applied sciences

MDPI

Article

Resource Utilization Scheme of Idle Virtual Machines for Multiple Large-Scale Jobs Based on OpenStack

Jueun Jeon [1], Jong Hyuk Park [2,*] and Young-Sik Jeong [1,*]

[1] Department of Multimedia Engineering, Dongguk University, Seoul 100-715, Korea; jry02107@dongguk.edu
[2] Department of Computer Science and Engineering, Seoul National University of Science & Technology (SeoulTech), Seoul 139-743, Korea
* Correspondence: jhpark1@seoultech.ac.kr (J.H.P.); ysjeong@dongguk.edu (Y.-S.J.); Tel.: +82-2-2260-3374 (Y.-S.J.)

Received: 4 September 2019; Accepted: 12 October 2019; Published: 15 October 2019

Abstract: Cloud computing services that provide computing resources to users through the Internet also provide computing resources in a virtual machine form based on virtualization techniques. In general, supercomputing and grid computing have mainly been used to process large-scale jobs occurring in scientific, technical, and engineering application domains. However, services that process large-scale jobs in parallel using idle virtual machines are not provided in cloud computing at present. Generally, users do not use virtual machines anymore, or they do not use them for a long period of time, because existing cloud computing assigns all of the use rights of virtual machines to users, resulting in the low use of computing resources. This study proposes a scheme to process large-scale jobs in parallel, using idle virtual machines and increasing the resource utilization of idle virtual machines. Idle virtual machines are basically identified through specific determination criteria out of virtual machines created using OpenStack, and then they are used in computing services. This is called the idle virtual machine–resource utilization (IVM–ReU), which is proposed in this study.

Keywords: Cloud Computing Service; Distributed Computing; Idle Virtual Machine; OpenStack; Computing Resource

1. Introduction

Generally, supercomputing and grid computing have been used to process data-intensive or complex problems in parallel. Most studies on large-scale job processing using supercomputing or grid computing have been focused on the development of task partition algorithms, such as K-means, particle swarm optimization, and cat swarm optimization. However, resources in networks or servers used in most companies or institutions are not used to the maximum, and become idle. Their mean operation rates have been measured at less than 20% to 30%. In addition, as much attention has been paid to the Fourth Industrial Revolution throughout the world, the importance of cloud computing that can analyze and process a large amount of unstructured data has also increased.

To accommodate exponentially increasing user requirements and efficiently use computing resources, cloud computing integrates a single physical resource into multiple logical resources or many physical resources into a single logical resource based on virtualization technology. Thus, studies on the use of cloud computing have been conducted to process large-scale jobs in a distributed manner. As described above, if computing resources are provided using logically created virtual machines based on a virtualization technique, the parallel processing of large-scale jobs, which is not possible in existing systems, can be achieved, and requirements of space and hardware costs can be reduced [1–12].

However, a survey by Amazon EC2 found that the mean use rate of virtual machines was 7.3% per week, and a survey by IBM found that 30% to 40% of all virtual machines held by cloud

computing service providers were idle for a long period of time. When virtual machines created through virtualization techniques to overcome the limitations of existing physical servers are not used for a long period of time, the unused computing resources are wasted [13–16].

Thus, this study proposes an idle virtual machine management scheme that provides additional service functions by using idle virtual machines within OpenStack, which is a cloud computing open-source project, as a form of infrastructure as a service that supports multiple platforms.

2. Related Works

Studies have been conducted on providing computing services to users by using virtual machines based on cloud platform environments. However, previous studies did not consider idle virtual machines when providing computing services to users; other studies reported that overhead cost occurred, and much time was wasted when the computing resources of idle virtual machines were assigned to other virtual machines.

2.1. CloudGC

Bo Zhang [13] used idle virtual machines in the OpenStack environment using the garbage collector principle that cleans the currently unused memory as one of the memory management techniques to provide computing resources to users. This method may have prevented wasting cloud resources as idle virtual machines were used, but it did not provide computing services to users using idle virtual machines.

2.2. Efficient Dynamic Resource Allocation

Praveenkumar [14] proposed a framework that processed data in parallel using Nephele, which was a cloud-based computing platform, to dynamically manage computing resources. This proposed framework removed virtual machines that were idle for a long period of time, and transferred the removed virtual machines to other virtual machines that needed additional computing resources. However, this framework requires a solution to the problem of taking a long time to search for and delete idle virtual machines that were neglected for a long period of time, as well as delivering additional computing resources to other virtual machines.

2.3. OpenStack Neat

Anton Beloglazov [17] proposed OpenStack Neat, which is an integrated dynamic virtual machine framework used to migrate minimally used or idle virtual machines to other virtual machines based on OpenStack, and to offload virtual machines to other hosts when overloads in virtual machines occur. It can increase the use of computing resources by monitoring idle virtual machines. However, if more than two global managers, which are used to determine the arrangement of virtual machines, are present at the same time due to service expansions, a collision may occur when deploying virtual machines. Thus, it is necessary to consider a circumstance where more than two global managers deploy virtual machines to a single host at the same time.

2.4. Optimized Virtual Resource Deployment

Anesul Mandal [18] proposed a scheme to recycle idle virtual machines through CloudSim, which was a framework for simulations of cloud computing services. The proposed scheme employed a broker concept, which helped users to book a virtual machine when needed, and recycled long-idle virtual machines assigned to users by a broker. However, virtual machines used in the study had the same specifications in the experiment, which did not consider various specifications of virtual machines.

2.5. Advanced Memory Reusing Mechanism

Gursharan Singhab [19] proposed a migration scheme that would reduce the data image size stored in host machines, and then migrate the data to other virtual machines. If the migrated data from an original host were then delivered, the images stored in the existing host's memory would be reused. This scheme contrasted with existing migration schemes that move a large amount of data between hosts to migrate virtual machines. However, because the above proposed scheme employs a pre-copy method that transferred the whole of the data first, and then transferred modified data later on to send a memory of virtual machine, which was present in the original host, it may cause overhead costs and affect the total time of the virtual machine migration.

The idle virtual machine–resource utilization (IVM–ReU) proposed in this study does not clean up the memory of virtual machines that are not used for a long time or migrate some computing resources to other virtual machines that require additional resources. This means that users choose to directly use the computing resources of idle virtual machines that are already crated, so there is no time issue or overhead to transfer computing resources.

Table 1 summarizes the idle virtual machine–resource utilization (IVM–ReU) and previous studies in comparison. The comparison criteria for each study were based on the framework used, whether idle virtual machines were considered to maximize resource utilization, whether virtual machines which is idle or unused for long periods of time were recycled, whether the specifications of various virtual machines were used, or whether providing a computing services capable of processing user-requested applications in parallel.

Table 1. Comparison of the idle virtual machine–resource utilization (IVM–ReU) with previous studies.

Paper Title	Framework	Idle Virtual Machine Considered?	Virtual Machine Recycle	Various Virtual Machine Specs	Computing Service Provided
CloudGC: Recycling Idle Virtual Machines in the Cloud	OpenStack	O	O	O	X
Efficient Dynamic Resource Allocation Using Nephele in a Cloud Environment	Nephele	O	X	O	X
OpenStack Neat: a framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack Clouds	OpenStack	O	O	O	X
Optimized Virtual Resource Deployment using CloudSim	CloudSim	O	O	X	X
Advanced Memory Reusing Mechanism for Virtual Machines in Cloud Computing	CloudSim	O	O	X	X
Our Proposed Scheme	OpenStack	O	O	O	O

3. Scheme of the IVM–ReU

The idle virtual machine–resource utilization (IVM–ReU) scheme that provides cloud computing services by selecting idle virtual machines present in OpenStack is shown in Figure 1. Figure 1-① shows that large-scale applications (i.e., jobs) submitted by requester (i.e., users) through OpenStack's Dashboard (i.e., IVM–ReU Website), are transferred to the IVM–ReU Manager. Figure 1-② shows the process of searching and selecting virtual machines that exist idle to handle requested tasks. Figure 1-③ shows that selected idle virtual machines are managed in a single pool, collecting static information such as the number of CPUs, total memory size, and dynamic information such as memory usage, the state of idle virtual machines. The information collected is transferred in the form of metadata to the IVM–ReU Manager as shown in Figure 1-④. The metadata of idle virtual machines allows IVM–ReU Manager to manage the state of idle virtual machines, which is based on which it designates some idle virtual machines as workers to handle tasks. Figure 1-⑤ shows the process of dividing and transferring a large-scale job into uniform-size tasks. These partitioned tasks are stored in the IVM–ReU Task

Saver before they are sent to workers. Figure 1-⑥ assigns split tasks to each worker in IVM–ReU Task Saver, and shows how each worker processes the tasks. Tasks that have been processed are sent to the IVM–ReU Task Saver as shown in Figure 1-⑦. Figure 1-⑧ shows the process of transferring the processed tasks from the IVM–ReU Task Saver to the IVM–ReU Manager. The transmitted tasks are merged into a single job as shown in Figure 1-⑨ and presented to the requester (i.e., user) via the IVM–ReU Web site (i.e., OpenStack Dashboard).

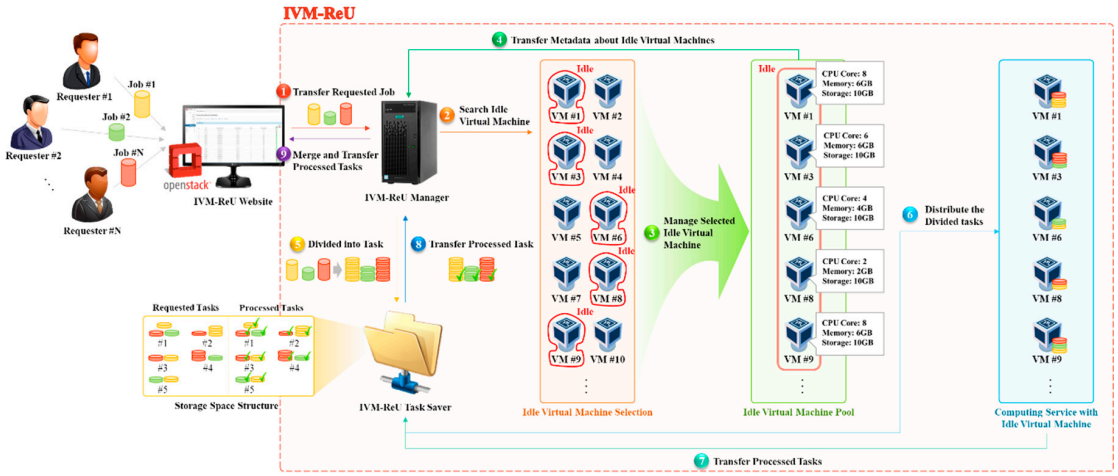

Figure 1. Basic operation diagram of idle virtual machine–resource utilization (IVM–ReU).

IVM–ReU largely consists of three parts: Idle Virtual Machine Selection, which searches for and selects virtual machines in an idle state among virtual machines generated in OpenStack; Idle Virtual Machine Pool, which collects dynamic and static information to manage selected idle virtual machines; and Computing Service with Idle Virtual Machine, which provides computing services by distributing the partitioned large-scale jobs into each of the idle-state virtual machines.

3.1. Idle Virtual Machine Selection

The Idle Virtual Machine Selection searches for and selects virtual machines that are in an idle state and have not been used by users for a long period of time among virtual machines generated in OpenStack. The criteria to determine the used or idle state of virtual machines are largely separated into three cases. These determination criteria are as follows. The higher the case number is, the more specific the determination criterion for an idle state is.

- Case 1. If a virtual machine is started up, it is in a used state; if a virtual machine is shut down, it is determined as being in an idle state.
- Case 2. If a virtual machine is started up, and users are logged into and using it, it is determined to be in a used state. However, if a virtual machine is started up, but users are not using it due to it being in a logged-out state, it is determined to be in an idle state.
- Case 3. If a virtual machine is started up, users are logged in, and the number of background processes is more than zero—that is, if a program is running in the background—it is determined to be in a used state. On the other hand, if users are logged into the virtual machine, but they are not using it—that is, zero programs are running in the background—it is determined to be in an idle state.

3.2. Idle Virtual Machine Pool

The static and dynamic information about idle virtual machines should be collected and managed to provide computing services based on idle virtual machines selected in the above-mentioned Idle Virtual Machine Selection process. Here, for static information, the virtual machine name, IP address,

the number of CPUs, and total memory size are included; for dynamic information, usable memory size, used memory, virtual machine state, and the number of allocated tasks are included.

IVM–ReU Manager, which globally manages virtual machines created inside OpenStack, requests static and dynamic information from virtual machines present in an idle state to collect static and dynamic information about idle virtual machines. A virtual machine that received the request collects and sends the static and dynamic information to IVM–ReU Manager in the format of metadata. The metadata delivered to IVM–ReU Manager are stored and collated. The attributes of the metadata are presented in Table 2.

Table 2. Metadata of Idle Virtual Machines for IVM–ReU.

Attribute	Description
Total memory	Total memory size (Megabyte, MB) assigned to the virtual machine
CPU	Number of all CPU cores assigned to the virtual machine
IP	IP address assigned to the virtual machine
Login id	The virtual machine user's id
Status	Login/Logout status information of virtual machine users
Process	Number of running processes by virtual machine user ID
Available memory	Available memory size (Megabyte, MB) of the virtual machine

The idle virtual machines and static and dynamic information about the machines are integrated and managed in the Idle Virtual Machine Pool. IVM–ReU Manager requests, collects, and manages information about each of the idle virtual machines continuously. Thus, changing dynamic information can be reflected immediately in the Idle Virtual Machine Pool so that users can cope with changing circumstances, such as the sudden use of idle-state virtual machines.

3.3. Computing Service with Idle Virtual Machine

Computing Service with Idle Virtual Machine partitions a requested large-scale job into tasks to provide computing services to users, and processes the tasks after distributing them to the selected idle virtual machines selected as workers. This study provides Sobel Edge Detection as a computing service, and the procedure performed by the Computing Service with Idle Virtual Machine is shown in Figure 2. Figure 2-① shows the process in which the requestor submits a large-scale job to the IVM–ReU Website (i.e., OpenStack Dashboard). Figure 2-② shows the process of evenly dividing the submitted job based on the task unit set by the requestor. Divided tasks are transferred and stored in the IVM–ReU Task Saver before they are assigned to the workers as shown in Figure 2-③. IVM–ReU Task Saver manages the requested and processed tasks separately. Figure 2-④ shows the process of assigning and processing tasks to workers. Figure 2-⑤ shows the process of transferring and storing the processed tasks to the IVM–ReU Task Saver. Figure 2-⑥ shows the process of transferring processed tasks to the IVM–ReU Manager. The transmitted tasks are merged into a large-scale job as shown in Figure 2-⑦. Figure 2-⑧ shows the process in which the IVM–ReU Manager provides the requestor with a large-scale job processed through the IVM–ReU Website.

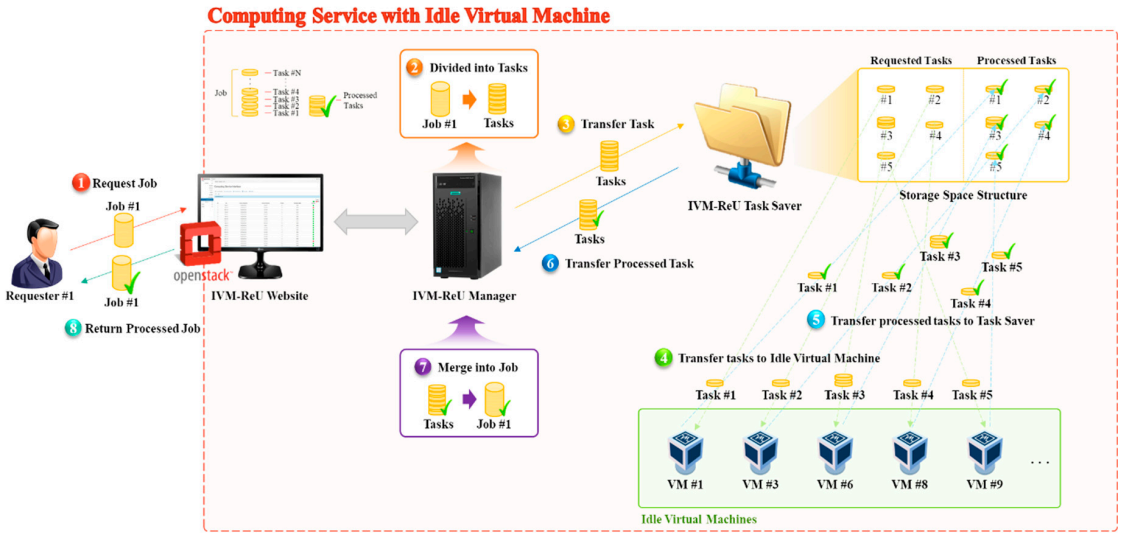

Figure 2. Computing service control flow in IVM–ReU.

4. Design of IVM–ReU

This study basically used Horizon, Cinder, Nova, Swift, Neutron, Glance, and Keystone services provided by OpenStack, and added the IVM–ReU feature that provides computing services to users through cloud computing. Figure 3 presents the overall structure of the used OpenStack and IVM–ReU architecture proposed in this study.

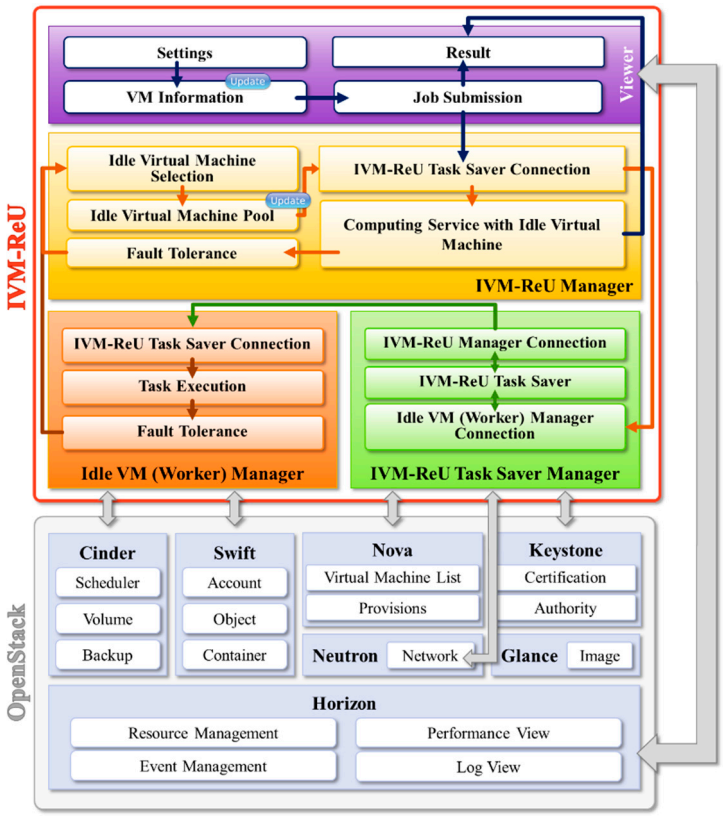

Figure 3. OpenStack and IVM–ReU architecture.

As verified in Figure 3, basic services provided by the OpenStack and IVM–ReU functions proposed in this study are closely connected. First, Nova, which provides computing services, is a core

function of OpenStack, which creates and manages instances by provisioning computing resources as needed to users. Swift is an object storage service, which consists of account and container, whose data are managed via a database, and object, which is directly stored in a storage space. Here, object performs tasks, such as the searching, downloading, and uploading of actual data and files, and container is a unit of storage space that can search objects or create and delete containers. In addition, account searches containers and manages user accounts. Glance manages virtual machine images to be used in various hypervisors, and provides image services that manage operating systems to be run in each of the virtual machines. KeyStone, which is an authentication service, manages the use of resources, such as computing, networking, and storage, inside the physical server through user authentication. Neutron provides network services for other OpenStack services, such as assigning IP addresses to created instances and managing them. Cinder, which is a block storage service, can create, add, and delete a volume, which is a storage space, in instances created through Nova, and it updates the modified volume information in the database. The created volumes may be allocated to single or multiple instances. Horizon is an OpenStack Dashboard service for users to request a task easily and rapidly through the web user interface (UI) and monitor the process and results in relation to basic services of OpenStack, such as resource and event management services such as instance creation, deletion, and management, performance, and log viewing [20].

The IVM–ReU architecture proposed in this study is intended to provide computing services to users who want to process a large-scale job based on the basic services of OpenStack. It largely consists of Viewer, IVM–ReU Manager, Idle VM (Worker) Manager, and IVM–ReU Task Saver Manager. Viewer visualizes a procedure that provides computing resources and services to users as a form of web UI by using the Horizon service of OpenStack, consisting of Setting, which represent basic settings such as user registration, login history, application registration and deletion, etc. to provide computing services that process a large-scale job in parallel through IVM–ReU; VM Information, which displays static, dynamic information and the states about all virtual machines created in OpenStack or idle virtual machines classified based on the idle-state determination criteria; Job Submission, by which users submit their jobs; and Result, which shows the processed task results.

IVM–ReU Manager, which provides computing services to users by selecting and managing idle virtual machines from the created virtual machines, consists of Idle Virtual Machine Selection, Idle Virtual Machine Pool, Computing Service with Idle Virtual Machine, Fault Tolerance, and IVM–ReU Task Saver Connection. Idle Virtual Machine Selection classifies virtual machines by searching for machines in an idle state. Idle Virtual Machine Pool collects and manages static and dynamic information, such as IP addresses, available memory size, and the status of idle virtual machines selected, to provide computing services to users through the use of cloud computing resources. Computing Service with Idle Virtual Machine designates a portion of the virtual machines present in an idle state as workers to be used in computing services, and divides large-scale jobs in proportion to the number of selected workers, thereby providing computing services after allocating the divided tasks to each of the workers. Job processing status and related information about jobs that are processed by each worker are displayed for users through Job Status in Result of Viewer. In addition, it merges processed tasks into a single large-scale job, and provides processed outcome files and detailed information to users through Result in the Viewer. Fault Tolerance resolves when an idle virtual machine that is processing tasks as a worker changes to a used state or when processing tasks are lost due to errors such as a network outage. IVM–ReU Task Saver Connection is a network connection function needed to transfer data to IVM–ReU Task Saver to store divided tasks or merge tasks processed by workers into large-scale jobs.

Idle VM (Worker) Manager consists of Task Saver Connection, which is a network connection function needed to transfer tasks processed by workers to IVM–ReU Task Saver or receive divided tasks from IVM–ReU Task Saver; Task Execution, which processes tasks allocated to each worker; and Fault Tolerance, which solves cases when a worker's state is changed to a used state due to a user's

login and use in the middle of task processing or when a loss is generated in a running task due to errors, such as memory overflow.

IVM–ReU Task Saver Manager consists of IVM–ReU Manager Connection, which communicates with IVM–ReU Manager to distribute divided tasks and collect processed tasks; Idle VM (Worker) Manager Connection, which communicates with Idle VM (Worker) Manager to transfer processed tasks or receive allocated tasks; and IVM–ReU Task Saver, which is a space that stores divided tasks to be allocated to each worker to provide computing services to users or tasks processed by workers before the tasks are collected by IVM–ReU Manager.

5. Implementation of IVM–ReU

When the proposed IVM–ReU is executed, four panels are displayed in the left panel of the OpenStack web page: Setting, VM Information, Job Submission, and Result, as shown in Figure 4.

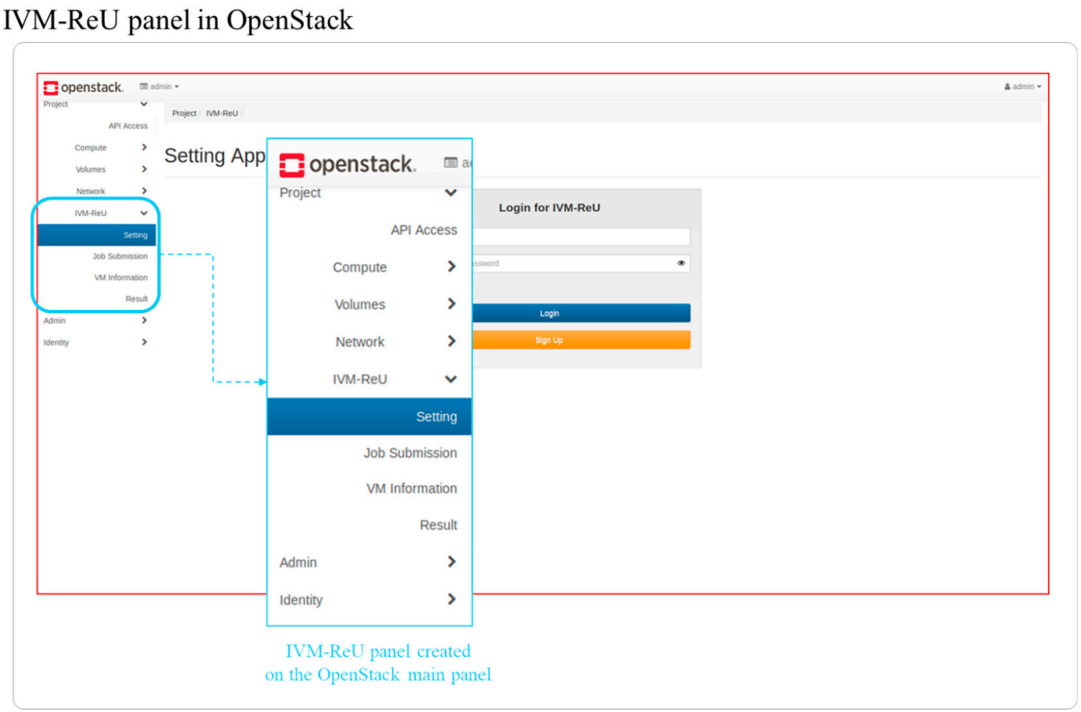

Figure 4. IVM–ReU panel in OpenStack.

First, the Setting consists of a Login History, which shows the login history of requesters accessing the IVM–ReU panel in OpenStack, and an Application List that allows users to add and remove required applications before submitting a job. The initial page of Setting shows the requester a separate login window in addition to the OpenStack account to manage the request so that only the person requesting the IVM–ReU computing service can create and delete applications and submit a large-scale job, as shown in Figure 5-① below. For requesters who do not have access to IVM–ReU, create an account as shown in Figure 5-②. On the other hand, a requester with IVM–ReU access can check the Login History and Application List by accessing the Setting page. Login History is composed of IP address, logged-in time, logged-out time, and status indicating login success as shown in Figure 5-③. Here, status shows success if the login is successful and fail if the login fails due to incorrect ID or password. Figure 5-④ is an Application List, which shows a list of applications registered by the requester.

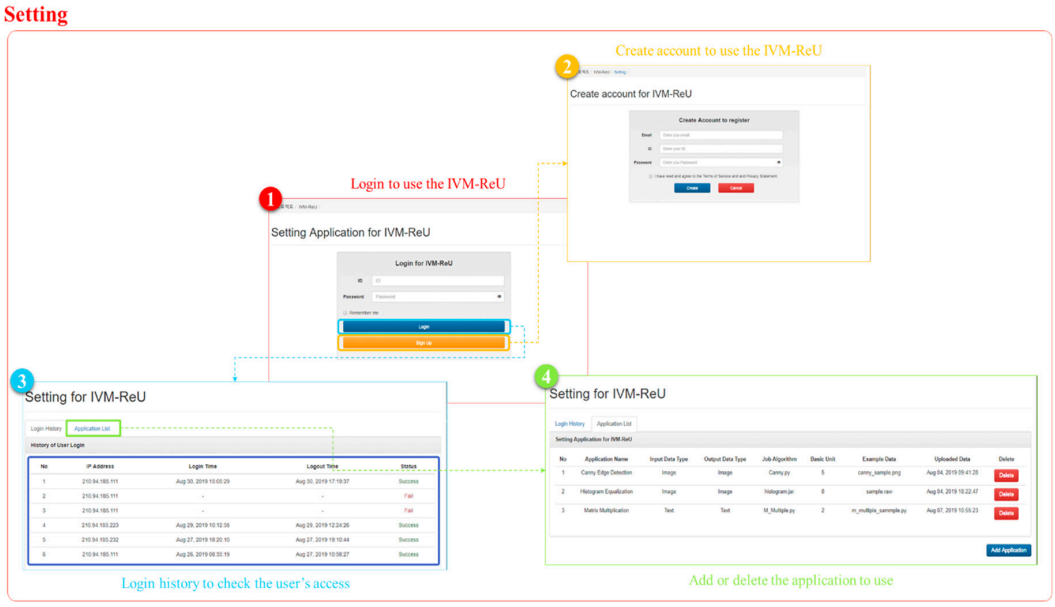

Figure 5. Setting view to register applications and users to use IVM–ReU.

If the requester wants to process another large-scale job except the previously registered application in the Application List, click Add Application button at the bottom to add the Add Application form to register additional applications as shown in Figure 6-①. The Add Application form is consists of the user id that registers the application, the application name, an additional description of the application, the format of the data to be entered into the application, the format of the data output through the application, the application's executable file, the default task unit, and the sample input data to be used by the application. Here, the mandatory items are marked in red. When Sobel Edge Detection is newly registered as an application, Sobel Edge Detection is added to the Application List as shown in Figure 6-②. Figure 6-③ shows that registered application is deleted from Application List by clicking the delete button.

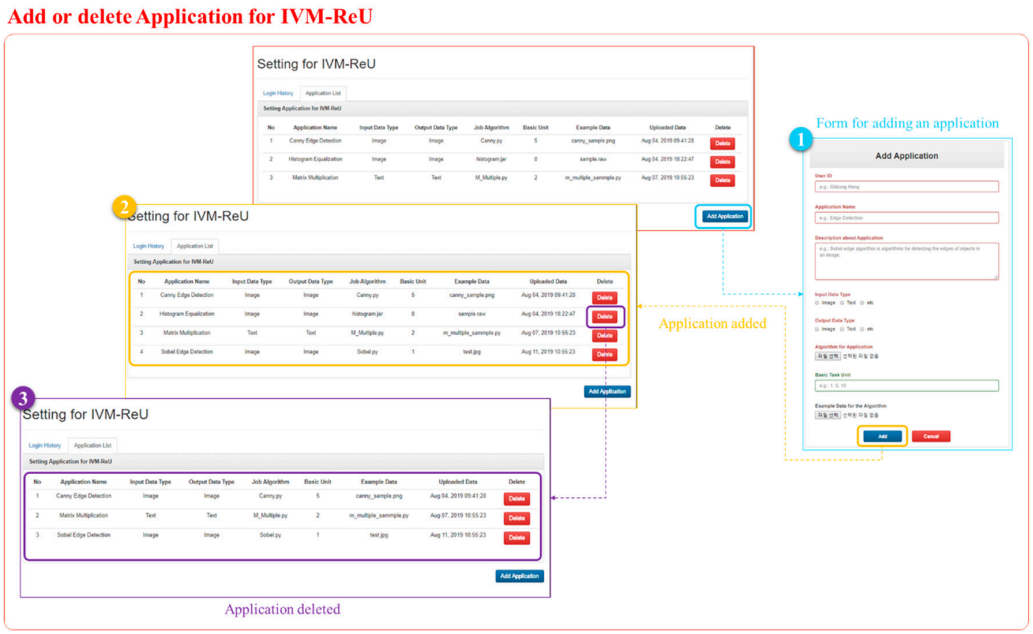

Figure 6. Adding and deleting applications for IVM–ReU.

VM Information consists of an ALL tab that displays information about all virtual machines created in OpenStack, and an IDLE tab that displays information about idle virtual machines. As shown in the following Figure 7, both the ALL and IDLE tabs display information about virtual machines in a table format. The data in the table is sorted in descending order based on the number of CPUs, the total memory size, and the Memory Usage, as shown in Figure 7 to prioritize the virtual machines with good specifications.

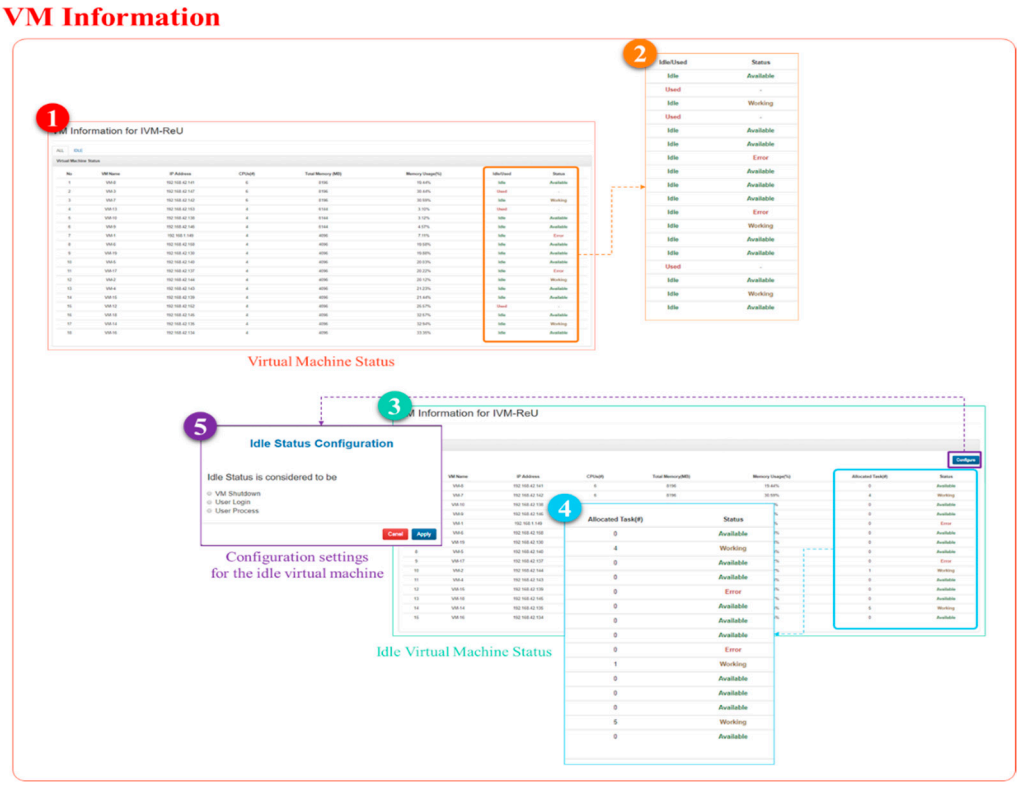

Figure 7. Status information view of virtual machine.

The table in the ALL tab is consists of static information such as the virtual machine names, IP addresses of virtual machines, number of CPUs, and total memory (static unit of memory is megabytes; MB), and dynamic information, such as memory usage (%), idle/used, and status of idle virtual machine as shown in Figure 7-①. The Idle / Used status of virtual machine is green when idle and red when other users are in use, as shown in Figure 7-②. Status Information for a virtual machine appears only if it is in an idle state. Figure 7-② shows the status of only the virtual machines in the idle state, where Status is divided into Available status, where tasks are not allocated to idle virtual machines; Working status, where tasks are allocated and are currently being processed in idle virtual machines; and Error status, where errors have occurred in idle virtual machines.

The table of IDLE tab is consists of static information such as name of virtual machine, IP address of virtual machine, number of CPUs, and total memory (static unit of memory is megabytes; MB), and dynamic information, such as memory usage (%), number of tasks assigned to the virtual machine and the status of the virtual machine. Status is the same as Status in the ALL tab. As shown in Figure 7-④, only the status of the idle virtual machine is displayed as Available, Working, and Error. The number of tasks assigned to a virtual machine shows a nonzero number only for virtual machines whose status is Working. In the IDLE tab, the Configuration button is present to set up the criteria to determine whether a virtual machine's state is idle, which is different from the All tab as shown in Figure 7-⑤. Configure consists of three criteria: VM Shutdown, User Login, and User Process. Depending on the

criterion set by a requester, the determination criteria of the idle state changes, resulting in a different list of idle-state virtual machines displayed in the IDLE tab.

Job Submission submits users' requested tasks to IVM–ReU, which is an added function of OpenStack. In Figure 8-①, the requester clicks the select button at the bottom right of the application to submit the work to the desired application. Figure 8-② shows the requester uploading input data to the selected application. Information about the application is written by the user who registered the application. After the requester uploads the input data and presses the Next button, the process of selecting workers to process a large-scale job in parallel appears as shown in Figure 8-③. Here, the requestor can check the number of CPUs, the total memory size, the memory usage, and the current state of the idle virtual machine for each idle virtual machine. If Status is Error, it will appear in the worker table, but the checkbox will be disabled so that the user cannot select the virtual machine. Figure 8-④ shows how the requestor enters the number of tasks to assign to selected workers. Figure 8-⑤ shows the information submitted by the requester. When the requestor clicks the Next button, a popup window related to job submission is created. Figure 8-⑥ shows the requester that the job was submitted successfully.

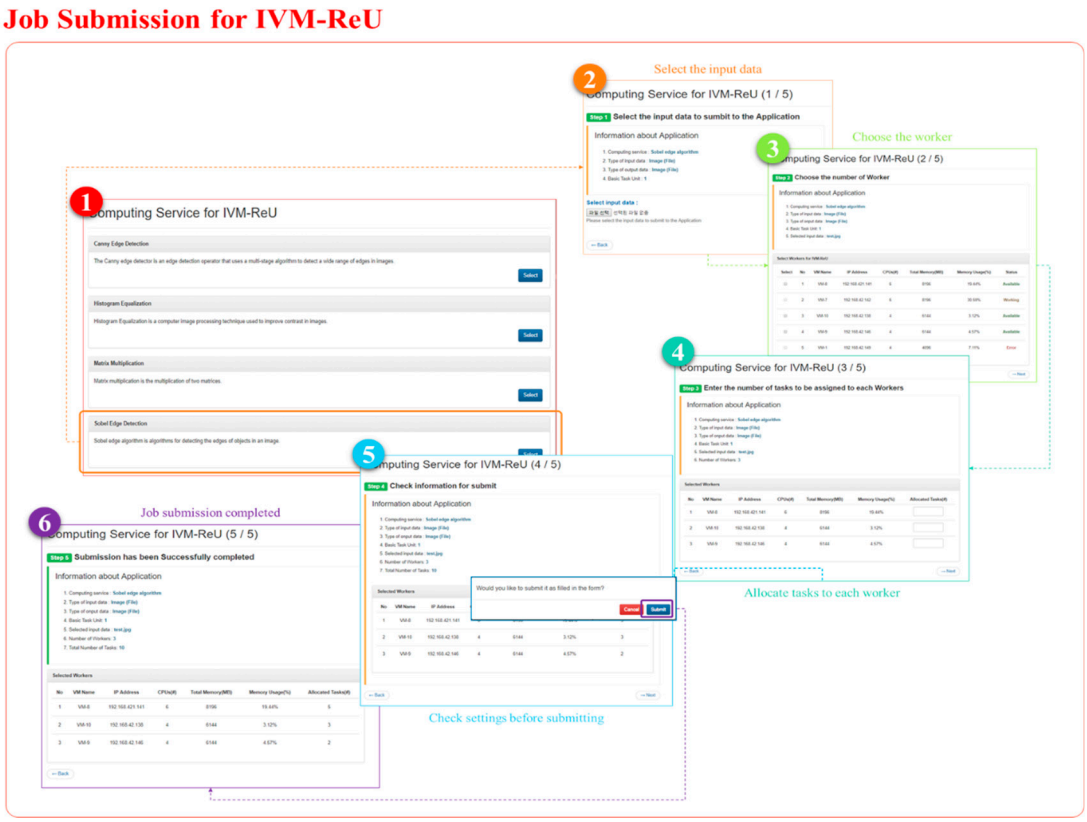

Figure 8. Submit jobs to receive compute services over IVM–ReU.

The job is evenly divided based on the contents set by the requester, and the divided tasks are transferred and assigned to each of the idle-state virtual machines. In this study, the Sobel Edge Detection algorithm was added as an application, and image files, which will be executed through the application, were submitted as shown in Figure 8 below.

The submitted tasks are divided into basic units that are set when creating an application for task division, and tasks of uniform size based on the number of tasks that the user submits and sets. In this case, the application is a repetitive a large-scale job that can be split, and even if the large-scale job is divided into tasks of equal size, there is no data dependency between the tasks. Partitioned tasks are distributed to each virtual machine based on the number of tasks that the requester previously entered

to assign to each virtual machine. Each virtual machine receives the relevant information such as input data (task) to be processed with the application and performs the task, and the processed results are merged by the IVM–ReU Manager.

Results are divided into Job Status tab and Job Results tab. First, Job Status tab is a page showing the processing status of submitted jobs, as shown in the following Figure 9-①.

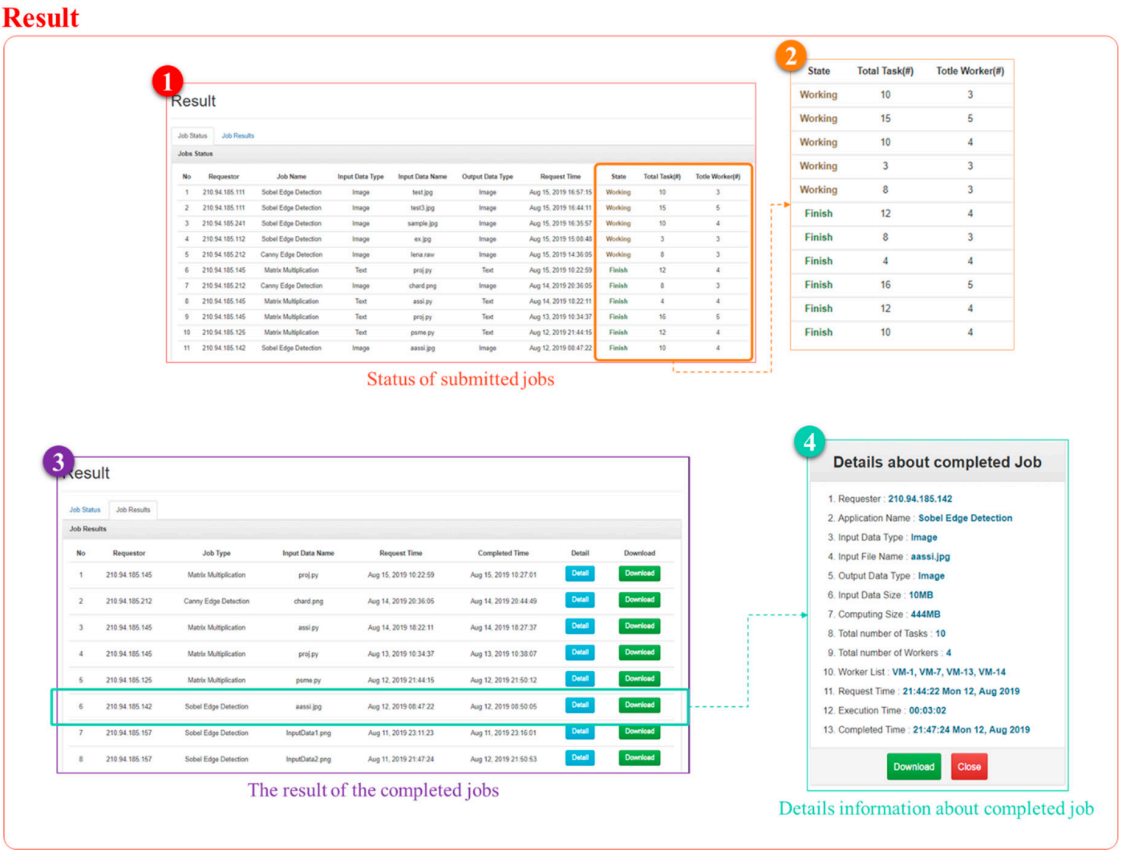

Figure 9. Result view to provide detailed information to the requester.

Job Status shows the requestor in table form information about the requestor's IP address, submitted job name, input data type, input data name, output data type, time the job was requested, job execution status, total number of tasks, and total number of workers. Here, the work processing status is divided into Working, which is processing the work, and Finish, which is completed, as shown in Figure 9-②.

Finally, Result displays the resulting files and detailed descriptions of the jobs that were previously completed in the form of a table and history. The Result table is consists of the requester's IP address, the type of submitted job, the name of the input data file, the time the job was requested, the time the job was completed, Detail button, which provides detailed information about the job, and download button for downloading the completed work file, as shown in Figure 9-③. When the requestor presses the download button, the processed work file is downloaded to the requester's computer. In this paper, an image file with Sobel Edge Detection is downloaded. If you click the Detail button to get detailed information about the processed job, a page with detailed information about the job will appear as shown in Figure 9-④ below. The Detail view allows the requestor to view information such as the requestor's IP address, the application name used to perform the task, input data type, input data file name, output data type, input data size, computing size, total number of split tasks, number of virtual machines used to process the task, number of virtual machines selected as worker, total execution time, completion time, and so on. Requester can download the output file of a job that has been processed

from the Result page, but requester can also view the details in the Detail window and download the output file of the job that has been processed.

6. Performance Evaluation

The IVM–ReU proposed in this paper conducted a performance evaluation on whether idle virtual machines in OpenStack provide computing services efficiently to the requester for a large-scale job. OpenStack used for performance evaluation is version 3.19.0, which is Stein, the latest release of OpenStack [21], and hardware specifications used as IVM–ReU Manager are 72 CPU cores, 123 GB of memory size, 200 GB of storage. In addition, the hardware specification of the server where several virtual machines are created includes 32 CPU cores, 134 GB of memory size, and 100 GB of storage; 8 CPU cores, 16 GB of memory size, and 100 GB of storage. The specifications of the virtual machines created inside each server consisted of four types, largely 2 CPU cores, 2 GB of memory size; 4 CPU cores, 4 GB of memory size; 4 CPU cores, 6 GB of memory size; 6 CPU cores, 8 GB of memory size. Among idle virtual machines created within OpenStack, the criteria to be selected as workers were from high-performance idle virtual machines to workers in descending order. The computing service was used as a Sobel Edge Detection algorithm mainly used in image processing, and the task unit for evenly dividing the large-scale job was set to 1 MB. Because 1 MB is the common size of JPEG files generated through smartphones, digital cameras, etc., the performance evaluation designated task unit as 1 MB, considering that the file to be submitted to one of the IVM–ReU's computing services, the file to be submitted to the Sobel Edge Detection, is JPEG.

Figure 10 compares the execution time when requester selects two to ten workers among idle virtual machines to process a large-scale job with different computing size through IVM–ReU computing service. The specifications of the idle virtual machines used in the performance evaluation consist of 6 CPU cores, and 8 GB of memory size (2 virtual machines); 4 CPU cores, and 6 GB of memory size (2 virtual machines); 4 CPU cores, and 4 GB of memory size (6 virtual machines), and the best specification is selected first. For example, if only two of the idle virtual machines are to be selected as workers, the selected specifications are two idle virtual machines with 6 CPU cores and 8GB of memory size. As Figure 10 shows, when the computing size is 1MB, two workers must be used to process the task, resulting in minimal execution time. When the computing size is 72 MB and 154 MB, if requester increase the number of workers to 3, 4, or 5 to process the job, the execution time gradually decreases, but if requester use more than 6 workers, it cannot see much difference. Especially when the computing size is more than 383 MB, there is a big difference in the execution time when selecting 2 workers and when selecting 8 workers. However, the execution time is similar when requester select 9 or 10 workers and when requester select 8 workers. Therefore, it can be seen that there are several workers that are most suitable for processing a large-scale task according to various computing sizes.

Figure 11 compares the execution time of a large-scale job with IVM–ReU applied to OpenStack and a large-scale job with one VM without applying IVM–ReU to OpenStack. IVM–ReU processes a large-scale job using up to 10 idle virtual machines, and the execution time for With IVM–ReU shown in Figure 11 is the average of execution times for 2 to 10 idle virtual machines derived from Figure 10 above. In addition, the specifications of the virtual machine used in the performance evaluation with "without IVM–ReU" is 6 CPU cores and 8 GB of memory size. Comparing "Without IVM–ReU" with "With IVM–ReU", the difference between "Without IVM–ReU" and "With IVM–ReU" is not clear if the computing size is 1 MB, but if the computing size is greater than 72 MB, the gap between "With IVM–ReU" and "Without IVM–ReU" is increased rapidly. T In other words, to process jobs or applications with large computing size, it is possible to effectively reduce execution time by using multiple virtual machines that are idle through IVM–ReU rather than using a single resource.

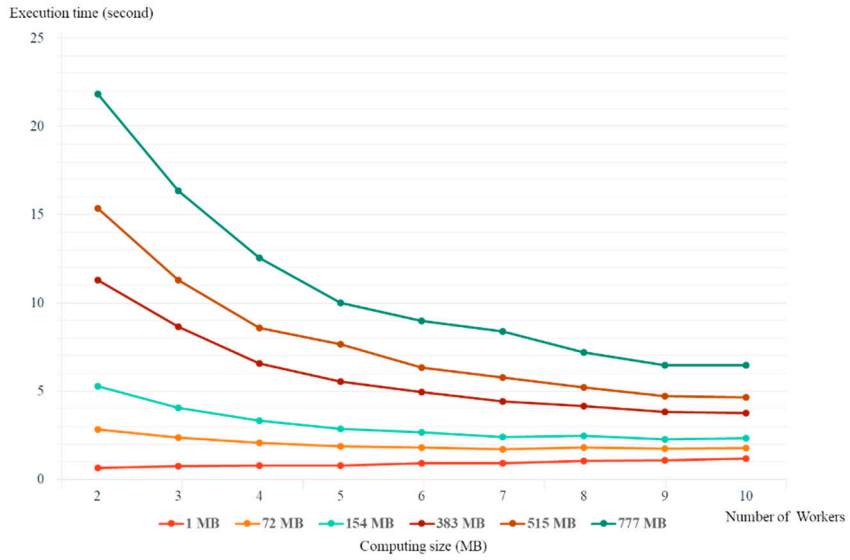

Figure 10. Comparison of execution time as the number of workers increases.

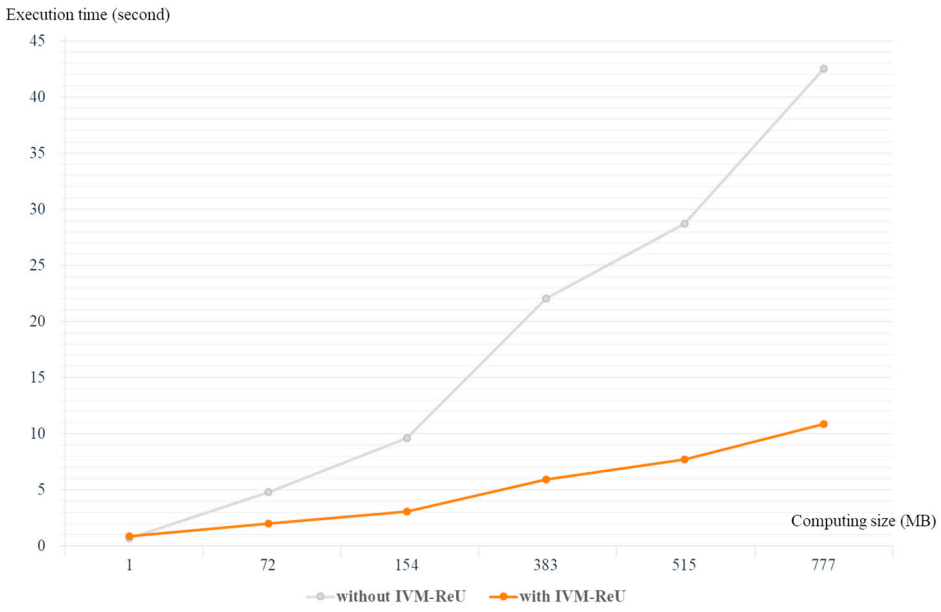

Figure 11. Comparison of execution time as the computing size increases.

7. Conclusions

This study proposes a scheme by which idle-state virtual machines are selected based on specific criteria out of virtual machines created through OpenStack, and computing services that can be handle applications (i.e., a large-scale job) in parallel are provided to users by using the selected idle virtual machines, which has not been proposed before in studies using OpenStack. By using the proposed IVM–ReU architecture and the three determination criteria that define whether the state of the created virtual machine is used or idle, a requester can easily request a job through the web UI provided by the OpenStack Dashboard service, and check the job processing in real time, the resulting files processed by the workers, and detailed information about the results. In addition, a large-scale job is evenly divided into tasks, which are then distributed based on the number of designated workers through IVM–ReU Manager. Prior to allocating the divided tasks to each of the workers, tasks are transferred and stored in IVM–ReU Task Saver. The stored tasks are assigned to each of the designated idle virtual machines

to be processed, and processed tasks are collected as a single result through IVM–ReU Manager via IVM–ReU Task Saver, which is then provided to the user.

As described above, the use of computing resources can be raised, thereby minimizing wasted computing resources by adding the IVM–ReU function to OpenStack, which reuses idle virtual machines that would be unused and neglected otherwise, and additional computing services can be provided to users.

Although only Sobel Edge Detection was used as a computing service for the IVM–ReU architecture proposed in this study, our future study will seek a method to provide various other computing services to users for large applications. In addition, although the user selected the worker himself and requested processing for the large-scale application by specifying the number of tasks to be assigned to each selected worker, the research will be carried out to automatically distribute a large-scale job or application to each worker by predicting with the number of workers and tasks suitable for each computing size in the IVM–ReU scheme.

Author Contributions: Conceptualization, J.J.; methodology, J.J.; software, J.J.; validation, J.J.; formal analysis, J.J.; investigation, J.J.; writing—original draft preparation, J.J.; writing—review and editing, J.J., J.H.P., Y.-S.J.; supervision, J.H.P., Y.-S.J.; project administration, J.H.P., Y.-S.J.; funding acquisition, J.H.P., Y.-S.J.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1A2C1088383).

Acknowledgments: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1A2C1088383).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kim, K.; Jang, J.; Hong, J. Loan/Redemption Scheme for I/O performance Improvement of Virtual Machine Scheduler. *Smart Media J.* **2016**, *5*, 18–25.
2. Rosenblum, M.; Garfinkel, T. Virtual machine monitors: Current technology and future trends. *IEEE Comput.* **2005**, *38*, 39–47. [CrossRef]
3. Kemchi, S.; Zitouni, A.; Djoudi, M. AMACE: Agent based multi-criterions adaptation in cloud environment. *Hum. Cent. Comput. Inf. Sci.* **2018**, *6*, 26. [CrossRef]
4. Ha, N.; Kim, N. Efficient Flow Table Management Scheme in SDN-Based Cloud Computing Networks. *J. Inf. Process. Syst.* **2018**, *14*, 228–238.
5. Baek, S.; Park, S.; Yang, S.U.; Song, E.; Jeong, Y. Efficient Server Virtualization using Grid Service Infrastructure. *J. Inf. Process. Syst.* **2010**, *6*, 553–562. [CrossRef]
6. Kumar, Y.; Sahoo, G. An Improved Cat Swarm Optimization Algorithm Based on Opposition-Based Learning and Cauchy Operator for Clustering. *J. Inf. Process. Syst.* **2017**, *13*, 1000–1013.
7. Wu, M.; Sun, X. Memory Conscious Task Partition and Scheduling in Grid Environments. In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, Washington, DC, USA, 8 November 2004.
8. Fahim, Y.; Rahhali, H.; Hanine, M.; Benlahmar, E.; Labriji, E.; Hanoune, M.; Eddaoui, A. Load Balancing in Cloud Computing Using Meta-Heuristic Algorithm. *J. Inf. Process. Syst.* **2018**, *14*, 569–589.
9. Moon, Y.; Yu, H.; Gil, J.; Lim, J. A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments. *Hum. Cent. Comput. Inf. Sci.* **2017**, *7*, 28. [CrossRef]
10. Yi, G.; Heo, Y.; Byun, H.; Jeong, Y. MRM: Mobile resource management scheme on mobile cloud computing. *J. Ambient Intell. Humaniz. Comput.* **2018**, *9*, 1245–1257. [CrossRef]
11. Kim, B.; Byun, H.; Heo, Y.; Jeong, Y. Adaptive Job Load Balancing Scheme on Mobile Cloud Computing with Collaborative Architecture. *Symmetry* **2017**, *9*, 65. [CrossRef]
12. Kim, H.; Han, J.; Park, J.H.; Jeong, Y. DIaaS: Resource Management System for the Intra-Cloud with On-Premise Desktops. *Symmetry* **2017**, *9*, 8. [CrossRef]
13. Zhang, B.; Dhuraibi, Y.; Rouvoy, R.; Paraiso, F. Lionel Seinturier. CloudGC: Recycling Idle Virtual Machines in the Cloud. In Proceedings of the 2017 IEEE International Conference on Cloud Engineering, Vancouver, BC, Canada, 4–7 April 2017.

14. Praveenkumar, V.P.; Sujatha, D.N.; Chinnasamy, R. Efficient Dynamic Resource Allocation Using Nephele in a Cloud Environment. *Int. J. Sci. Eng. Res.* **2012**, *3*, 1–5.
15. Huh, J.; Seo, K. Design and test bed experiments of server operation system using virtualization technology. *Hum. Cent. Comput. Inf. Sci.* **2016**, *8*, 1–21. [CrossRef]
16. Lim, J.B.; Yu, H.C.; Gil, J. An Intelligent Residual Resource Monitoring Scheme in Cloud Computing Environments. *J. Inf. Process. Syst.* **2018**, *14*, 1480–1493.
17. Beloglazov, A.; Buyya, R. OpenStack Neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds. *Concurr. Comput. Pract. Exp.* **2015**, *27*, 1310–1333. [CrossRef]
18. Mandal, A.; Karmakar, K. Optimized Virtual Resource Deployment using CloudSim. *Int. J. Innov. Eng. Technol.* **2015**, *5*, 1–12.
19. Singhab, G.; Behalab, S.; Taneja, M. Advanced Memory Reusing Mechanism for Virtual Machines in Cloud Computing. In Proceedings of the 3rd International Conference on Recent Trends in Computing 2015, Ghaziabad, India, 12–13 March 2015.
20. OpenStack Docs Overview. Available online: https://docs.openstack.org/liberty/install-guide-ubuntu/overview.html (accessed on 17 January 2019).
21. OpenStack Releases. Available online: https://releases.openstack.org/ (accessed on 4 August 2019).