J Netw Syst Manage (2018) 26:339–360
https://doi.org/10.1007/s10922-017-9418-z

Server Selection, Configuration and Reconfiguration Technology for IaaS Cloud with Multiple Server Types

Yoji Yamato[1]

Received: 28 April 2016 / Revised: 17 July 2017 / Accepted: 26 July 2017 /
Published online: 3 August 2017

Abstract We propose a server selection, configuration, reconfiguration and automatic performance verification technology to meet user functional and performance requirements on various types of cloud compute servers. Various servers mean there are not only virtual machines on normal CPU servers but also container or baremetal servers on strong graphic processing unit (GPU) servers or field programmable gate arrays (FPGAs) with a configuration that accelerates specified computation. Early cloud systems are composed of many PC-like servers, and virtual machines on these severs use distributed processing technology to achieve high computational performance. However, recent cloud systems change to make the best use of advances in hardware power. It is well known that baremetal and container performances are better than virtual machines performances. And dedicated processing servers, such as strong GPU servers for graphics processing, and FPGA servers for specified computation, have increased. Our objective for this study was to enable cloud providers to provision compute resources on appropriate hardware based on user requirements, so that users can benefit from high performance of their applications easily. Our proposed technology select appropriate servers for user compute resources from various types of hardware, such as GPUs and FPGAs, or set appropriate configurations or reconfigurations of FPGAs to use hardware power. Furthermore, our technology automatically verifies the performances of provisioned systems. We measured provisioning and automatic performance verification times to show the effectiveness of our technology.

Keywords Performance · Cloud computing · IaaS · FPGA · GPU · Baremetal · Container · OpenStack

✉ Yoji Yamato
yamato.yoji@lab.ntt.co.jp

[1] NTT Software Innovation Center, NTT Corporation, 3-9-11 Midori-cho, Musashino-shi 180-8585, Japan

1 Introduction

Infrastructure as a service (IaaS) cloud services have advanced recently [1–9]. Users can use virtual resources, such as virtual servers, virtual networks, virtual routers, on demand from IaaS service providers and can coordinate on-premise servers using service coordination technologies such as [10–13]. Users can install the OS and middleware, such as DBMS, web servers, application servers and mail servers, to virtual servers themselves. Open source IaaS software has also become wide spread, and adoption of OpenStack is increasing. The NTT group has also been launching IaaS services based on OpenStack since 2013.

Early cloud systems are composed of many PC-like servers. Hypervisors, such as Xen [14] or kernel-based virtual machines (KVMs) [15], virtualize these servers and virtual machines (VMs) on hypervisors using distributed processing technology, such as MapReduce [16], to achieve high computational performance.

However, recent cloud systems change to make the best use of recent advances in hardware power. For example, the number of CPU cores of a cloud server has been increasing, and more than 10 cores is normal in cloud servers and the number of CPU cores of PCs remains 2–6. To use a large amount of core CPU power, some providers have started to provide container-based virtual servers (hereinafter, containers) with little performance degradation and baremetal servers (hereinafter, baremetal), which do not virtualize physical servers. Moreover, some cloud providers use special servers with strong graphics processing units (GPUs) to process graphic applications or special servers with field programmable gate arrays (FPGAs) to accelerate specific computation logics. For example, Microsoft's search engine Bing uses FPGAs to optimize search processing [17].

To use the recent advances in hardware power, users can benefit from high performance of their applications. However, to achieve this, users need to design appropriate server configurations and have much technical skills. Therefore, our objective was to enable cloud providers to provide user resources on appropriate hardware based on user functional and performance requirements, so that users can benefit from high performance of their applications. We previously studied technology to select appropriate provisioning types of baremetals, containers, or VMs based on user requests. In this paper study, we investigated technology that selects appropriate servers for user resources from various types of hardware, such as GPUs and FPGAs for unskilled users or sets appropriate configurations or reconfigurations of FPGAs for skilled users to provide high-performance systems.

The rest of this paper is organized as follows. In Sect. 2, we introduce an IaaS platform called OpenStack and review provisioning technology and cloud hardware. In Sect. 3, we explain typical scenarios and clarify current problems. In Sect. 4, we discuss the server selection function of our technology for allocating user resources to appropriate servers and the server reconfiguration function to adapt hardware to user usages. In Sect. 5, we confirm the performances of the proposed technology. We compare our work in Sect. 6 and summarize the paper in Sect. 7.

2 Overview of Cloud Technology

2.1 Outline of OpenStack

OpenStack and Amazon Web Services [18] are major IaaS platforms. Though the basic idea for our proposed technology is independent from IaaS platforms, we measured the performance of our proposal on OpenStack, as discussed in Sect. 5. Thus, we explain OpenStack as an example of an IaaS platform in this subsection. Note that the functions of OpenStack are similar to other IaaS platforms.

OpenStack is composed of function blocks that manage each virtual resource and those that integrate other function blocks. Figure 1 shows OpenStack function blocks. Neutron manages virtual networks. The open virtual switch (OVS) [19] and other software switches can be used as virtual switches. Nova manages compute servers. Hypervisors usages are general, but containers, such as Docker containers and baremetal servers provisioned by Ironic, can also be controllable. OpenStack provides two storage management function blocks; Cinder for block storage and Swift for object storage. Glance manages image files for compute servers. Heat [20] orchestrates these function blocks and provisions multiple resources according to a template text file. Ceilometer is a monitoring function of resource usage. Keystone is a function block that enables single sign-on authentication among other OpenStack function blocks. The functions of OpenStack are used through Representational State Transfer (REST) APIs. There is also a Web GUI called Horizon that uses the functions of OpenStack. To utilize cloud APIs [21] and Web services technologies such as [22–28], users can coordinate other systems easily.

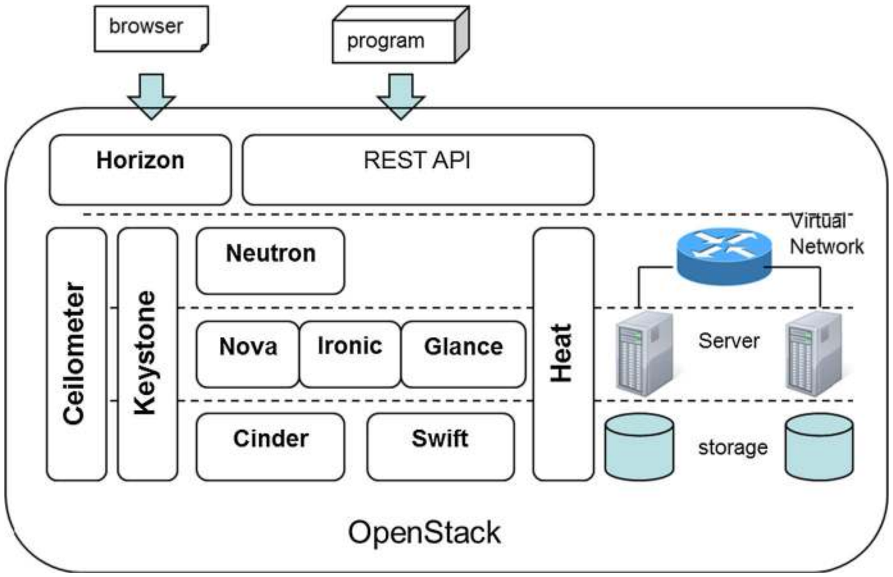


Fig. 1 OpenStack architecture

2.2 Provisioning Type

In this subsection, we explain the provisioning types of cloud compute resources; baremetal, container, and VM.

Baremetal is a non-virtualized physical server. IBM SoftLayer provides baremetal cloud services that have characteristics of prompt provisioning and pay-per-use billing. In OpenStack, the Ironic component provides baremetal provisioning. Because baremetal is a dedicated server, flexibility and performance are high but provisioning and start-up time are long.

A container technology is OS virtualization. OpenVZ or FreeBSD jail have been used for virtual private servers (VPSs) [29] for many years. Compute resources are isolated with each unit called a container but the OS kernel is shared among all containers. Containers do not have kernel flexibility, but container creation only requires a process invocation and a short time for start up. Virtualization overhead is also small.

Hypervisor technology is hardware virtualization, and VMs behave on emulated hardware; thus, users can customize the VM OS flexibly. The major hypervisors are Xen, KVM, and VMware ESX. Virtual machines have flexible OSs and live migrations but exhibit low performances and long start up time.

2.3 Advances in Hardware

In this subsection, we explain the computational units of cloud servers; CPU, GPU and FPGA.

A CPU is the most general computational unit on a server or PC and is designed for low latency. It has a huge cache, advanced control, and a strong arithmetic unit. In cloud servers, multi-core CPUs with more than ten cores are generally used.

A GPU is a computational unit originally for graphics processing and is designed for high throughput. It has a small cache, simple control, and energy-efficient arithmetic unit, and many arithmetic logic units compute in parallel to achieve high throughput. Servers for graphics processing have strong GPUs. Recently, GPU programming, such as the compute unified device architecture (CUDA) [30], that involves GPU computational power not only for graphics processing has become popular.

Furthermore, to program without walls between the CPU and GPU, the heterogeneous system architecture (HSA) [31], which allows shared memory access from the CPU and GPU and reduces communication latency between them, has been extensively discussed.

An FPGA is an integrated circuit designed to be configured by a customer after manufacturing. An FPGA is the most widely spread programmable logic device. The advantages are that an FPGA can be configured to add new functions and reconfigured partly with lower cost than an application specific integrated circuit (ASIC). Formerly, FPGAs were mainly for academic use, but currently cloud servers use FPGAs to accelerate processing of search or memcached [32], which is a distributed cache system. The examples are Microsoft's Bing search [17] and IBM's NoSQL engine enhancement by FPGAs [33].

3 Typical Scenarios and Problems of Current Technology

With the advances in hardware power described in Sect. 2, high performance cloud services can be provided.

We show typical usage examples when various types of hardware comprise cloud services. When users operate applications for editing, recognizing, and analyzing graphics, their compute resources should be deployed on servers with strong GPUs. When users would like to accelerate NoSQL engines, their compute resources should be deployed on servers with FPGAs, the configurations of which accelerate NoSQL processing. Generally, performances of applications suitable for parallel processing and pipeline processing can be improved much [34] (e.g., some applications show more than one hundred times compared to CPU [35]) by GPU or FPGA processing.

Currently, there is no cloud provider that provides various types of hardware, such as FPGAs and strong GPUs. IBM SoftLayer provides baremetal and VM servers but no cloud provider provides FPGA hardware for specified computational logics. Therefore, when cloud providers provide various types of hardware in the future, users need to design appropriate server configurations and have much performance knowledge to benefit from high-performance applications.

To reduce user loads for designing server structures, we studied a technology to select the appropriate provisioning type of baremetal, container, or VM based on user performance requirements. This technology analyzes abstract templates of OpenStack Heat to confirm server connection patterns, such as the Web 3-tier model, and selects an appropriate provisioning type to satisfy the performance requirements, such as TPC-C [36] transaction throughput. TPC-C is a benchmark method of online transaction processing defined by TPC (Transaction Processing Performance Council). It measures tpmC (Transactions processed within 1 min) of assumed actual business operations. However, provisioning is only for normal CPU servers, and FPGAs or strong GPU servers are out of this paper's scope to select.

There have been studies on scheduling to arrange user resources on heterogeneous hardware [37, 38]. The typical issue is that to assign tasks appropriately to heterogeneous hardware is difficult for users. Performances are depended on computer resources specifics such as number of CPU/GPU, bandwidth of memory/network and application types of suitability to offload on GPU. For task scheduling, [37] targets MapReduce performance improvements and achieves high performances to assign Map tasks based on CPU and GPU performance ratio. The work of [38] discusses resource allocations on cloud with heterogeneous hardware.

Our work targets a platform which judges and deploys user applications where to run on a cloud with CPU, GPU and FPGA servers even users do not know much about each server specifications by automatic server selections, batch deployments of computer resources using Heat and automatic performance verifications which are proposed in Sect. 4.

4 Proposal of Performance Aware Server Selection, Configuration, Reconfiguration and Automatic Performance Verification Technology

In this section, we propose server selection, configuration, reconfiguration and automatic performance verification technology for cloud providers to satisfy user performance and functional requirements. Our proposed technology involves functions, an IaaS controller, such as OpenStack, various types of cloud hardware, and a test case database (DB). The figures describe OpenStack as an IaaS controller, but OpenStack is not a precondition of the proposed technology.

In Sect. 4.1, we explain the server selection steps. In Sect. 4.2, we explain the server configuration and reconfiguration steps. In Sect. 4.3, we explain the automatic performance verification steps of the provisioned servers described in Sects. 4.1 and 4.2.

4.1 Server Selection

Figure 2 shows the server selection steps based on user functional and performance requirements. These steps are for users who do not have much performance knowledge. There are four steps involved to select a server.

1. A user specifies functional and performance requirements to the server selection function. The functional requirements include low layer information of OS conditions such as normal Linux or customized Linux or non-Linux and high layer information of what applications run on the compute resources. For examples of the latter information, specifying Web server application, graphic analysis (which is suitable to GPU) and encryption processing (which is

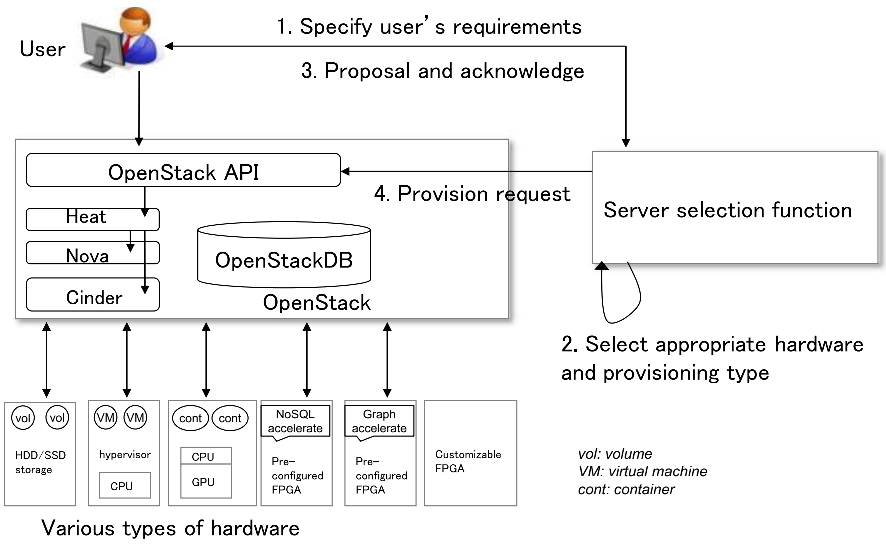


Fig. 2 Processing steps of server selection

suitable to FPGA). The performance requirements include server throughput and/or response time conditions. For examples of throughput, specifying certain index values of Himeno benchmark [39], TPC-C, UnixBench [40], etc.

2. The server selection function selects and proposes a server where compute resources are deployed. This proposal contains a specified server with a specified provisioning type. Figure 3 shows a server selection flow chart.
   First, if user requirements require a certain level or higher performance of specified computation, the server selection function selects the FPGA hardware configured for specified computation logics by baremetal provisioning. Because virtualization cancels out the benefits of an FPGA, we need to select berametal provisioning for FPGA servers.
   Second, if user requirements require a certain level or higher performance of graphics processing, the server selection function selects servers with strong GPUs by baremetal or container provisioning. Because VMs cannot sufficiently control GPUs, we need to select berametal or container for GPU control. However, container technology such as Linux Container (LXC), Docker or Hyper-V Container cannot change OS kernel setting, we must deploy compute resources which need customize OS by baremetal provisioning.
   Finally, for other general uses, the server selection function selects normal CPU servers by baremetal, container, or VM provisioning.
   For performance requirements, we pre-measured several performances (e.g., basic performance results of UnixBench in Sect. 5). The server selection function determines provisioning servers based on pre-measured performances data of throughput and response time. For example, when users request certain index values of Himeno benchmark, TPC-C, UnixBench, etc, our method selects appropriate servers which performance indexes satisfy user requests

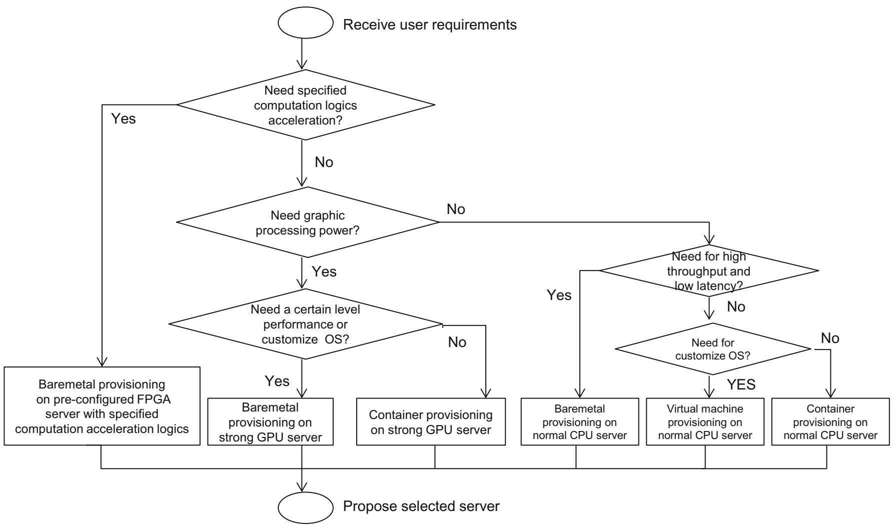

Receive user requirements
Need specified computation logics acceleration?
Yes
No
Need graphic processing power?
No
Yes
Need for high throughput and low latency?
Yes
No
Need a certain level performance or customize OS?
No
Yes
Need for customize OS?
No
YES
Baremetal provisioning on pre-configured FPGA server with specified computation acceleration logics
Baremetal provisioning on strong GPU server
Container provisioning on strong GPU server
Baremetal provisioning on normal CPU server
Virtual machine provisioning on normal CPU server
Container provisioning on normal CPU server
Propose selected server

Fig. 3 Server selection flow chart

(e.g., for graphical analysis, select GPU). However, our method cannot guarantee performances. Instead of guarantee, our method provides automatic performance verification results for users to support users' judge in Sect. 4.3. And locations of data centers affect response times much. Therefore, the server selection function selects servers on data center near to users' locations preferentially.

3. A user confirms the proposal and replies with an acknowledgement to the server selection function when the user satisfies the proposal. If the user does not satisfy the proposal, the user may re-select and send a specified server with a specified provisioning type.
4. The server selection function requests an IaaS controller to deploy the compute resource on specified hardware with a specified provisioning method. Then, the IaaS controller creates compute resources. Note that if users would like to create not only one compute server but several resources, such as virtual routers, the server selection function sends templates that describe the user environment structures by JavaScript Object Notation (JSON) and provisions them by OpenStack Heat [20] or other orchestration technology [41].

4.2 Server Configuration and Reconfiguration

Figure 4 shows FPGA server configuration and reconfiguration steps based on user requirements and usage change. These steps are for users who have much technical knowledge on FPGAs.

Configuration steps:

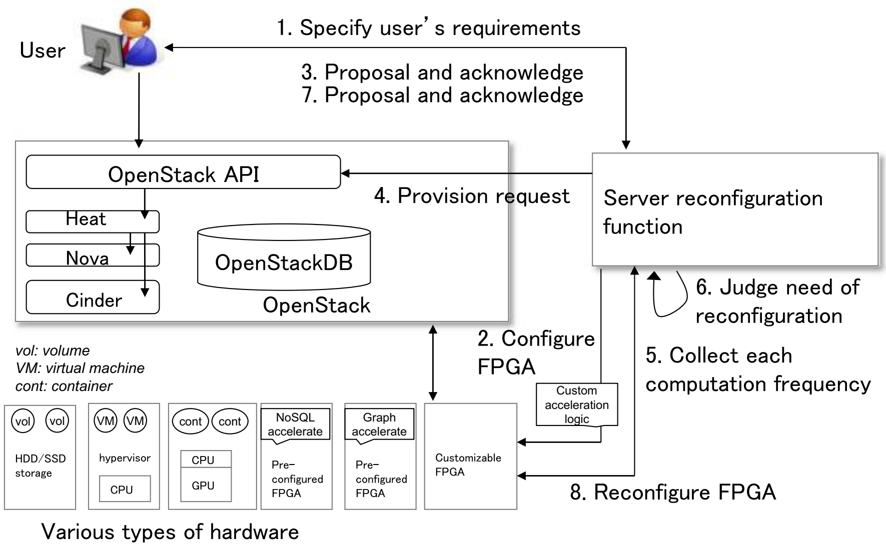


Fig. 4 Processing steps of server configuration and reconfiguration

1. A user specifies functional and performance requirements to the server reconfiguration function. Function requirements are what type of application is operated on the compute resources or the specified configuration to be set to the FPGA. Performance requirements are server throughput or latency.
2. The server reconfiguration function selects an un-used FPGA server and sets the configuration to accelerate specified computation logics to the FPGA based on step 1. If a specified configuration is not received in step 1, the configuration is set based on application types. For example, if an application is NoSQL, configuration for NoSQL is set from a pre-defined template because NoSQL acceleration by FPGA is commonly known [33]. Because FPGA configuration can be done within several 10 s of ms, there is not much of a problem regarding configuration time when cloud providers configure an FPGA for each user. The server reconfiguration function proposes baremetal provisioning on the configured FPGA to the user. Figure 5 shows a FPGA configuration flow chart.
3. A user confirms the proposal and replies with an acknowledgement to the server reconfiguration function when the user satisfies the proposed configuration. If the user does not satisfy the proposed configuration, the user may re-design and send a specified configuration of the FPGA.
4. The server reconfiguration function requests an IaaS controller to deploy the compute resource on the configured FPGA server by baremetal provisioning. Then, the IaaS controller provisions the compute resource by baremetal provisioning.

   The next reconfiguration steps are for FPGA reconfiguration for adapting to user usage change during user operation. With this reconfiguration, compute

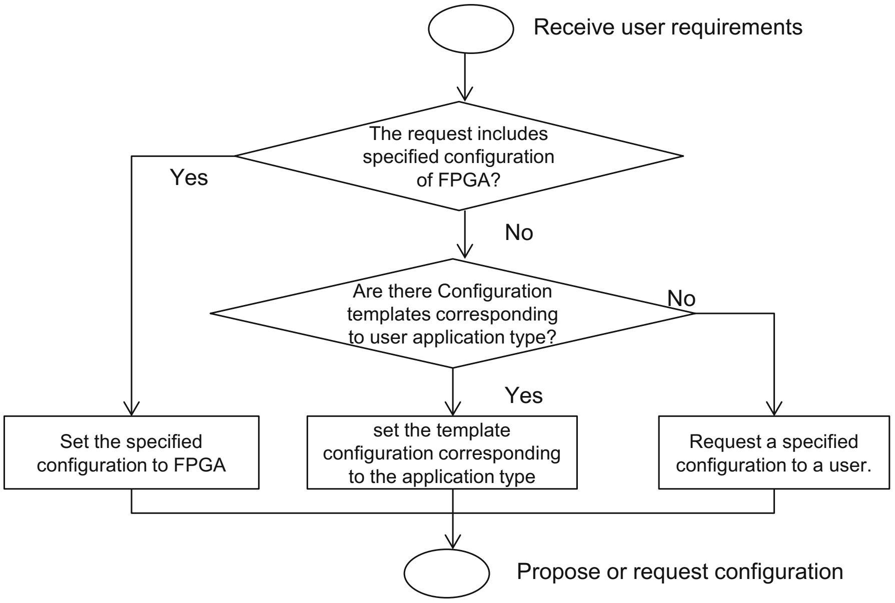

Receive user requirements
The request includes specified configuration of FPGA?
Yes
No
Are there Configuration templates corresponding to user application type?
No
Yes
Set the specified configuration to FPGA
set the template configuration corresponding to the application type
Request a specified configuration to a user.
Propose or request configuration

Fig. 5 Server configuration flow chart

resources can be adapted to data increase or traffic change. Note that these latter reconfiguration steps can be applied to pre-configured FPGA servers provisioned in Sect. 4.1 step 4.

Reconfiguration steps:

5. The server reconfiguration function periodically collects each computation frequency in the FPGA server. For example, graph analysis number per minute and NoSQL processing number per minute are collected. OpenStack itself does not collect server traffic or computation frequency of provisioned servers. Therefore, we run a cron process which outputs computation frequency on each provisioned FPGA server, and then a server reconfiguration function collects these data periodically.

We assume periodical intervals are daily, weekly or monthly. This is because high frequency proposal of server reconfiguration is bothering for users. When the server reconfiguration function collects data, network load can be balanced by time shift of collecting each server data. (e.g., collecting user A's servers in night and collecting user B's servers in midday)

6. The server reconfiguration function determines whether FPGA reconfiguration is needed by matching each computation frequency on the server and accelerated computation logic of the FPGA. For example, if the FPGA server has accelerated graph analysis logics but the graph analysis number per minute is 10 and NoSQL processing number per minute is 1000, reconfiguration to accelerate NoSQL processing should be done. Specifically, the server reconfiguration function attempts to reconfigure when the non-accelerated computation frequency exceeds a certain threshold value. When the server reconfiguration function determines that reconfiguration is needed, it proposes the reconfiguration to the user.
6. The user confirms the proposal and replies with an acknowledgement to the server reconfiguration function when the user satisfies the proposed reconfiguration. If the user does not satisfy the proposed reconfiguration, the user may re-design and send a specified configuration of the FPGA or may reject the reconfiguration proposal.
   When users reject proposals of reconfiguration, cloud services continue operations on current server configurations. There are users who do not want to reconfigure servers during production operations. For them, we provide choices to users whether cloud providers propose server reconfiguration or not. Or we take a method to extend interval of server reconfiguration checks when users reject proposals of reconfiguration.
8. The server reconfiguration function reconfigures the user's FPGA to accelerate the computation logics which frequency of the computation is increased. Because recent FPGAs can be reconfigured even during in operation, we use existing reconfiguration technologies of Altera or Xilinx. For example, Altera Stratix V, Cyclone V and Xilinx Virtex series support dynamic reconfiguration.

Existing reconfiguration technologies can reconfigure FPGA logics within several seconds and reconfiguration times do not become huge overhead.

Though reconfiguration completes within several seconds, users who want reliability may hesitate to accept reconfiguration proposals because cloud servers are on actual operation. We assume about 1/3 users accept reconfiguration proposal. When carriers such as NTT released new hosting or cloud plans with more resources as same prices as previous plans, about 1/3 users moved to new plans fast. So, we estimate 1/3 users are positive to change configuration which can enhance performances. To increase reconfiguration, we plan to replicate users' cloud environment by OpenStack Heat, conduct automatic performance tests on the replicated environment and show results of performance improvements to users. This enables users to confirm the effect of reconfiguration beforehand, then we suppose more users accept reconfiguration proposals.

And the author recently studies to enhance performances not only changing FPGA logic like this subsection but also changing resource amount or ratio of CPU and GPU. We will report it in another paper.

4.3 Automatic Performance Verification

In this subsection, we discuss the automatic performance verification function of our proposed technology for provisioned user resources. Because estimating application performance before provisioning is difficult, this verification shows that the actual provisioned resource satisfies user performance requirements. If users do not satisfy the confirmed performances, users can delete cloud resources and try another one.

We previously proposed an automatic verification function for cloud user environments [42–47]. It replicates virtual environments, extracts regression test cases corresponding to the installed OS, middleware, and applications, and executes them by using the Jenkins tool to verify software patches automatically.

Figure 6 shows the automatic performance verification steps. Note that virtual resources, such as virtual routers, are already created in Sect. 4.1 step 4 by OpenStack Heat or [41].

1. The automatic performance verification function selects appropriate performance verification test cases from the test case DB. This function selects test cases not only for each individual server performance but also for several servers such as transaction processing of the Web 3-tier model. For example, Unix benchmark UnixBench or 3-tier model transaction benchmark TPC-C can be used for performance test cases.
2. The automatic performance verification function executes the performance test cases selected in Step 1. We use the Jenkins tool to execute performance test cases. Although performance verification is targeted for compute servers, verification test cases are executed for all resources in a user environment. In a case in which containers with web servers are under one virtual load balancer, web server performance needs to be tested via the virtual load balancer.

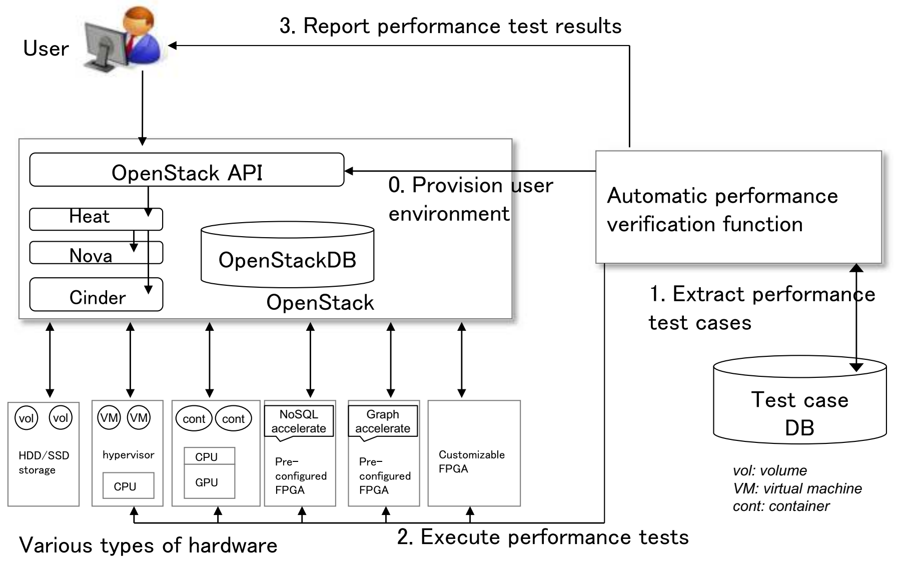


Fig. 6 Processing steps of automatic performance verification

3. The automatic performance verification function collects the results of test cases for each user environment using the Jenkins tool. Collected data are sent to users via mail or Web. Regarding to throughput, for examples, TPC-C's tpmC can be used for online transaction performance index and Himeno benchmark's GFlops results can be used for index of fluid analysis on GPU. Regarding to response time, as same as throughput, TPC-C transaction response time can be used.

Users confirm whether the performances satisfy their requests from the data. After confirming throughput and response time results for user applications, users judge to start IaaS cloud service usages.

5 Performance Confirmation of Proposed Technology

The proposed technology provisions compute resources on an appropriate server based on user requirements and shows performances by automatic verification. In the case of FPGA server use, updated computational logics are reconfigured by adapting to changes in user usage. In general, GPU performance is 10–100 times better than that of CPUs for graphics processing, and configured FPGA performance is several times better than that of GPUs and is 10–100 times better than that of CPUs for computations suitable for FPGA. Our technology can reduce server number for users.

We measured performance to show the effectiveness of the proposed technology. We used Ironic for baremetal provisioning, Docker for container, and KVM for VM.

5.1 Performance Measurement Conditions

We measured two performance patterns. In pattern 1, provisioning times of baremetal, container, and VM were measured using the OpenStack instance creation API. The FPGA requires baremetal provisioning and GPU requires container or baremetal provisioning to control. Though the FPGA configuration setting only took several 10 ms, the total provisioning time which includes network setting, volume attachment and other setting needs to be confirmed. Because containers or VMs share one physical server, we confirm performance degradation of several concurrent instance creations for container and VM cases.

In pattern 2, automatic performance verification times are measured. We do not need FPGA or GPU performance itself but automatic performance verification overheads, so we confirm automatic performance verification processing times using VMs with changing concurrent processing amount. Because server selection takes only a few seconds, we did not measure such performance.

Because our work uses container for GPU processing and baremetal for FPGA processing, we also pre-measured base performance indexes of UnixBench to compare performances. For other benchmarks or applications performance enhancements with heterogeneous hardware, we will refer existing technologies results such as [37, 38, 48, 49].

Measurement patterns:

- Pattern 1 measurement item. Server provisioning time (the time of server start up from instance creation API call).
- Pattern 2 measurement items. Tester resource preparation, test case selection, and test case execution times.
  Tester resource preparation includes network connection or port setting of the virtual switch and compute servers. The selected performance test case is a transaction performance test of TPC-C.
- Pre-measurement items. UnixBench is conducted to acquire UnixBench performance indexes. UnixBench is a major system performance benchmark and we measured to confirm basic performances. These data can be used for server selection in Sect. 4.1.

User environment configuration:

- Each user environment has two compute servers, two volumes, two virtual Layer 2 networks, and one virtual router.
- Each compute server has one attached volume of 10 GB with CentOS 6.
- Apache 2.1 and Tomcat 6.0 are installed on one volume, and MySQL 5.6 is installed on one volume for the compute servers' software.

Concurrent processing amount:

- Changing from 1 to 5.

5.2 Performance Measurement Environment

Figure 7 is the test environment. There are many servers for OpenStack virtual resources, the main server used in our study was server selection, configuration, reconfiguration, and automatic performance verification server. These servers were connected with Gigabit Ethernet via Layer 2 and Layer 3 switches.

Figure 7 also shows the physical and virtual servers and the modules in each server. For example, the OpenStack API server was a virtual server, it was in both the Internet and control segments, and its modules were a cinder scheduler, cinder API, nova-api, keystone, glance-registry, nova-scheduler, Ironic, and PXE for baremetal boot. Two servers were used for redundancy.

Figure 8 lists the specifications and usage for each server. For example, in the DB case (6th row), the hardware is HP ProLiant BL460c G1, server is a physical server, name is DB, the main usage is OpenStack and test case DB, the CPU is a Quad-Core Intel Xeon 1600 MHz*2 and the number of cores is 8, RAM was 24 GB, assigned HDD is 72 GB, and there are four network interface cards (NICs).

5.3 Performance Measurement Results

Figure 9 shows a pre-measured basic performance of UnixBench. Vertical axis shows UnixBench performance index value and horizon axis shows each server with changing number of virtual servers. Before our heterogeneous cloud services launch, we measure more basic performances with plural servers. Based on them, the server selection function selects appropriate servers for user requirements.

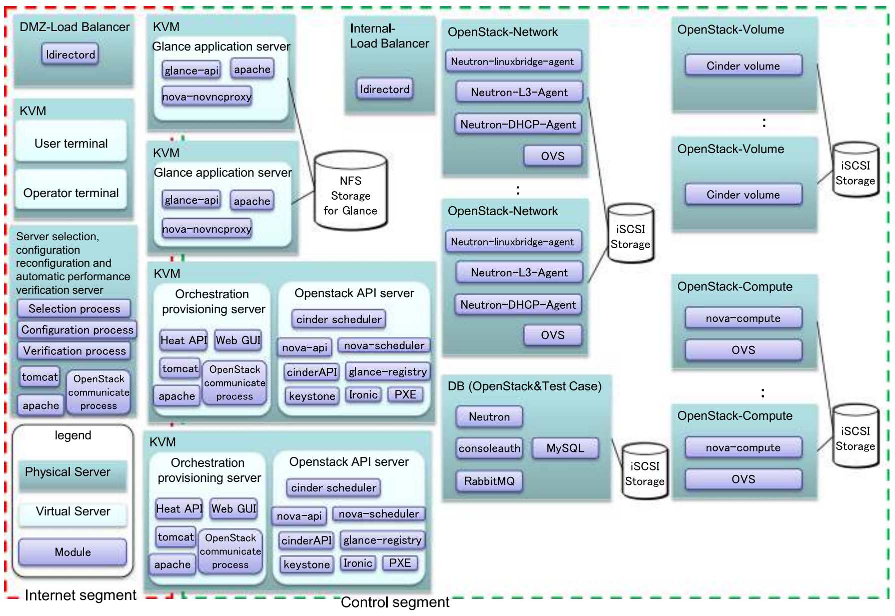
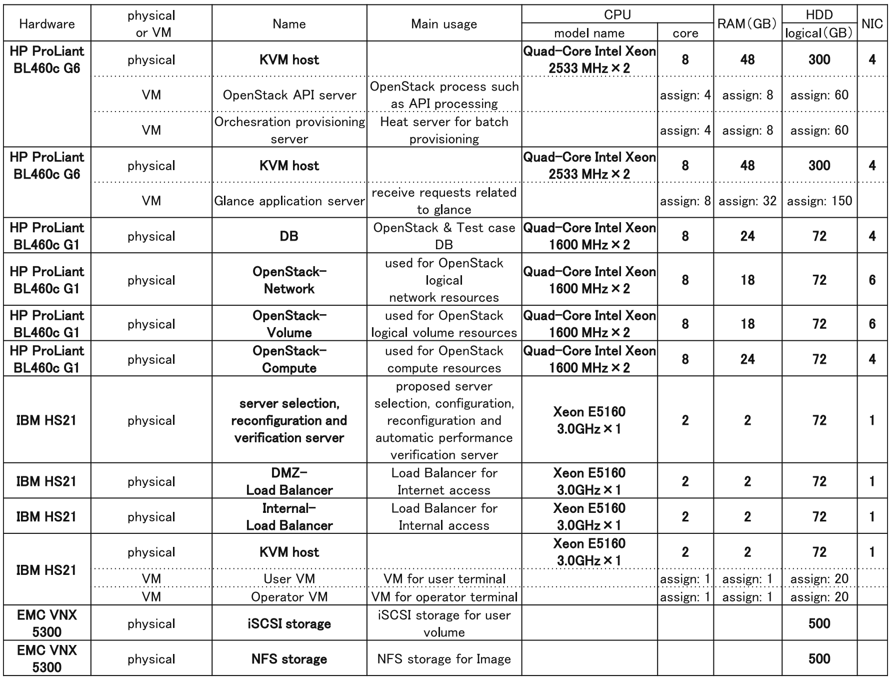


Fig. 7 Test environment for confirming proposed technology



Hardware	physical or VM	Name	Main usage	CPU		RAM(GB)	HDD	NIC
				model name	core		logical(GB)	
HP ProLiant BL460c G6	physical	**KVM host**		**Quad-Core Intel Xeon 2533 MHz × 2**	**8**	**48**	**300**	**4**
	VM	OpenStack API server	OpenStack process such as API processing		assign: 4	assign: 8	assign: 60	
	VM	Orchesration provisioning server	Heat server for batch provisioning		assign: 4	assign: 8	assign: 60	
HP ProLiant BL460c G6	physical	**KVM host**		**Quad-Core Intel Xeon 2533 MHz × 2**	**8**	**48**	**300**	**4**
	VM	Glance application server	receive requests related to glance		assign: 8	assign: 32	assign: 150	
HP ProLiant BL460c G1	physical	**DB**	OpenStack & Test case DB	**Quad-Core Intel Xeon 1600 MHz × 2**	**8**	**24**	**72**	**4**
HP ProLiant BL460c G1	physical	**OpenStack-Network**	used for OpenStack logical network resources	**Quad-Core Intel Xeon 1600 MHz × 2**	**8**	**18**	**72**	**6**
HP ProLiant BL460c G1	physical	**OpenStack-Volume**	used for OpenStack logical volume resources	**Quad-Core Intel Xeon 1600 MHz × 2**	**8**	**18**	**72**	**6**
HP ProLiant BL460c G1	physical	**OpenStack-Compute**	used for OpenStack compute resources	**Quad-Core Intel Xeon 1600 MHz × 2**	**8**	**24**	**72**	**4**
IBM HS21	physical	**server selection, reconfiguration and verification server**	proposed server selection, configuration, reconfiguration and automatic performance verification server	**Xeon E5160 3.0GHz × 1**	**2**	**2**	**72**	**1**
IBM HS21	physical	**DMZ-Load Balancer**	Load Balancer for Internet access	**Xeon E5160 3.0GHz × 1**	**2**	**2**	**72**	**1**
IBM HS21	physical	**Internal-Load Balancer**	Load Balancer for Internal access	**Xeon E5160 3.0GHz × 1**	**2**	**2**	**72**	**1**
IBM HS21	physical	**KVM host**		**Xeon E5160 3.0GHz × 1**	**2**	**2**	**72**	**1**
	VM	User VM	VM for user terminal		assign: 1	assign: 1	assign: 20	
	VM	Operator VM	VM for operator terminal		assign: 1	assign: 1	assign: 20	
EMC VNX 5300	physical	**iSCSI storage**	iSCSI storage for user volume				**500**	
EMC VNX 5300	physical	**NFS storage**	NFS storage for Image				**500**	

Fig. 8 Server specifications of test environment

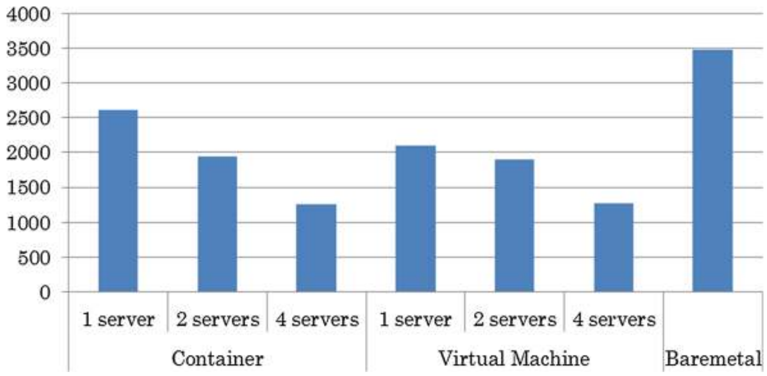


Fig. 9 UnixBench performance indexes of 3 provisioning types

Figure 10 shows a comparison of start-up provisioning times including a broken axis. When there were several concurrent provisioning numbers, the average provisioning time is shown. Baremetal provisioning took more than 500 s and much longer than container provisioning. This is because baremetal provisioning requires data transfer for PXE boot and takes a long time. Regarding the VM cases, OS start up took a long time. However, a container only requires process invocation and takes less time. Therefore, if users need temporal graphics editing or analyzing

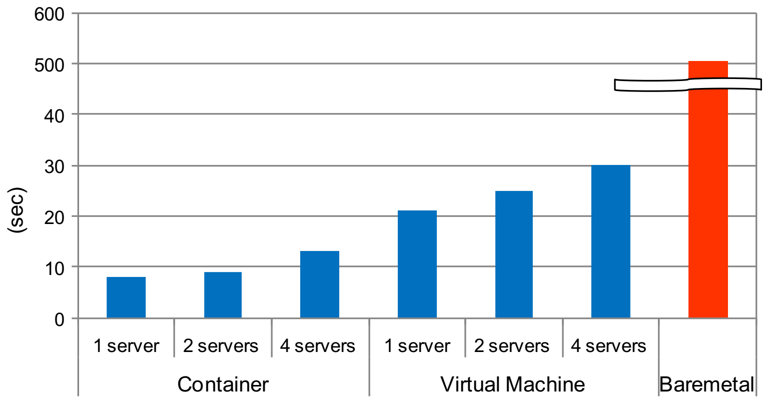


Fig. 10 Processing time comparison of 3 provisioning types

power, it is appropriate to provision containers on strong GPU servers and to delete containers when graphics processing is over. Regarding the specified computational logic processing of an FPGA, more than 500 s is needed before users actually use even though FPGA configuration takes only few seconds.

Figure 11 shows the processing times of verification to confirm feasibility of automatic performance verification. In all cases of concurrent processing (1, 3 and 5), test case selection and execution remained almost the same; about 15 min. When the concurrent processing amount was 3 or 5, tester resource preparation, which includes network connection setting, port setting and so on, took more time compared to a single processing case. This is because OpenStack load increased. However, automatic performance verification can be done within 30 min after the provisioning time in Fig. 10. Therefore, users can judge whether they can start to use heterogeneous cloud services in a sufficiently short time.

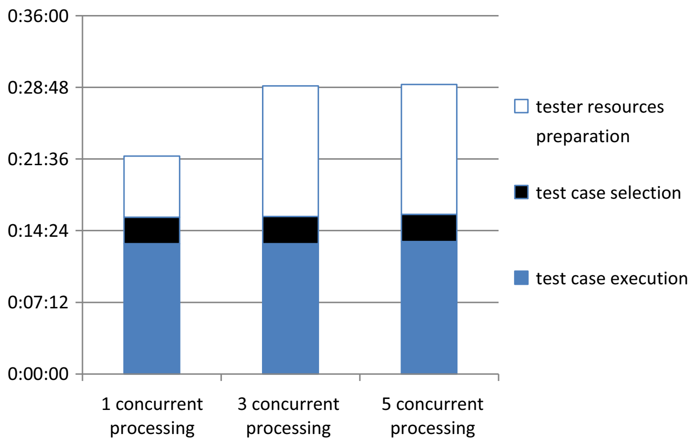


Fig. 11 Processing time of automatic performance verification

Of course, Figs. 10 and 11 only show examples of provision time and time of performance confirmation with TPC-C, users need to confirm actual performances of their applications using automatic performance verification function.

6 Related Work

Wuhib studied dynamic resource allocation on OpenStack [50]. There have been studies on the resource arrangement on hosting services to effectively use physical server resources [51, 52]. Similarly, we also used resource arrangement technology on OpenStack but focused on resolving problems of appropriate server selection from various types of hardware. We previous studied an appropriate provisioning type of virtual machine, container or baremetal to users. However, FPGA or GPU servers are out of this paper's scope.

For heterogeneous programming, PGI Accelerator Compilers with OpenACC Directives [53] can compile C/C++/Fortran codes with OpenACC directives and deploy execution binary to run on GPU and CPU. OpenACC directives indicate parallel processing sections, then PGI compiler creates execution binary for GPU and CPU based on the specified directives. Aparapi (A PARallel API) of Java [54] is API to call GPGPU from Java. Using this API, Java byte code is compiled to OpenCL and run when it is executed. To control FPGA, development tools of OpenCL for FPGA are provided by Altera and Xilinx (for example, Altera SDK for OpenCL [55]). Of course, these technologies need clear indications of OpenACC, Aparapi or OpenCL descriptions for GPU or FPGA executions. We also propose PaaS for applications described by general programming languages to deploy offloadable logics on GPU and FPGA automatically [56, 57]. The work of [56] detects registered code patterns from users' applications source codes, then outputs OpenCL C codes and deploys them to each server. This paper can use these improvements of compiler technologies.

There have been studies on the use of FPGAs in cloud systems such as Microsoft's Bing search [17], JP Morgan's banking system to enhance scalability [58], and memcached acceleration by FPGAs [59]. Some Microsoft researchers have the opinion that Moore's law is coming to an end because the cost per transistor has not decreased much; therefore, dedicated processing of appropriate servers and not parallel processing of uniform servers has become important. We also expect the number of cloud providers using FPGAs will increase.

Besides FPGAs, the plastic cell architecture (PCA) [60] is a programmable logic device. Because early FPGAs did not have sufficient reconfigurability, the PCA targets self-reconfiguration adapting to usage, function, specifications, and environment. Though the use of the PCA is not wide spread, the proposed server reconfiguration function of our technology partly uses the concept of PCA.

For cloud business, GPU and FPGA adoptions are increasing in cloud data center, but general users cannot use them easily. FPGAs of Microsoft Bing search [17] are not provided for cloud user directly. Amazon Web Services GPU instances enable users to utilize GPU resources by CUDA [30], but there are few CUDA programmers. Our company NTT is now studying to provide cloud services with GPU and FPGA. Thus, when NTT starts heterogeneous cloud services, this technology can be provided as application PaaS (aPaaS) to users. Using this technology, users can improve application performances without much knowledge of GPU and FPGA, because this technology can run graphical processing on GPU and encryption processing on FPGA automatically. Cloud providers can reduce cost because cloud providers can enhance application performances much by few number of GPU/FPGA servers. Users also can coordinate outer systems using service coordination technologies such as [61–66]. We also consider to use GPU/FPGA servers to IoT applications [67–74].

7 Conclusion

We proposed a server selection, configuration, reconfiguration and automatic performance verification technology to provide high performance cloud services based on user requirements using various types of hardware such as FPGAs or strong GPU servers.

The server selection function of our technology analyzes user functional and performance requirements and provisions baremetal FPGA servers with specified configuration for accelerating specified computational logics, provisions strong GPU servers as baremetals or containers for graphics processing, and provisions normal CPU servers as baremetals, containers, or VMs for general applications. The server configuration and reconfiguration functions set configurations to FPGAs to accelerate specified computational logics based on user requirements. Moreover, they collect the frequency of each computation in the operation phases and reconfigure FPGAs to accelerate computation. After compute resource provisioning, the automatic performance verification function of our technology automatically executes performance test cases; thus, users can confirm system performance.

We confirmed the server provisioning time and automatic performance verification time through performance measurements. We found that our technology can quickly provide appropriate user environments.

In the future, we will implement our technology not only for OpenStack but also for other IaaS platforms such as CloudStack. We will also increase the number of performance test cases for actual use cases of IaaS services. In the same performance test cases, we will increase the number of FPGA configuration templates for various computational logics to accelerate various applications. We will then cooperate with IaaS cloud service providers to provide managed services in which providers propose appropriate servers.

Authors' Contributions YY carried out all work of the proposed technology and drafted the final manuscript.

Compliance with Ethical Standards

Conflict of interests The author declares that he has no competing interests.

References

1. Yamato, Y., Nishizawa, Y., Nagao, S., Sato, K.: Fast and reliable restoration method of virtual resources on OpenStack. IEEE Trans. Cloud Comput. (2015). doi:10.1109/TCC.2015.2481392
2. Yamato, Y., Nishizawa, Y., Muroi, M., Tanaka, K.: Development of resource management server for production IaaS services based on OpenStack. J. Inf. Process. **23**(1), 58–66 (2015)
3. Yamato, Y., Shigematsu, N., Miura, N.: Evaluation of agile software development method for carrier cloud service platform development. IEICE Trans. Inf. Syst. **E97-D**(11), 2959–2962 (2014)
4. Yamato, Y.: Cloud storage application area of HDD-SSD hybrid storage, distributed storage and HDD storage. IEEJ Trans. Electr. Electron. Eng. **11**(5), 674–675 (2016)
5. Yamato, Y.: OpenStack hypervisor, container and baremetal servers performance comparison. IEICE Commun. Express **4**(7), 228–232 (2015)
6. Yamato, Y., Katsuragi, S., Nagao, S., Miura, N.: Software maintenance evaluation of agile software development method based on OpenStack. IEICE Trans. Inf. Syst. **E98-D**(7), 1377–1380 (2015)
7. Yamato, Y.: Use case study of HDD-SSD hybrid storage, distributed storage and HDD storage on OpenStack. In: 19th International Database Engineering \& Applications Symposium (IDEAS2015), pp. 228–229, July 2015
8. Yamato, Y., Nishizawa, Y., Nagao, S.: Fast restoration method of virtual resources on OpenStack. In: IEEE Consumer Communications and Networking Conference (CCNC2015), pp. 607–608, Jan 2015
9. Yamato, Y.: Implementation evaluation of amazon SQS compatible functions. IEEJ Trans. Electron. Inf. Syst. **135**, 561–562 (2015)
10. Sunaga, H., Takemoto, M., Yamato, Y., Yokohata, Y., Nakano, Y., Hamada, M.: Ubiquitous life creation through service composition technologies. In: World Telecommunications Congress 2006 (WTC2006), May 2006
11. Yamato, Y., Tanaka, Y., Sunaga, H.: Context-aware ubiquitous service composition technology. In: The IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006), pp. 51–61, Apr 2006
12. Yamato, Y., Nakano, Y., Sunaga, H.: Study and evaluation of context-aware service composition and change-over using BPEL engine and semantic web techniques. In: IEEE Consumer Communications and Networking Conference (CCNC 2008), pp. 863–867, Jan 2008
13. Takemoto, M., Yamato, Y., Sunaga, H.: Service elements and service templates for adaptive service composition in a ubiquitous computing environment. In: The 9th Asia-Pacific Conference on Communications (APCC2003), vol. 1, pp. 335–338, Sept 2003
14. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), pp. 164–177, Oct 2003
15. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: OLS '07: The 2007 Ottawa Linux Symposium, pp. 225–230, July 2007
16. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th Symposium on Opearting Systems Design and Implementation (OSDI'04), pp. 137–150, Dec 2004
17. Putnam, A., Caulfield, A.M., Chung, E.S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G.P., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Larus, J., Peterson, E., Pope, S., Smith, A., Thong, J., Xiao, P.Y., Burger, D.: A reconfigurable fabric for accelerating large-scale datacenter services. In: Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA'14), pp. 13–24, June 2014
18. Amazon Elastic Compute Cloud web site, http://aws.amazon.com/ec2
19. Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., Shenker, S.: Extending networking into the virtualization layer. In: Proceedings of 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII), Oct 2009
20. OpenStack Heat web site, https://wiki.openstack.org/wiki/Heat

21. Yamato, Y., Moriya, T., Ogawa, T., Akahani, J.: Survey of public IaaS cloud computing API. IEEJ Trans. Electron. Inf. Syst. **132**, 173–180 (2012)
22. Sunaga, H., Yamato, Y., Ohnishi, H., Kaneko, M., Iio, M., Hirano, M.: Service delivery platform architecture for the next-generation network. In: ICIN2008, Session 9-A, Oct 2008
23. Yamato, Y.: Method of service template generation on a service coordination framework. In: 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov 2004
24. Miyagi, Y., Yamato, Y., Nakano, Y., Hirano, M.: Support system for developing web service composition scenarios without need for web application skills. In: IEEE International Symposium on Applications and the Internet (SAINT'09), pp. 193–196, July 2009
25. Yamato, Y., Ohnishi, H., Sunaga, H.: Development of service control server for web-telecom coordination service. In: IEEE International Conference on Web Services (ICWS 2008), pp. 600–607, Sept 2008
26. Yamato, Y., Nakano, Y., Sunaga, H.: Context-Aware Service Composition and Change-over Using BPEL Engine and Semantic Web, pp. 257–270. INTECH Open Access Publisher, Rijeka, Croatia (2010)
27. Yamato, Y., Ohnishi, H., Sunaga, H.: Study of service processing agent for context-aware service coordination. In: IEEE International Conference on Service Computing (SCC 2008), pp. 275–282, July 2008
28. Nakano, Y., Yamato, Y., Takemoto, M., Sunaga, H.: Method of creating web services from web applications. In: IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007), pp. 65–71, June 2007
29. Kamp, P.-H., Watson, R.N.M.: Jails: confining the omnipotent root. In: Proceedings of the 2nd International SANE Conference, May 2000
30. Sanders, J., Kandrot, E.: CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley, ISBN-0131387685, 2011
31. Kyriazis, G.: Heterogeneous system architecture: a technical review. White paper, HSA Foundation, Aug 2012. http://developer.amd.com/wordpress/media/2012/10/hsa10.pdf
32. Fitzpatrick, B.: Distributed caching with memcached. Linux Journal **2004**(124), 5 (2004)
33. Brech, B., Rubio, J., Hollinger, M.: Data Engine for NoSQL—IBM Power Systems Edition. White Paper, IBM (2015)
34. Che, S., Li, J., Sheaffer, J.W., Skadron, K., Lach, J.: Accelerating compute-intensive applications with GPUs and FPGAs. In: IEEE Symposium on Application Specific Processors (SASP 2008), pp. 101–107, June 2008
35. Jones, D.H., Powell, A., Bouganis, C.S., Cheung, P.Y.K.: GPU versus FPGA for high productivity computing. In: IEEE International Conference on Field Programmable Logic and Applications (FPL 2010), pp. 119–124, Aug 2010
36. TPC-C web site, http://www.tpc.org/tpcc/
37. Shirahata, K., Sato, H., Matsuoka, S.: Hybrid map task scheduling for GPU-based heterogeneous clusters. In: IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), pp. 733–740, Dec 2010
38. Lee, G., Chunz, B.-G., Katzy, R.H.: Heterogeneity-aware resource allocation and scheduling in the cloud. In: Proceedings of HotCloud, pp. 1–5, 2011
39. Himeno Benchmark web site, https://openbenchmarking.org/test/pts/himeno
40. UnixBench web site, https://code.google.com/p/byte-unixbench/
41. Yamato, Y., Muroi, M., Tanaka, K., Uchimura, M.: Development of template management technology for easy deployment of virtual resources on OpenStack. J. Cloud Comput. (2014). doi:10.1186/s13677-014-0007-3
42. Yamato, Y.: Automatic verification technology of software patches for user virtual environments on IaaS cloud. J. Cloud Comput. **4**, 4 (2015). doi:10.1186/s13677-015-0028-6
43. Yamato, Y.: Automatic system test technology of user virtual machine software patch on IaaS cloud. IEEJ Trans. Electr. Electron. Eng. **10**(S1), 165–167 (2015)
44. Yamato, Y.: Performance-aware server architecture recommendation and automatic performance verification technology on IaaS cloud. Serv. Oriented Comput. Appl. (2016). doi:10.1007/s11761-016-0201-x
45. Yamato, Y.: Server structure proposal and automatic verification technology on IaaS cloud of plural type servers. In: International Conference on Internet Studies (NETs2015), July 2015

46. Yamato, Y.: Automatic verification technology of virtual machine software patch on IaaS cloud. World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng. **9**(1), 347–354 (2015)
47. Yamato, Y.: Automatic verification for plural virtual machines patches. In: The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp. 837–838, July 2015
48. Vignesh, T.R., Wenjing, M., David, C., Gagan, A.: Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. In: The 24th ACM International Conference on Supercomputing (ICS'10), pp. 137–146, 2010
49. Crago, S., Dunn, K., Eads, P., Hochstein, L., Kang, D.-I., Kang, M., Modium, D., Singh, K., Suh, J., Walters, J.P.: Heterogeneous cloud computing. In: IEEE International Conference on Cluster Computing (CLUSTER), pp. 378–385, Sept 2011
50. Wuhib, F., Stadler, R., Lindgren, H.: Dynamic resource allocation with management objectives—implementation for an OpenStack cloud. In: Proceedings of Network and service management, 2012 8th international conference and 2012 workshop on systems virtualiztion management, pp. 309–315, Oct 2012
51. Liu, X., Zhu, X., Padala, P., Wang, Z., Singhal, S.: Optimal multivariate control for differentiated services on a shared hosting platform. In: Proceedings of the IEEE Conference on Decision and Control, pp. 3792–3799, 2007
52. Urgaonkar, B., Shenoy, P., Roscoe, T.: Resource overbooking and application profiling in shared hosting platforms. In: Symposium on Operating Systems Design and Implementation, pp. 239–254. ACM Press, 2002
53. PGI compiler web site, https://www.pgroup.com/resources/accel.htm
54. Aparapi web site, http://developer.amd.com/tools-and-sdks/opencl-zone/aparapi/
55. Altera SDK for OpenCL web site, https://www.altera.com/products/design-software/embedded-software-developers/opencl/documentation.html
56. Yamato, Y.: Optimum application deployment technology for heterogeneous IaaS cloud. J. Inf. Process. **25**(1), 56–58 (2017)
57. Yamato, Y.: Proposal of optimum application deployment technology for heterogeneous IaaS cloud. In: 6th International Workshop on Computer Science and Engineering (WCSE2016), pp. 34–37, June 2016
58. Richards, P., Weston, S.: Technology in Banking—A Problem in Scale and Complexity. Stanford University, Stanford (2011)
59. Lim, K., Meisner, D., Saidi, A.G., Ranganathan, P., Wenisch, T.F.: Thin servers with smart pipes: designing SoC accelerators for Memcached. In: Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13), pp. 36–47, June 2013
60. Ito, H., Konishi, R., Nakada, H., Oguri, K., Nagoya, A., Inamori, M.: Dynamically reconfigurable logic LSI-PCA-1: the first realization of plastic cell architecture. IEICE Trans. Inf. Syst. **E86-D**(5), 859–867 (2003)
61. Yokohata, Y., Yamato, Y., Takemoto, M., Sunaga, H.: Service composition architecture for programmability and flexibility in ubiquitous communication networks. IEEE International Symposium on Applications and the Internet Workshops (SAINTW'06), pp. 145–148, Jan 2006
62. Nakano, Y., Yamato, Y., Sunaga, H.: Web-service-based avatar service modeling in the next generation network. In: The 7th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT2008), pp. 52–57, Apr 2008
63. Nakano, Y., Takemoto, M., Yamato, Y., Sunaga, H.: Effective web-service creation mechanism for ubiquitous service oriented architecture. In: The 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE 2006), pp. 85, June 2006
64. Takemoto, M., Ohishi, T., Iwata, T., Yamato, Y., Tanaka, T., Tokumoto, S., Shimamoto, N., Kurokawa, A., Sunaga, H., Koyanagi, K.: Service-composition method and its implementation in service-provision architecture for ubiquitous computing environments. IPSJ J. **46**(2), 418–433 (2005)
65. Yamato, Y., Sunaga, H.: Context-aware service composition and component change-over using semantic web techniques. In: IEEE International Conference on Web Services (ICWS 2007), pp. 687–694, July 2007
66. Yokohata, Y., Yamato, Y., Takemoto, M., Tanaka, E., Nishiki, K.: Context-aware content-provision service for shopping malls based on ubiquitous service-oriented network framework and authentication and access control agent framework. In: IEEE Consumer Communications and Networking Conference (CCNC 2006), pp. 1330–1331, Jan 2006

67. Yamato, Y., Fukumoto, Y., Kumazaki, H.: Predictive maintenance platform with sound stream analysis in edges. J. Inf. Process. **25**, 317–320 (2017)
68. Yamato, Y., Fukumoto, Y., Kumazaki, H.: Proposal of shoplifting prevention service using image analysis and ERP check. IEEJ Trans. Electr. Electron. Eng. **12**(S1), 141–145 (2017)
69. Yamato, Y., Fukumoto, Y., Kumazaki, H.: Analyzing machine noise for real time maintenance. In: Eighth International Conference on Graphic and Image Processing (ICGIP2016), Oct 2016
70. Yamato, Y.: Proposal of vital data analysis platform using wearable sensor. In: 5th IIAE International Conference on Industrial Application Engineering 2017 (ICIAE2017), pp. 138–143, Mar 2017
71. Yamato, Y., Fukumoto, Y., Kumazaki, H.: Security camera movie and ERP data matching system to prevent theft. In: IEEE Consumer Communications and Networking Conference (CCNC2017), pp. 1021–1022, Jan 2017
72. Yamato, Y.: Experiments of posture estimation on vehicles using wearable acceleration sensors. In: The 3rd IEEE International Conference on Big Data Security on Cloud (BigDataSecurity 2017), May 2017
73. Yamato, Y., Fukumoto, Y., Kumazaki, H.: Proposal of real time predictive maintenance platform with 3D printer for business vehicles. Int. J. Inf. Electron. Eng. **6**(5), 289–293 (2016)
74. Yamato, Y., Kumazaki, H., Fukumoto, Y.: Proposal of lambda architecture adoption for real time predictive maintenance. In: 2016 Fourth International Symposium on Computing and Networking (CANDAR2016), pp. 713–715, Nov 2016

Yoji Yamato received his B. S., M. S. degrees in Physics and Ph.D. degrees in General Systems Studies from University of Tokyo, Japan in 2000, 2002 and 2009, respectively. He joined NTT Corporation, Japan in 2002. Currently he is a senior Research Engineer of NTT Network Service Systems Laboratories. There, he has been engaged in developmental research of Cloud computing and Peer-to-Peer networks. Dr. Yamato is a senior member of IEEE and IEICE.