe 1.1 specification including single-root input/output virtualization (SR-IOV). With ExpEther, an SR-IOV device can be shared among multiple computes at the PCI Express level. This

means that the NVMe card can be used by two computes simultaneously whereas the OS in each compute utilizes the storage card as if the storage card were in a local slot. The "exclusive write" operation is implemented by using "compare and write" operation, which is also specified in NVMe 1.1. These resources are connected through ExpEther with two paths of 10G-Ethernet through the standard Ethernet switch (NEC QX-S5828T).

The resource manager and JRMS are installed in a virtual machine (VM) on the same machine by using Virtual Box. The operating system (OS) for the resource manager is CentOS 7.2, and the JRMS is CentOS 6.4. The version of the OpenStack is Kilo. The resource manager is connected to the ExpEther network (Ethernet), and JRMS is connected to each compute via an Internet protocol (IP) network.

In this implementation, a software platform including the OS of compute nodes is static though it can be installed after reconfiguration of the hardware by using a standard Ironic process. Apache Spark, storage software, and an accelerator runtime manager were installed on the computes in advance. These reconfigurable software parts are out of the scope of this paper and will be described in the future work to implement the mechanism for delivering these software platforms dynamically.

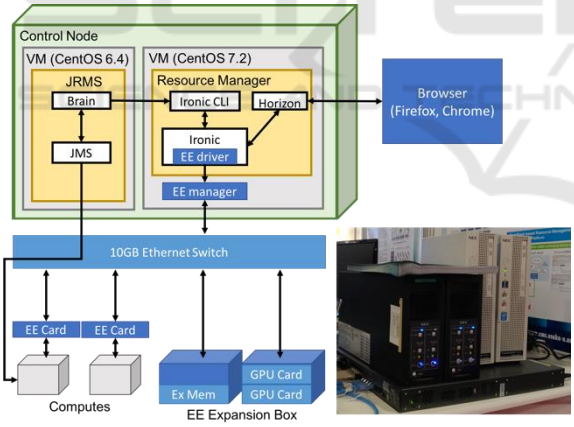

Figure 7: Experimental system.

4.2 Operation

Figure 8 shows an example scenario to explain the control sequence. First, a job request is submitted to the JRMS. It contains a user's requirements for the computing platform. In this example, it indicates two nodes both installed with a single GPU (K-5000). Both nodes share the same external NVMe storage, though it is a local storage from the software view of each node.

The JRMS puts the job into the queue. Then it makes a list of the resources that can be used to satisfy the request. The list is sent to Brain, which has a database of the resources including their current status. In this experimental system, two computes, two GPUs, and an NVMe storage card are in the resource pool. On the nodes, the software platform is pre-installed and networks among nodes are fixed because we want to focus on the hardware reconfigurability in this example scenario. In addition, the group IDs of ExpEther are assigned as #1 and #2 for each node in advance. The nodes share an external NVMe storage card in the pool by using the software storage engine we have developed.

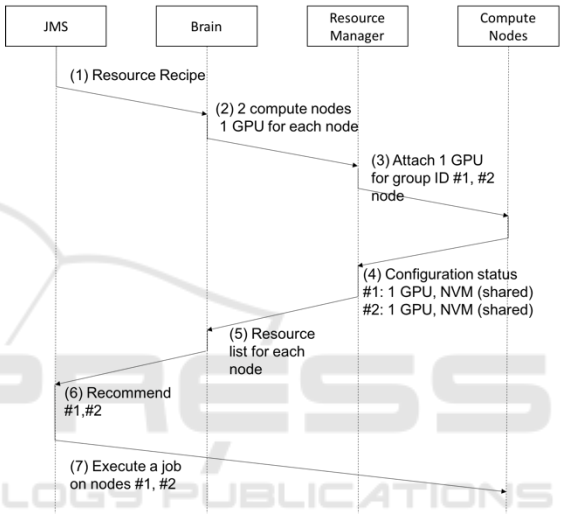

Figure 8: Sequence of job execution.

Brain always gathers the information on resources and puts it in the database. When the resource list arrives, by referring to the database, Brain determines the devices (GPU in this scenario) for each compute node.

Then the resource manager is indicated to change the configuration to attach a GPU to each node at the PCI Express level. To manage the PCI Express connection, the group ID of each ExpEther chip connected to the compute and IO device is recorded in the device list. The group ID of the unused devices in the pool is set to Group ID = #0. The group ID can be chosen from #1 to #4096. In this scenario, the group IDs of #1 and #2 are assigned for each GPU. Then the GPUs are attached to each node and hot-plugged in the OS of each node. We set the waiting time for completion of the hot-plug process in OS to 20 sec. This is longer than the necessary time for the hot-plug process, but it differs and deviates depending on the computing environment and the conditions. It should be shortened by investigating the

hot-plug process in future work. In the end, the job is executed on a two-node cluster in which both nodes are installed with a GPU in accordance with the user's request.

The hot plug process of the GPUs can be seen by the horizon GUI of OpenStack framework as shown in Figure 9. The GUI is modified to show the PCI Express tree just like "lspci –t" command in LINUX. It is the server-view that shows the PCI Express connections from a compute node to endpoint devices.

The device tree before the job execution is shown in the left part of Figure 9. Two PCI Express trees with Group IDs of #1 and #2 are shown here. There are NVMe storage cards in both nodes. These devices are the same device simultaneously shared. The red mark of the storage device indicates the device is shared among other nodes. Those nodes can be seen by clicking the red mark.

Before the job execution, no GPU was seen in either PCI Express tree because both are in the resource pool with a group ID set as #0. When the job is executed with a request for using GPUs, the resource manager connects them to each server by setting group IDs of each GPU as #1 and #2. Then in the right part of Figure 9, GPUs are seen in both PCI Express trees. After that, the job is automatically executed. We found that the JRMS can execute the expected procedure including dynamically reconfiguring a computer hardware platform.

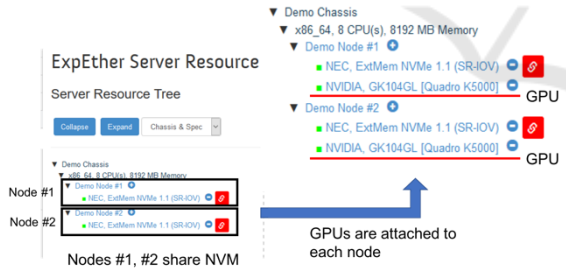

Figure 9: Hot plug process on OpenStack management window.

4.3 Performance

Using the experimental setup described in the previous section, we also try to see whether the system can work as a HPC system.

The main concern about the performance is the latency of the interconnection to form a resource pool, that is, the latency of the ExpEther chip, cable, and switch. The latency is about 700 ns for the ExpEther chip and 5 ns/m for the cable. A single hop of a switch is about 500 ns, which means a roundtrip takes about 3 µs.

Performance degradation by the latency highly depends on the application. Amano's group already evaluated it in detail by investigating the performance scale-up in accordance with the number of GPUs (Nomura et al., 2014; Mitsuishi et al., 2016). In addition, this work focuses on the HPC use in a cloud. Most cloud users understand the computer resources in a cloud perform worse than those specially designed on premises for high cost. Therefore, in this study, we have investigated whether the Spark benchmark software, logistic regression, is accelerated by GPUs connected from the resource pool.

Because the IO performance degradation of the ExpEther is not significant, we were able to enjoy acceleration by GPUs while one Spark benchmark, logistic regression, was performed. We could not compare the performance with the system in which the GPUs are directly mounted on the computer because the computer we used in this experiment is too compact to install GPUs inside.

On the other hand, we can conclude that even if the cloud service provider deploys a conventional 1-U machine that cannot install GPUs, it can provide GPU computing by using this reconfigurable mechanism. This increases the probability to obtain the computer platform from a cloud that perfectly matches one's request.

4.4 Reconfiguring Time

Another important performance to consider is how much time it takes to reconfigure the system. So far, we do not have any technology to shorten the OS boot time even though it takes several tens of seconds or more. It is much longer than any other process in reconfiguration. Therefore, if the reconfiguration process needs a reboot, reconfiguration lasts almost as long as the reboot of the OS. This happens in the case of, for example, attaching an un-hot-pluggable device, adding a special driver for a device that the OS does not support, allocating memory, etc.

Without a reboot, changing PCI Express configuration by ExpEther only takes a few µsec at the hardware level because it can be executed by just sending a control packet for changing group ID of the ExpEther chip with the target device. However, the device must be recognized by the OS in which a hot-plug handler coordinates the attach/detach process before the hot-plugged device can be utilized by software. It takes about 10 sec or more depending on the device and status of the compute. Therefore, using

this experimental setup, we set the waiting time to 20 sec for hardware reconfiguration. This waiting time dominates the reconfiguration time in non-reboot cases, so far. Further work should be done to evaluate the hot-plug time precisely and to minimize reconfiguring time to make this system faster for the hardware reconfiguration.

5 EVALUATION IN CLOUD USECASE

Our organization, the Cybermedia Center, provides computer resources to researchers and students of Osaka University as well as other universities. It has three major computing systems: a vector super computer (SXAce), a scalar cluster computer (HCC), and a scalar cluster computer with GPU (VCC) (Hpc.cmc.osaka-u.ac.jp, 2017).

The VCC consists of 65 servers with IO expansion capability using ExpEther. However, in the current operation, the configuration is not changed dynamically. The reconfiguration is done only every half a year by surveying users' plans for the computing platform utilization; for example, monthly use of 4 nodes with 2 GPU machines for 200 hours, 16 nodes with PCI-SSD for 100 hours, etc. In accordance with the users' plans, the computer platform configuration for the next half a year is determined, that is, the system configuration is rigid for half a year.

We evaluated how the utility rate and job waiting time can be improved by adopting the proposed dynamic reconfiguration.

5.1 Resource Utilization Simulation

First, the resource utility rate is estimated for a dynamic reconfigurable hardware platform by using that of a statically configured computer system as a reference. By using a job scheduling simulator, ALEA (Klusácek and Rudová, 2010), we have investigated how the utility rate changes in accordance with a various types of hardware configuration and job streams.

ALEA can deal with common problems of job scheduling in clusters and grids, like heterogeneity of jobs and resources and dynamic runtime changes, and provide a handful of features including a large set of various scheduling algorithms, several standard workload parsers, and a set of typical fairness-related job ordering policies.

The job and resource scheduling algorithm is out of the scope of this paper because determining the best algorithm is too elaborate. This is because this system is so flexible that the simulation conditions to consider are very diverse.

Therefore, we fixed some conditions to simplify the simulation and roughly find out the dependency of the utility rate on the hardware reconfiguration and job workload. The simulated system is a GPU cluster system with 64 compute nodes and 64 GPUs. The simulation conditions are as follows.

A) Scheduling algorithm is FIFO.
B) Reconfiguration is applied only for the number of nodes and GPUs.
C) All workload is the same as fixed execution time.
D) A node can accept only a single job at a time.
E) Number of nodes a job requests is fixed to 8 or 16.

The utility rate of the GPU is investigated for the computer hardware reconfiguration of static and reconfigurable cases. For the static configuration, we investigated the five different configurations in Table 1.

Table 1: Cluster configurations.

Cluster set	Number of GPU / Node set		
#1	1 / 64	-	-
#2	2 / 8	1 / 48	0 / 8
#3	2 / 16	1 / 32	0 / 16
#4	4 / 8	2 / 16	0 / 40
#5	4 / 10	2 / 12	0 / 42

For all the cluster configuration sets, the average GPU utility is plotted in Figure 10. The average utility rate for the reconfigurable hardware is also plotted in the figure as a dotted line because it is independent from the static cluster configuration. For #1, #2, and #3, the numbers of GPUs for nodes are well balanced to execute jobs with the node and GPU number randomized from 0 to the minimum GPU number of the cluster set, that is, 1 for cluster sets #1 to #3 and 2 for cluster sets #4 and #5. However, because the total number of GPUs is limited to 64, for #4 and #5 that include four-GPU machines, the number of nodes without GPUs becomes dominant. Thus, more jobs that request GPUs have to wait for GPU nodes to be released by the currently executed job when the job finishes.

The utility rate of GPU for the configuration of cluster set #5 is shown in Figure 10 as an example. It varies over time when a job cannot be assigned its requested computer resources in terms of the number

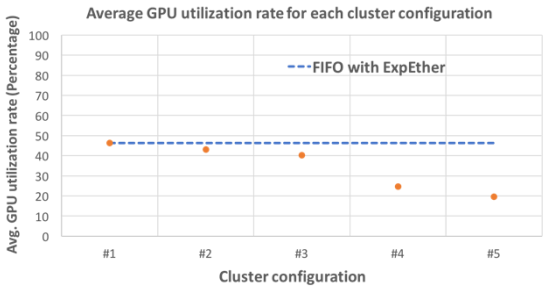

Figure 10: Utility rate depending on cluster configuration.

of nodes or GPUs. The job stays in a queue waiting for finished jobs to release the resources. This happens even if some resources are available. For example, if the job requests four-GPU nodes, it cannot be executed even if two-GPU nodes are available. Then the GPUs on the two-GPU nodes are left unused until a job with two-GPU nodes is submitted. By using a reconfigurable hardware platform, unused GPUs are put back into the resource pool, then four-GPU nodes are configured by using them, and thus, a four-GPU job can be executed without waiting time. This results in decreasing the time for executing all jobs by about 42% that of the rigid system, in addition to increasing the average GPU utility rate from 20% to 47 % in the example use case shown in Figure 11.

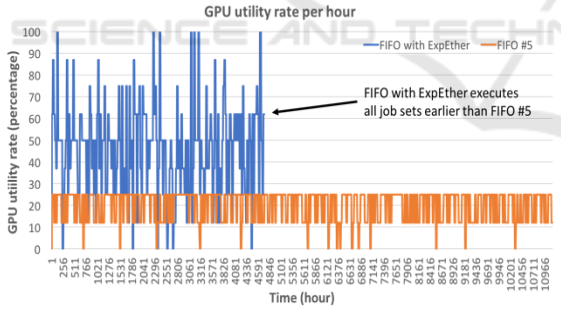

Figure 11: Utility rate of GPU of cluster set #5.

5.2 Job Waiting Time Estimation with Real Operation

We investigated the real utilization record of the VCC system to see whether the Hi-IaaS can increase resource utility. Currently, the cluster configuration is fixed in this half a year as shown in Table 2.

With this static configuration, some users' jobs were congested. Figure 12 shows the worst case in the months of 2Q-3Q in 2016. The figure only shows nodes #19-#23 of 65 to simplify the explanation. On Sep. 28, the blue job had been executed on nodes #22

Table 2: Current cluster configurations of VCC.

Node Number #	GPUs per Node
#0 to #4	4
#5 to #10	3
#11 to #21	2
#22 to #64	0

and #23. Then at 15:00, the yellow job was enqueued. Although the yellow job did not request GPUs, it was executed on the GPU machine (nodes #19, #20, #21) because other nodes (#22, #23) were occupied. At 20:00, a purple job that requested two nodes with a GPU was enqueued. However, all the GPU machines (nodes #19, #20, #21) were occupied at that time, and thus, the job had to be kept waiting. At 20:00 on Sep. 29, the yellow job was completed. Then the purple GPU job was executed. The waiting time for the purple job was 25.1 hours.

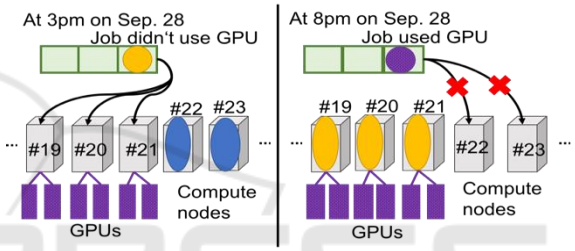

Figure 12: Stacked jobs can be executed by using this platform.

In this case, there were compute nodes unoccupied (nodes #22, #23) when the purple job with a GPU request was enqueued. At the same time, GPUs on nodes #19-#21 were not utilized. Therefore, if we can dynamically reconfigure the system by putting unused GPUs on nodes #19-#21 back in the resource pool and then attach them to the free compute nodes (nodes #22, #23), the purple job will be able to be executed without experiencing such a long waiting time.

This is just an example case, but we expect that further development will lead us to make the reconfiguration dynamically and automatically without a system engineer monitoring the job operation 24 hour x 7 days, which is required under the current operation.

6 RELATED WORK

This work succeeded the research on disaggregated computing platform using ExpEther (Yoshikawa et al., 2014; Suzuki et al., 2016), although the previous work focused on scale up ability and a hardware

mechanism for simultaneously sharing devices among multiple servers. On the other hand, this paper focuses on reconfigurability to be used as a HPC platform in a cloud.

In terms of disaggregated computing systems, Intel Rack Scale Architecture and OpenComputeProject (Presentations.interop.com, 2017; Open Compute, 2017) are attracting growing interest in computer industries. However, the disaggregated resources are distributed within a rack. Han et al. (2013) considered data center scale disaggregation mainly focusing on performance degradation in memory disaggregation caused by an interconnect network. Our work also has been aiming at data center scale disaggregation. The difference is we realize it by pure open standard interface PCI Express and Ethernet. For the performance consideration, Amano's team has investigated performance in detail and succeeded in scaling performance along with the number of GPUs (Nomura et al., 2014; Mitsuishi et al., 2016). Katrinis et al. (2016) also published a research plan for cloud data center scale disaggregation but so far it is at a vision level.

In terms of reconfigurability in a cloud, a lot of work has done on the cloud management frameworks including OpenStack (Sefraoui, Aissaoui and Eleuldj, 2014; Xu et al., 2014). The object of reconfiguration is a VM-based system although this work treats the hardware devices comprising servers in order to drastically change the function and performance of the computing platform.

In terms of job resource allocation, Lee, Chun and Katz (2011), worked on resource allocation and scheduling together, the job of which is executed in a hetero computing platform in a cloud. Because the work used a current cloud service as it is, the resource allocation was performed by using instances on the service menu. There is a possibility that the scheduling algorism can be applied for our reconfigurable hardware platform with low-level hardware resource allocation resulting in more dynamic control for job execution.

In terms of HPC as a service, Wheeler et al. (2012) made a framework to dispatch a user's job over different HPC system including BlueGene

7 CONCLUSIONS

We presented the concept of High performance computing Infrastructure as a Service (Hi-IaaS) and a system that realizes it. The system consists of job and resource cross management software with reconfigurable hardware that can make computer hardware from the resource pool by attaching/detaching computers and devices at the Peripheral Component Interconnect Express (PCI Express) level in accordance with a user's request. It is also equipped with a middleware/software platform for high performance data analysis that is increasingly being used as high performance computing (HPC).

In this paper, we focused on the reconfigurable hardware of Hi-IaaS and developed a small experimental setup of the proposed system with two computes, two graphics processing units (GPUs), and a shared non-volatile memory express standard (NVMe) storage card. One piece of Spark benchmark software (logistic regression) was executed to investigate whether the proposed dynamic reconfiguration can be performed while enjoying high performance computing by GPU acceleration at the same time.

In addition, the simulation results showed the effectiveness of reconfigurable platform for the resource utility rate increased from 20% to 47% and job execution time reduced by 42% in a 64-node system. Finally, we found our system can eliminate the 25-hour waiting time recorded as the worst case in the half-year real job operational record of our university's computing center.

8 FUTURE WORK

We have been further investigating the effectiveness of this system in terms of various job workloads and system configurations. Next, we will investigate suitable algorithms for the reconfiguration that fit these job workloads and system configuration variations. We will also try to investigate the dynamic reconfiguration process and performance in a real-world big system by using a VCC system when it can be utilized for experimental usage that does not conflict with ordinary HPC services.

ACKNOWLEDGEMENT

The authors thank Takuya Yamada and Masaharu Shimizu of Osaka University for their helpful discussion, Professor Jason Cong of UCLA, Peichen Pan and Allan Wu of Falcon Computing Solution Inc. for providing their accelerator runtime management software, Ryota Hayasaka, Toru Sasagawa, Takaaki Noda, Mineko Marugami, and Yasuhiro Dairaku for developing experimental system, Takashi Takenaka,

Takeo Hosomi, and Yuich Nakamura for their helpful discussion, Yumi Matsumoto and Kazuhisa Shiota of NEC solution innovators for their helpful discussion. This research achievement is partly brought through the use of the supercomputer PC cluster for large-scale visualization (VCC). This work was supported by JSPS KAKENHI Grant Number JP16H02802.

REFERENCES

Docs.aws.amazon.com. (2017). *Linux Accelerated Computing Instances - Amazon Elastic Compute Cloud*. [online] Available at: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/accelerated-computing-instances.html [Accessed 15 Mar. 2017].

GitHub. (2017). *irifed/softlayer-mpicluster*. [online] Available at: https://github.com/irifed/softlayer-mpicluster [Accessed 15 Mar. 2017].

Sanders, C. (2017). *Azure N-Series preview availability*. [online] Azure.microsoft.com. Available at: https://azure. microsoft.com/en-us/blog/azure-n-series-preview-availability/ [Accessed 15 Mar. 2017].

Hadoop.apache.org. (2017). *Welcome to Apache™ Hadoop®!*. [online] Available at: http://hadoop.apache.org/ [Accessed 15 Mar. 2017].

Spark.apache.org. (2017). *Apache Spark™ - Lightning-Fast Cluster Computing*. [online] Available at: http://spark.apache.org/ [Accessed 15 Mar. 2017].

Presentations.interop.com. (2017). *Intel Rack Scale Architecture Overview*. [online] Available at: http://presentations.interop.com/events/las-vegas/ 2013/free-sessions---keynote-presentations/download/463 [Accessed 15 Mar. 2017].

Open Compute. (2017). *Home*. [online] Available at: http://www.opencompute.org/ [Accessed 15 Mar. 2017].

Han, S., Egi, N., Panda, A., Ratnasamy, S., Shi, G. and Shenker, S. (2013). Network support for resource disaggregation in next-generation datacenters. *Proceedings of the 12th ACM Workshop on Hot Topics in Networks - HotNets-XII*.

Suzuki, J., Hidaka, Y., Higuchi, J., Yoshikawa, T. and Iwata, A. (2006). ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform. *14th IEEE Symposium on High-Performance Interconnects (HOTI'06)*.

Yoshikawa, T., Suzuki, J., Hidaka, Y., Higuchi, J. and Abe, S. (2014). Bridge chip composing a PCIe switch over ethernet to make a seamless disaggregated computer in data-center scale. *2014 IEEE Hot Chips 26 Symposium (HCS)*.

OpenStack. (2017). *Software » OpenStack Open Source Cloud Computing Software*. [online] Available at: https://www.openstack.org/software/ [Accessed 15 Mar. 2017].

Gridscheduler.sourceforge.net. (2017). *Open Grid Scheduler: The official Open Source Grid Engine*. [online] Available at: http://gridscheduler.sourceforge.net/ [Accessed 15 Mar. 2017].

Nomura, S., Mitsuishi, T., Suzuki, J., Hayashi, Y., Kan, M. and Amano, H. (2014). Performance Analysis of the Multi-GPU System with ExpEther. *ACM SIGARCH Computer Architecture News*, 42(4), pp.9-14.

Mitsuishi, T., Suzuki, J., Hayashi, Y., Kan, M. and Amano, H. (2016). Breadth First Search on Cost-efficient Multi-GPU Systems. *ACM SIGARCH Computer Architecture News*, 43(4), pp.58-63.

Hpc.cmc.osaka-u.ac.jp. (2017). *Cybermedia Center, Osaka University*. [online] Available at: http://www.hpc.cmc.osaka-u.ac.jp/en/ [Accessed 15 Mar. 2017].

Klusácek, D. and Rudová, H. (2010). Alea 2: job scheduling simulator. *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 61, pp.1-10.

Suzuki, J., Hidaka, Y., Higuchi, J., Hayashi, Y., Kan, M. and Yoshikawa, T. (2016). Disaggregation and Sharing of I/O Devices in Cloud Data Centers. *IEEE Transactions on Computers*, 65(10), pp.3013-3026.

Katrinis, K., Syrivelis, D., Pnevmatikatos, D., Zervas, G., Theodoropoulos, D., Koutsopoulos, I., Hasharoni, K., Raho, D., Pinto, C., Espina, F., Lopez-Buedo, S., Chen, Q., Nemirovsky, M., Roca, D., Klos, H. and Berends, T. (2016). Rack-scale disaggregated cloud data centers: The dReDBox project vision. *Proceedings of the 20th Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp.690-695.

Sefraoui, O., Aissaoui, M. and Eleuldj, M. (2014). Dynamic Reconfigurable Component for Cloud Computing Resources. *International Journal of Computer Applications*, 88(7), pp.1-5.

Xu, F., Liu, F., Jin, H. and Vasilakos, A. (2014). Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions. *Proceedings of the IEEE*, 102(1), pp.11-31.

Lee, G., Chun, B. and Katz, R. (2011). Heterogeneity-aware resource allocation and scheduling in the cloud. *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'11)*, pp.1-5.

Wheeler, M., Pencheva, G., Tavakoli, R., Shae, Z., Jamjoom, H., Sexton, J., Sachdeva, V., Jordan, K., Kim, H., Parashar, M. and AbdelBaky, M. (2012). Enabling High-Performance Computing as a Service. *Computer*, 45(10), pp.72-80.