A Novel Framework for Resource Orchestration in OpenStack Cloud Platform

Afaq Muhammad[1], Wang-Cheol Song[2]
[1]Department of Computer Science and IT, Sarhad University of Science and IT
Peshawar Pakistan
[email: afaq24@gmail.com]
[2]Department of Computer Engineering, Jeju National University
Jeju, South Korea
[e-mail: kingiron@gmail.com]
*Corresponding author: Wang-Cheol Song

Received November 6, 2017; revised March 6, 2018; revised May 29, 2018; accepted June 5, 2018; published November 30, 2018

Abstract

This work is mainly focused on two major topics in cloud platforms by using OpenStack as a case study: management and provisioning of resources to meet the requirements of a service demanded by remote end-user and relocation of virtual machines (VMs) requests to offload the encumbered compute nodes. The general framework architecture contains two subsystems: 1) An orchestrator that allows to systematize provisioning and resource management in OpenStack, and 2) A resource utilization based subsystem for vibrant VM relocation in OpenStack. The suggested orchestrator provisions and manages resources by: 1) manipulating application program interfaces (APIs) delivered by the cloud supplier in order to allocate/control/manage storage and compute resources; 2) interrelating with software-defined networking (SDN) controller to acquire the details of the accessible resources, and training the variations/rules to manage the network based on the requirements of cloud service. For resource provisioning, an algorithm is suggested, which provisions resources on the basis of unused resources in a pool of VMs. A sub-system is suggested for VM relocation in a cloud computing platform. The framework decides the proposed overload recognition, VM allocation algorithms for VM relocation in clouds and VM selection.

Keywords: Orchestration, OpenStack, resource allocation, VM migration

1. Introduction

In the framework of cloud computing, resource provisioning is the method of assigning networking, compute, and (ultimately) energy resources to a cluster of applications, in a way that propose to jointly fulfill the performance necessities of the users of the cloud resources, the applications, and the infrastructure suppliers. The requirements of the suppliers revolve around operational and effective resource deployment within the limitations of SLAs with the cloud users. Operational resource deployment is usually achieved through virtualization tools, which enable numerical pooling of resources through customers and applications. The goals of the cloud users are subjective to the scaling of existing resources as a concequence of changing application demands, their availability and application performance [1].

In this study, an instrumentation framework for resource provisioning and organization is suggested, which allows to automate resource organization and provisioning for users across network and clouds by: 1) manipulating APIs delivered by the cloud supplier in order to compute and allocate/control/manage storage resources; 2) networking with SDN controller to obtain the details of the existing resources, and inculcating the rules/changes to organize the network dependent on the cloud service necessities. For resource distribution, a proposed algorithm distributes resources on the basis of unused resources in a pool of VMs. Moreover, a service perception model is used, which is supplied to the orchestrator for provisioning the resources dynamically and analytically based on the necessities of the demands made by remote end-user.

To that end, in this paper, we present an integrated orchestration of cloud and network resources for on-demand resource provisioning to meet dynamic service requirements. Our proposed orchestration framework manages network resources through interaction with ONOS SDN controller [28], and handles cloud resources through interaction with OpenStack [16]. The orchestrator also communicates with a monitoring module to collect cloud and network infrastructure-related information stored in an internal database server. In addition, we use a service abstraction model [23], which is delivered to the orchestrator for provisioning the resources systematically and dynamically.

Recently, the power depletion of cloud computing has enhanced, which is reliant on the resource usage, particularly the CPU usage. The drawback of power depletion stimulated several researchers all over the world to study the power depletion of CPU [2], network [3], etc. Therefore, service suppliers need to rigorously manage with the physical resource to guarantee an ability of high QoS without violating SLA. In infrastructure as a service (IaaS), the abundantly loaded aggressive alliance of VMs or physical machine (PM) on the same PM is the source of SLA desecration. Therefore, a fit procedure is required by the cloud suppliers to achieve the resource in a lively way as to guarantee QoS for the customers. Henceforth, we consider that typically common process that initiates an overloaded machine is a damaging alliance that can be avoided by making live relocation at a suitable time, as along with taking into consideration the cost of transfer.

The main test is that which VM should be relocated and where the relocated machine should be placed. Most studies emphasis on resource use to make choice for relocation action [4, 5], but in this case relocation action may result in performance deterioration if not controlled suitably during the relocation process. Altered VMs have dissimilar workload features and arrangements [6] that lead to diverse relocation costs.

In this study, a sub-system is suggested for vibrant VM relocation in a cloud platform. The structure implements the suggested overload recognition, VM allocation and VM selection algorithms for vibrant VM relocation. Moreover, with the support of experiments, it is displayed that the suggested algorithms outdo the procedures that are considered for the determination of assessment [7].

In this paper, the following research problems are investigated:

- Abstraction of service requirements. In order to make orchestrator easily understand the service requirements, they required to be abstracted.
- Allocation of resources. In order to fulfil the requirements of the requested service, an element is needed to allocate resources.
- Management of network and cloud resources. Automation of resource provisioning and management for users across networks and clouds.
- Initiation of VM migration. It is important to determine a suitable time for migrating VMs from an overloaded compute node.
- Selection of VMs for migration. After overload detection, only those VMs need to be selected which are utilizing most of the resources of a compute node.
- VMs placement. It is necessary to select the most efficient compute node for VM allocation.

The following points have been outlined in order to address the aforementioned issues:

- Deployment of a physical testbed that comprises an OpenStack platform, implementing the orchestrator and a VM migration framework.
- Delivering the abstracted service requirements to the orchestrator by designing and implementing a service abstraction communicator.
- Develop and implement algorithms for resource allocation, overload detection, VM selection for migration, and placement on the most efficient compute node.

The rest of the paper is organized as follows. In Section 2, related work is presented. In Section 3, the orchestration framework, its major components, and the performance analysis of the proposed resource allocation algorithm are presented. In Section 4, a framework for VM migration working in parallel with the orchestrator, its components, and the performance analysis of the three proposed algorithms are discussed. The paper concludes in Section 5.

2. Related Work

Resource provisioning [8] includes the management, withdrawing, design,and manipulation of cloud resources, i.e., storage, compute, and network, to fulfil client needs, while imitating to operational intents of the cloud service providers at the same time. We discuss that cloud arrangement is extremely difficult. Primarily, as numerous novel proposals [9, 10, 11, 12] have expressed, cloud management is essentially intricate due to the scale, heterogeneity, simultaneous infrastructure and user services, that share a mutual set of physical resources. Secondly, arrangements of several resource types interrelate with each other e.g. the sites of VMs have an outcome on storage location, which in turn affect bandwidth usage outside and within to a data center. Thirdly, cloud resources have to be prearranged in a manner that not only realizes operational objectives of provider, but also guarantees that the client SLAs can be frequently met as runtime circumstances change.

Resource orchestration in cloud system is dissimilar from the customary network systems and the characteristics of the operators, cloud's services, and architecture yields some

necessities in resource management which should be sensibly addressed [29, 30]. It includes discovery, allocation, and monitoring of physical resources, such as RAM, CPU cores, network bandwidth, and disk space.

Resource allocation is the process of allocating available resources to the required cloud applications over the Internet. An improper allocation of resources may lead to ceasing of services. Recently, the cloud computing resource distribution model and algorithm have been extensively studied. Among numerous resource allocation plans, [13] recommended a virtual resource organization model that assigns resources by means of the resource reserve plan and division, while assuring the usefulness for the users to consume the virtual resources. In [14], authors suggested a virtual resource distribution mechanism based on usefulness, but the authors only considered single dimension as the CPU. In this study, the difficulty of virtual cloud resource distribution has been addressed by increasing it two dimensions, i.e., memory and CPU.

Multi-Criteria Decision Analysis (MCDA) [31] is a famous concept that helps in selecting a solution among several options by assessing multiple criteria, specifically in decision making. In [24], authors have used MCDA to select an optimum alternative among the available resources. A physical machine which has the available resources to meet the requested service requirements, is selected among several other physical machines. More precisely, resource provisioning is disintegrated into independent tasks, where each task is carried out on a physical machine in a data center. Based on the statistics retrieved from each physical machine, the configurations are carried out through MCDA for making a decision. If there is no physical machine in the data center that can offer its resources then no action is taken. Otherwise, a physcial machine is selected among other physical machines using MCDA.

One of the initial works [15], which emphases on vibrant VM relocation, was applied to unburden an overburdened physical machine. Though, the model intended for the optimization of the vibrant VM distribution has taken into account the cost of VM relocation, the authors did not applied any algorithm for determining when it was compulsory to perform the VM distribution optimization. The considered model was appealed periodically to regulate the VM distribution, which needs an additional performance any essential demand for optimization operation.

OpenStack, one of the eminent cloud platforms for both private and public clouds, was proclaimed in 2010 [16]. Among numerous existing sub-projects, OpenSack Nova [17] is the central project of OpenStack, which offers IaaS on demand. An example/VM can be tossed by means of OpenStack Nova on the effective compute node, which meets the client's desires. Though, OpenStack supports quick relocation technique, the manager has to physically mediate within the suitable time to transfer a VM occurrence from one compute node to another. This characteristic of OpenStack is valuable whenever a compute node, where many VM illustrations are running, requires to maintain or reallocate load. Though, the essential obligation for leading a vibrant live relocation, which warranties the SLA and QoS, is the VM relocation decision taken at an appropriate time. This deficiency of vibrant VM relocation leads to look for asuitable method to measure and monitor the load so as to migrate VMs to effective compute nodes. This method should be compliant with the resource usage requisite of on demand up/down scaling. Furthermore, new virtualization methods do not deliver enough performance segregation among the VMs. The disagreement for physical resources amongst VMs leads to dissimilar performance impact level among the VMs.

In order to regulate the time to invoke the relocation of VMs from a host, an experiment for setting a utilization threshold was primarily suggested in [18]. The main idea of their experiment was to adjust an operational threshold based on the CPU operation. In this study, the similar method is followed, but the host is discovered to be burdened if the mean of the former *n* CPU utilization quantifications is more than the pre-defined threshold value.

After shaping an overloaded host, the next step is to choose the VM(s) to transfer from one host to another. The three policies of VM selection particularly used in this study are provided as follows:

- Minimum Migration Time (MMT): This policy suggest that the VM selected to migrate needs least amount of time to complete relocation, related to other VMs assigned to the host.
- Random Choice (RC): This policy arbitrarily chooses VMs to transfer.
- Maximum Utilization (MU): This policy is based on certain VMs which are essential to be transferred in such a way that VMs with utmost CPU utilization, related to other VMs, are considered primarily.

The VM reallocation issue is handled by mapping VMs to the most well-organized hosts. In the preceding two years, there has been incredible awareness in this field and several algorithms have been suggested for VM placement. Most works emphasis on the CPU operation as the most significant resource and describe hosts in terms of their CPU ability and VMs in terms of their CPU load [18, 19, 20]. In [21], authors use the First Fit (FF) method which checks all the hosts and explores the appropriate host where the CPU exploitation is the minimum. Instead, a few reports make the problem multi-dimensional by also keeping in view a few other resource types such as memory and I/O. In this study, the latter method has been surveyed by taking in account the RAM in addition to CPU when determining effective compute nodes for VM placement.

More specifically, in this work, a methodology has been proposed and applied in a real environment for the renowned cloud computing software called OpenStack. In this methodology, every compute node sends RAM and CPU utilization numbers of each VM instance that is organized on this compute node. Based on these numbers, the algorithms running on the control node identify an overloaded VM occurrence, choose it, and place it on an effective compute node.

3. Resource Orchestration Framework and its Major Components

This section presents a novel service-oriented resource orchestration framework. The objective of the orchestration framework is to deliver main components in order to organize cloud and network resources necessary for cloud services to fulfill requests generated by distant end-users.

3.1 Proposed Architecture

The general architecture of suggested orchestration framework is demonstrated in **Fig. 1**. It comprises two essential building blocks: (i) The Orchestrator, (ii) Service Abstraction Model. We utilized Service Abstraction Model (SAM) for the orchestrator, with which the orchestrator comprehends the necessities of the requested services and delivers the resources for the services. The orchestrator is accountable for resource distribution based on the requests produced by distant end-users. The Cloud Resource Communicator (CRC) and Network

Resource Communicator (NRC) segments of orchestrator are intended to manage cloud and network resources respectively. Infra Monitor segment gathers network and cloud infrastructure-related data stored in an internal database server for usage by the Resource Allocator.

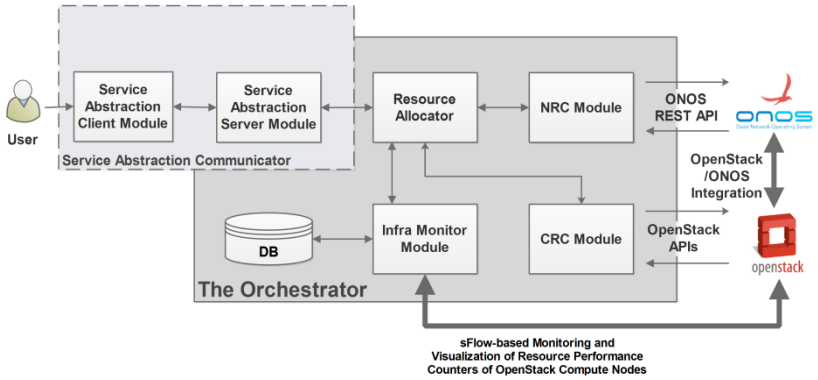

Fig. 1. Proposed orchestration framework

3.2 Service Abstraction Model

With our existing service concept model [22], the orchestrator obtains the abstracted necessities of the requested services, and maintains networks performance to meet the requested service requirements. The orchestrator deals with the vibrant nature of the services by simply varying the networks accordingly. Our service abstraction model abstracts the requested service parameters to legacy network services and support future. The general concept of the service concept model is portrayed in **Fig. 2**.

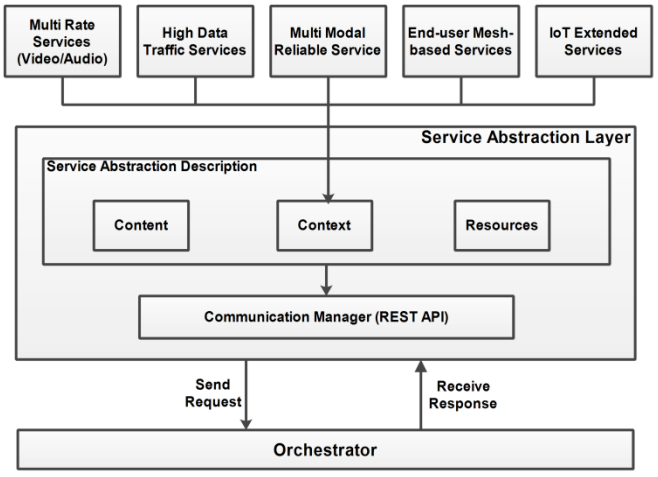

Fig. 2. The service abstraction model

The service requests are defined in the service concept depiction of service concept layer. The requirements are categorized into three classifications, i.e., Context, Content and Resources. The service-related parameters are delivered by Content. They may be resolution and standard of a audio bit-rate, video, and QoS. The user-related parameters are enclosed by

Context. They may be location and schedule of the service, or concern of the user. The requirements of the infrastructural resources are delivered by Resource. All these three sets of parameters are transformed into XML format, and are analyzed at service concept depiction module. After that, the communication manager module assign these requirements of a service to the orchestrator.

The communication between service applications and service abstraction layer is carried out in XML format. The request handler manager handles the incoming request and stores them in a database, where each request is assigned a status flag. Initially, the flag status is pending. All the requests having pending status are inspected in the database and are sent to the orchestrator to be processed via communication manager. The communication manager receives the response about the acceptance or rejection of the service request by the orchestrator, which is then changed in the database accordingly.

3.2.1 Service Abstraction Communicator

A JSON-RPC server-client segment is implemented to transport inattentive service requests to the orchestrator. The general structure of this module comprises a method that is defined in the abstracted service parameters and requested object. As described in **Fig. 3**, the service concept server modules and client interconnect with each other by means of a TCP channel. The inattentive service constraints are sent as a request from the service abstraction client unit to the service abstraction server module where they are analyzed and distributed to the orchestrator.

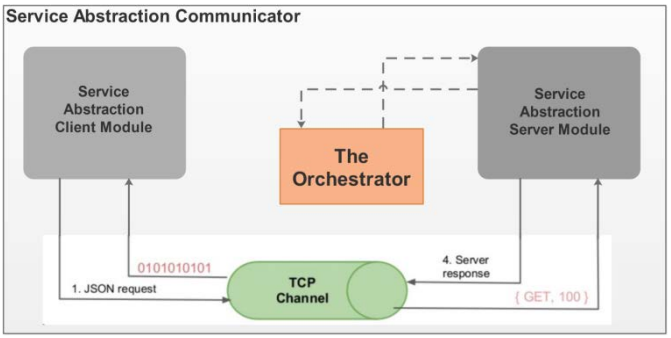

Fig. 3. Service Abstraction Communicator Module

3.3 The Orchestrator

It is the core building block of our suggested orchestration framework. Its essential module, Resource Allocator assigns resources in the form of VMs in order to meet the requirements of the requested applications, which are requested by distant end-users. Once a request is established, it is essential to define what VMs are the most suitable to be assigned. This problem is resolved by algorithms for VM selection. One such model can be choosing a VM arbitrarily from a group of VMs allocated to the host. Instead, based on our planned algorithm, Algorithm 1, named maximum RAM minimum CPU utilization (MRMC) procedure, VMs with the highest amount of RAM are primarily selected, and then out of these particular VMs, the VM with the lowest CPU consumption averaged over the last n quantifications is choosen. The algorithm takes n number of preceding CPU utilizations, RAM and CPU application numbers of VMs, and a pre-defined CPU utilization threshold. It begins with first arranging

the VMs in terms of their RAM utilization in an ascending order. Then, from the arranged list of VMs, the average value of the preceding n number of CPU utilizations of each VM is determined. If the intended mean value of CPU utilization of any specific VM in the arranged VM list is less than or equal to the pre-defined CPU operation threshold value, it is designated and reverted as an output of the algorithm for resource distribution.

Algorithm 1: Maximum RAM Minimum CPU utilization (MRMC) Algorithm

```
Input: n, vms_utilization_stats, cpu_threshold
Output: a VM to select
selected_vm ← None
minTempVal ← 0
sum ← 0
mean ← 0
for i = 0 to length(vm_list) do
    minTempVal ← i
    for j = i+1 to length(vm_list) do
        if (vm_ram_map[j] < vm_ram_map[minTempVal] then
            minTempVal ← j
        endif
    done (endfor)
    swap vm[i] with vm[minTempVal]
done (endfor)
for i = 0 to length(vm_list) do
    sum, mean ← 0
    for j = 0 to n do
        sum ← sum + vm_cpu_map[j]
    done (endfor)
    mean ← sum/n
    if mean ≤ cpu_threshold then
        selected_vm ← vm_list[i]
        break
    endif
done (endfor)
return selected_vm
```

Moreover, the Big-O time complexity for Algorithm 1 is given as:

$$T(n) = O(m^2 + mn)$$

where *m* is vm_list.

3.3.1 Network Resource Communicator

The NRC module is liable for handling the network infrastructure resources. As shown in **Fig. 4**, the NRC module interrelates with the SDN controller in order to get the details of existing resource, as well as to POST the rules/changes to manage the network. We use ONOS as the SDN controller. The NRC module uses the REST API of ONOS controller in order to obtain the information of devices, flows etc. It can GET the LINKS, INTENTS, FLOWS, and DEVICES. It can also POST intent as per the application requirements. The south bound API of ONOS offers an idea of the real physical network infrastructure.

There is a function/method defined in the NRC segment, which recovers the statistics linked to the network devices and links from the ONOS SDN controller and groups it in a variable. This data is joint with the orchestrator for managing and observing purpose. The NRC correspondent module also permits for installing host-to-host targets on ONOS SDN controller by means of HTTP POST method.

3.3.2 Cloud Resource Communicator

The CRC segment is intended for interrelating with the OpenStack and handling the compute/network resources inside OpenStack conferring to the user requirements. As illustrated in the **Fig. 5**, OpenStack delivers diverse APIs that can be utilized to allocate/control/manage resources on the physical infrastructure. The CRC module uses the uniqueness of API to verify with the OpenStack cloud in the very first step. Then it utilizes the

neutron API to create/modify/delete occupant networks inside OpenStack, and compute API to cope compute resources.

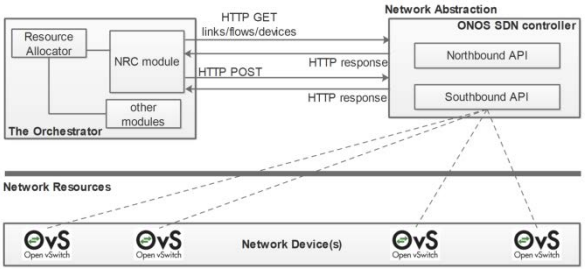

Fig. 4. Network resource communicator module

There is a function/method defined in the CRC segment, which recovers the data related to the previously propelled VM cases in OpenStack and collects it in a variable. Another function for making networks uses HTTP POST request to make occupant networks in OpenStack. It initially uses the unique information and makes a session with OpenStack and then makes a network with the name as stated in the parameters. There are also numerous other functions defined in CRC module that permit the orchestrator to remove the VM cases and networks in OpenStack clouds.

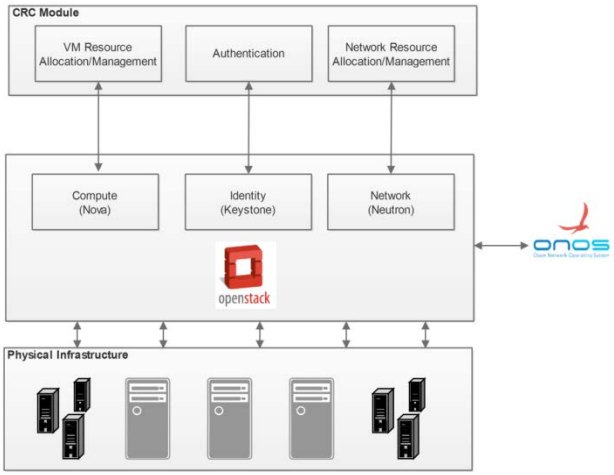

Fig. 5. Cloud resource communicator module

3.4 Implementation of Proposed of Proposed Resource Orchestration Framework

As depicted in **Fig. 6**, the CRC module of the Orchestrator handles compute/network resources in OpenStack rendering to the user requirements. A function/method has been defined in this segment which obtains numerous options as input variables and uses HTTP POST to establish a VM/instance in OpenStack. Similarly, the NRC module handles the network resources by interacting with ONOS SDN controller. The dafault Module Layer 2 (ML2) plug-in in OpenStack Neutron allows OpenStack networking to simultaneously utilize several layer 2 networking technologies existing in data centers. However, in the testbed under

consideration, the integration of ONOS SDN controller with OpenStack mechanism driver enables ONOS SDN controller to handle layer 2 networking instead of the default OVS mechanism driver. Similarly, ONOS layer 3 plug-in replaces OpenStack's default router plugin for layer 3 routing purpose.

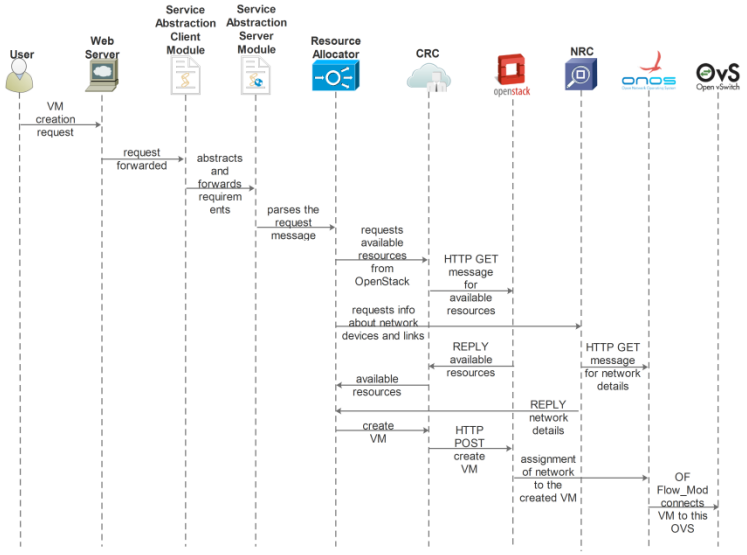

Fig. 6. Exchange of messages between different modules for VM creation

3.4.1 Performance Analysis of Proposed MRMC Algorithm

When a request is received, it is compulsory to assign the suitable VMs in terms of accessible resources. These VMs can be either choosen by means of efficeient methodology or arbitrarily. Nevertheless, arbitrary VM choice may lead to performance deprivation because the choosen VMs may have maximum CPU utiliation and least RAM. We address this issue by suggesting MRMC algorithm, which selects VMs on the basis of maximum RAM and minimum CPU utilization. It receives historical information on the resource usage by VMs running on the compute nodes and sends back a set of VMs to be assigned. The VMs experience dynamic workloads, which leads to the facts that the CPU and RAM usage by any VM arbitrarily changes over time.

The enactment of our proposed MRMC algorithm is matched with two algorithms, i.e., resorce allocation algorithm and the random VM selection scheme given in [23]. The assessment is first performed for VM choice on the basis of the amount of used RAM from the pool of VMs. All the algorithms are run ten times with the time difference of five seconds between each run. It can be evidently seen in **Fig. 7** that MRMC algorithm outclasses the arbitrary VM selection scheme. For each run, the MRMC algorithm selects VM(s) with least RAM utilization, whereas the arbitrary VM selection scheme selects VM(s) without taking into consideration the quantity of RAM accessible on each VM. The proposed MRMC also outclasses the MCDA algorithm as the MCDA algorithm makes choice on the basis of existing resources. It basically allocates a compute node which has the offered resources to meet the requirements of the requested service. On the other hand, our suggested MRMC algorithm has two-dimensional selection criteria. It first chooses those VMs from a pool of VMs which have the maximum amount un-utilized RAM for distribution.

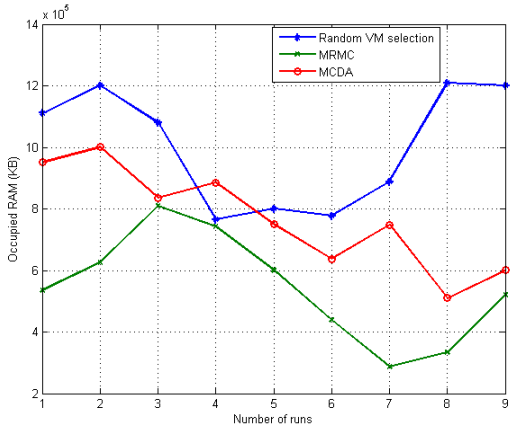

Fig. 7. Performance analysis of MRMC for minimum occupied RAM

In the next step, the proposed MRMC algorithm assigns the resource (VM) from a set of previously selected VM(s) having the maximum amount of un-utilized RAM by defining the VM with the least CPU utilization averaged over the last n quantifications. Instead, the arbitrary VM selection and MCDA algorithms assign resources irrespective of the CPU utilization for each run. The performance of the suggested MRMC algorithm is matched with arbitrary VM selection and MCDA algorithms on the basis of the least CPU utilization. All the algorithms are run ten times with the time difference of five seconds between each run. It is obvious from **Fig. 8** that MRMC algorithm outperforms the MCDA algorithms and random VM selection scheme and. The resource (VM) with the minimum CPU utilization is assigned by means of MRMC algorithm, whereas the arbitrary VM selection and MCDA algorithm do not consider CPU utilization when distributing resources.

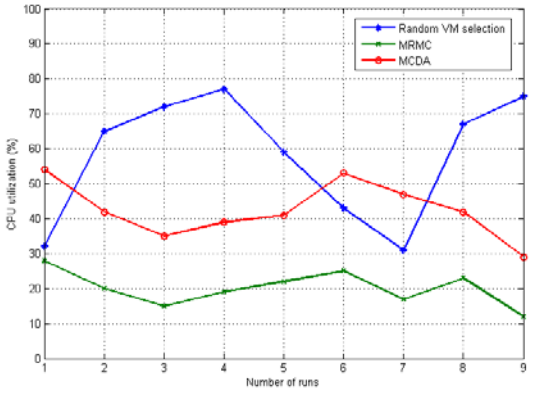

Fig. 8. Performance Analysis of MRMC for minimum CPU utilization

4. Resource Utilization-based Migration of VMs in OpenStack Clouds

In this section, we present an architecture and application of a framework for vibrant VM relocation in cloud data centers based on the OpenStack platform. The framework comprises a controller node and multiple instances of compute nodes. The purpose of the framework is to

deliver vibrant live relocation to regulate the VM instances on compute nodes to unload a number of VMs from an loaded compute node. It is pertinent to mention here that the role of VM migration framework comes into play after the orchestrator of previous section has assigned resources to the requested service in the form of the most optimum VM. This particular VM needs to be continuously monitored to avoid any overload situations. The VM migration framework works in parallel with the orchestrator and migrates VM(s) from an overloaded compute node to avoid performance degradation of the overall system. We discuss this issue in detail by concentrating on the subsequent key points:

- Noticing an overloaded compute node, so that some VM instances may be transferred to other well-organized compute nodes.
- Based on RAM and CPUutilization, selecting the overloaded VM instance(s) from a compute node.
- Detecting the selected VM instance(s) for relocation on other well-organized compute nodes.

4.1 System Model

The framework implements vital components for observing VMs and hypervisors, collecting resource utilization numbers, directing instructions and messages between the conducting VM live relocations and system elements. It allows the application of the three suggested algorithms for vibrant VM relocation: identifying overloaded compute node, choosing a VM based resource consumption, and assigning a VM to an effective compute node.

Fig. 9 represents the general architecture of the suggested framework that is organized in OpenStack to elude an overloaded compute node by leading vibrant VM relocation at a suitable time. It comprises four fundamental building blocks: (i) Stats Aggregator, (ii) Stats Database, (iii) Migration Manager, and (iv) Algorithms Repository. In the subsequent sub-section , we will discuss each of the framework's components in detail.

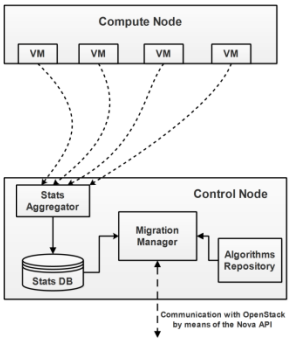

Fig. 9. Proposed system model

4.1.1 Stats Aggregator

A module that is positioned on the control node and is liable for gathering data on the resource utilization by VM instances and hypervisors and then advancing it to the statistics database, which can also be shared with other components. The data is collected by means of libvirt's API [24] in the form of the RAM consumed by hosts and VM instances. The CPU and RAM data are both collected occasionally and defer to to the Stats Database module of the framework.

4.1.2 Stats Database

The stats database is used for storing old data on the resource utilization by hypervisors and VM instances. The database is occupied by the stats aggregator organized on the same control node. The CPU and RAM utilization data of VM instances are occasionally defered to this module, which are then used by the relocation manager to govern the VM instances that are using majority of their own compute nodes' resources.

4.1.3 Migration Manager

The migration manager is responsible for leading VM relocations and making VM distribution choices, which results in ridding VMs from an loaded compute node. It runs the overload recognition algorithm when resource utilization numbers are obtained from the Stats DB module. If an overload state is identified, it runs the VM selection algorithm to choose the VM instances, which are utilizing maximum CPU and RAM resources. Then it regulates the effective compute nodes in order to place particular VMs on them, and entreats OpenStack API for live relocation of the choosen VM instances.

4.1.4 Algorithm Repository

This source is organized to store routine decision-making algorithms for vibrant VM migration, i.e., compute node overload recognition, VM choice, and VM distribution algorithms.

4.2 Proposed Structure and Algorithms

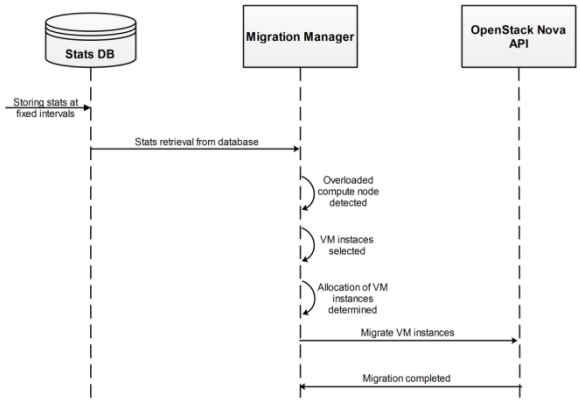

Fig. 10. Exchange of messages between components for VM instances migration

Fig. 10 represents the interchange of messages for managing a compute node overload situation. First, the relocation manager identifies an overload of the compute node using the overload recognition algorithm. Then, by means of suggested VM selection algorithm, the relocation manager selects VM instances based on their CPU utilizations. Next, the relocation manager begins the VM distribution algorithm with the list of selected VMs along with their utilized resources and states of the compute nodes attained from the stats database as arguments. Lastly, based on the VM distribution generated by the algorithm, the relocation manager requests the OpenStack Nova API for the suitable VM live relocation.

4.2.1 Overload Detection Algorithm

The overload recognition algorithm displayed in Algorithm 2 is a modest algorithm which takes *n* number of prior CPU utilizations, CPU and RAM utilization data of a compute node, and pre-defined RAM utilization and CPU utilization threshold values. It begins with modifying the well-defined variables. Then, from the list of compute nodes, the mean values of the last *n* number of CPU utilizations and the last *n* number of RAM utilizations for each compute node is determined. If the considered average value of RAM utilization or CPU utilization of any specific compute node is more than or equal to the pre-defined CPU utilization threshold value, it is choosen and sent back by the algorithm as an overloaded compute node.

Algorithm 2: Compute Node Overload Detection Algorithm

```
Input: n, cn_utilization_stats, cpu_threshold, ram_threshold
Output: overloaded_cn_list
sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
for i = 0 to length(cn_list) do
    sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
  for j = 0 to n do
      sum_cpu ← sum + cn_cpu_map[j]
      ram_cpu ← sum + cn_ram_map[j]
  done (endfor)
  mean_cpu ← sum_cpu/n
  mean_ram ← sum_ram/n
  if (mean_cpu ≥ cpu_threshold) || (mean_ram ≥ ram_threshold) then
      selected_cn_list ← cn_list[i]
      break
  endif
done (endfor)
return overloaded_cn
```

The Big-O time complexity for Algorithm 2 is given as:

$$T(n) = O(mn)$$

where *m* is cn_list.

4.2.2 VM Selection Algorithm

When an overloaded compute node is distinguished, it is essential to regulate what VMs are to be transferred. This matter is resolved by algorithms for VM selection. One such instance can be choosing a VM arbitrarily from a pool of VMs allocated to the host. Otherwise, based on our suggested algorithm called minimum RAM maximum CPU utilization (minRmaxC) algorithm as shown in Algorithm 3, VMs with the determined amount of RAM usage are first choosen, and then out of these choosen VMs, the VMs with the maximum CPU utilization be around the last *n* quantifications are choosen. The algorithm takes n number of preceding CPU utilizations, CPU and RAM utilization numbers of VMs, and a pre-defined CPU utilization threshold. It begins with initially categorizing the VMs in terms of their RAM utilization in a descending order. Then, from the organized list of VMs, the average value of the last n number of CPU utilizations of each VM is determined. If the considered mean value of CPU utilization of any specific VM in the sorted VM list is more than or equal to the pre-defined CPU utilization threshold value, it is choosen and sent back as an output of the algorithm for VM relocation.

A VM selected for relocation must have maximum RAM and CPU utilizations between all other VMs located on any specific compute node. More specifically, a VM with extreme utilization is the most well-organized case. To achieve this objective, we govern the utilization of a VM by (1). Where $c(i)$ is the VM CPU utilization and $m(i)$ is its RAM utilization. Hence,

in case there are *n* number of VMs for relocation, the one which has the uppermost utilization is choosen.

$$U(i) = \sum_{i=0}^{n} c(i) * \sum_{i=o}^{n} m(i) \quad (1)$$

Meanwhile the cost of VM live relocation is typically determined by its memory footprint. Thus, it can be said, relocation time of a VM is directly comparative to the memory size of that VM. As a result, memory size of VMs is a good measure for the cost of relocation. Therefore, in case there are options for relocation, the VM which has the minium memory footprint is appropriate for relocation.

The relocation cost of a VM can be determined by the (2)

$$Migration\ cost(i) = U(i) + s(i) \quad (2)$$

Where $U(i)$ is the VM with uppermost utilization given by (1), and $s(i)$ is the allocated memory size to an *ith* VM.

Algorithm 3: Minimum RAM Maximum CPU utilization (minRmaxC) Algorithm

```
Input: n, vms_utilization_stats, cpu_threshold
Output: a VM to migrate
selected_vm ← None
maxTempVal ← 0
sum ← 0
mean ← 0
for i = 0 to length(vm_list) do
    maxTempVal ← i
    for j = i+1 to length(vm_list) do
        if (vm_ram_map[j] > vm_ram_map[maxTempVal] then
            maxTempVal ← j
        endif
    done (endfor)
    swap vm[i] with vm[maxTempVal]
done (endfor)
for i = 0 to length(vm_list) do
    sum, mean ← 0
    for j = 0 to n do
        sum ← sum + vm_cpu_map[j]
    done (endfor)
    mean ← sum/n
    if mean ≥ cpu_threshold then
        selected_vm ← vm_list[i]
        break
    endif
done (endfor)
return selected_vm
```

More precisely, based on the VM selection criteria for VM migration, the VMs with the maximum CPU utilization averaged over the last *n* quantifications are first selected.

$$CUM_i = \frac{1}{n}\sum_{k=0}^{n} CU_i(t-k)$$

$M(i,j)$ is a 2D array that contains VMs based on the following condition:

$$VM_i = \begin{cases} 1, & CUM_i \geq x \\ 0, & CUM_i < x \end{cases}$$

Where x is a threshold for CPU utilization. VM_i is an indicator that shows if *ith* VM is in $M(i,j)$ matrix or not. The VM for migration is selected based on minimum RAM, which is expressed by following equation:

$$VM_b = \min\left(M(RAM_{util})\right) \quad (3)$$

Moreover, the Big-O time complexity for Algorithm 3 is given as:

$$T(n) = O(m^2 + mn)$$

where *m* is vm_list.

4.2.3 VM Allocation Algorithm

In order to recieve the effective compute nodes for hosting VM instances, the VM distribution algorithm involves an OpenStack Nova-scheduler which conducts a general distribution process. More specifically, it sends back the effective compute nodes on which the VM instances would be placed. Thus, the algorithm first chooses a list of the light load compute nodes for the heavy load VM instances. Formerly, this list is transported to the OpenStack-Nova API in order to begin the VM relocation process between source and destination compute nodes. The pseudo-code for algorithm is presented in Algorithm 4, which clarifies the technique of choosing the light load compute node for the hefty load VMs. The algorithm receipts n number of preceding CPU utilizations, CPU and RAM utilization data of a compute node, as well as CPU and RAM utilization data of VMs. It starts with modifying the distinct variables. Formerly, from the list of compute nodes, the mean values of the last n number of CPU utilizations and the last n number of RAM utilizations for each compute node is determined. If the intended mean value of compute node's CPU utilization is more than the CPU utilization of any specific VM and the calculated average value of compute node's RAM utilization is more than the RAM utilization of that specific VM, it leads to the fact that compute node has sufficient resources to manage the transferred VM. Henceforth, the algorithm chooses and returns this compute node for VM instance settlement.

Algorithm 4: VM Instance Allocation Algorithm

```
Input: n, cn_stats, vm_stats
Output: Available_cn_list
sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
for i = 0 to length(cn_list) do
    sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
  for j = 0 to n do
      sum_cpu ← sum + cn_cpu_map[j]
      ram_cpu ← sum + cn_ram_map[j]
  done (endfor)
  mean_cpu ← sum_cpu/n
  mean_ram ← sum_ram/n
  if (mean_cpu > vm_cpu) && (mean_ram > vm_ram) then
      selected_cn_list ← cn_list[i]
      break
  endif
done (endfor)
return available_cn_list
```

The Big-O time complexity for Algorithm 4 is given as:

$$T(n) = O(mn)$$

where *m* is cn_list.

4.3 Performance Analysis of Proposed Algorithms

The suggested architecture has been authenticated in the cloud computing platform-based testbed conferred in Section 3.4. Initially it is essential to produce the work load in a suitable way in order to replicate an accurate data. For this purpose, software called Lookbusy [25] is used, which is a simple application for load generation on a Linux system. It allows to produce expected, fixed loads on CPUs, and also uphold choosen amounts of memory active.

Numerous experimentations are done to assess the proposed algorithms. For this purpose, the suggested overload detection algorithm has been examined for two cases, i.e., the

influence on an overall system without and with the implementation of this algorithm. The proposed VM choice and VM distribution algorithms have been matched with arbitrary VM assortment and arbitrary VM distribution schemes. A VM instance type with 32 MB amount of RAM assigned to it, has been launched on each compute node. Load has been produced on arbitrary VM instances of compute node 1, which results in an augmented CPU utilization of the VM instances.

During the experimentation, the overload recognition algorithm has been implemented by the relocation manager module, which perceives an overloaded compute node based on pre-defined threshold value. The performance of our suggested overload detection algorithm has been matched for two situations: 1) overload scenario without any algorithm used, and 2) direct overload detection [26]. The suggested algorithm distinguishes an overloaded compute node if the average of the last n CPU utilization measurements is more than the pre-defined threshold value, while the direct overload recognition identifies an overloaded compute node as soon as the CPU utilization exceeds the pre-define threshold value. The disadvantage in direct overload detection is that mostly a compute node goes beyond the threshold for a very short period of time because there are always CPU utilization spikes in real time while performing tasks. In this case, a compute node would be distinguished as an overload node as soon as its CPU utilization is beyond the threshold for a very small period of time. The granularity can be choosen conferring to the requirement; though, in most of the cases the system is not too delicate to the CPU overloads. In the case when no overload recognition algorithm is implemented, the compute nodes stay overloaded, which might result in the deprivation of an overall system/service. **Fig. 11** represents the recognition of compute node for each run of the suggested overload recognition and direct overload recognition algorithms. It is clear from the figure that the compute node is perceived as an overloaded compute node by direct overload detection algorithm as soon as it exceeds the threshold value of 80% CPU utilization. On the other hand, the suggested overload recognition algorithm identifies an overloaded compute node if the average of the last n CPU utilization measurements is more than the pre-defined threshold value. In the case when the algorithm is not implemented, there is no overload recognition even if the CPU utilization exceeds the threshold value of 80%.

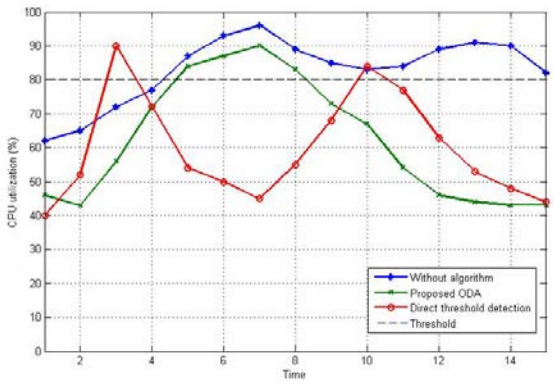

Fig. 11. Performance analysis of the proposed overload detection algorithm

When an overloaded compute node has been distinguished, the following step is to choose the VM instances which are consuming most of the resource of that compute node. These VM instances can be choosen arbitrarily, but this might lead to performance deprivation because the choosen VM instances may the least CPU utilization at the time of choice. Another problem with VM choice is that if the choosen VM has the higher RAM assignment, the

relocation time mandatory for its relocation would be high, which might cause an interruption in the process of the applications running on that VM. We address the above-mentioned problems by suggesting minRmaxC algorithm, which chooses VM instances on the basis of least RAM and maximum CPU utilization. It receives historical information on the resource usage by VM instances running on the compute nodes and returns a set of VM instances to be choosen. The performance of the suggested minRmaxC algorithm is compared with arbitrary VM selection and maximum utilization [20] algorithms on the basis of the maximum CPU utilization.

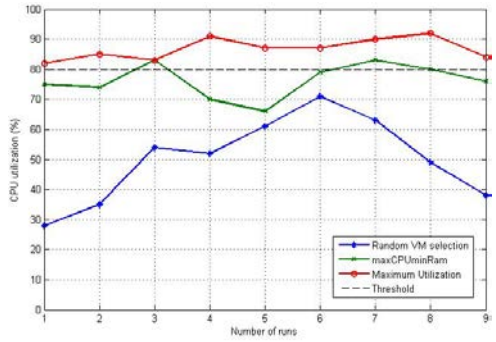

Fig. 12. Performance analysis of the proposed minRmaxC-based VM selection algorithm

It is certain from **Fig. 12** that minRmaxC algorithm outclasses the arbitrary VM selection scheme. The VM instances with the maximum CPU utilization are choosen by means of minRmaxC algorithm, whereas the arbitrary VM selection scheme chooses VM instances irrespective of the CPU utilization for each run. Though, the maximum utilization algorithm chooses VMs which have higher CPU utilizations than the VMs choosen by suggested minRmaxC algorithm, the previous does not consider the RAM factor when selecting VMs, which may have influence on the minimum relocation time required for transferring the choosen VMs [20].

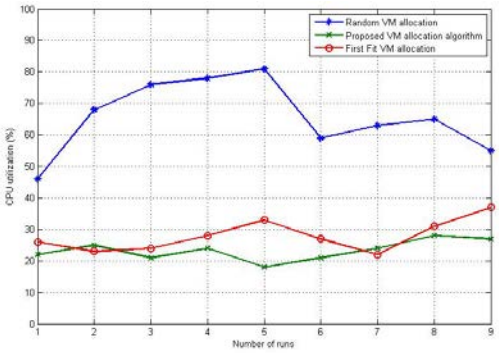

Fig. 13. Performance analysis of the proposed VM allocation algorithm

After choosing the overloaded VM instances, the relocation manager segment runs the VM distribution algorithm to place the choosen VM instances on an effective compute node. The performance of the suggested algorithm is matched with arbitrary VM distribution scheme and first fit VM distribution algorithm [21]. The first fit algorithm starts with the first compute

node and governs the obtainability of the required CPU resources. If it finds adequate resources, it places the transferred VM on that compute node, otherwise goes to the next compute node. Instead, our suggested algorithm finds the most effective compute node amongst all the nodes. For this purpose, it determines the mean RAM utilization and mean CPU utilization to conclude the most effective compute node from a list of compute nodes. If the considered mean values fulfill the requirements of the transferred VM, it selects that compute node for VM placement. In case of random VM allocation scheme, the VMs are placed on arbitrary compute nodes.

Fig. 13 represents the contrast of our suggested VM distribution algorithm with the arbitrary VM distribution algorithm and first fit VM distribution algorithm. It can be easily seen that the suggested VM distribution algorithm outclass the first fit VM distribution algorithm by choosing light load compute nodes in terms of CPU utilization. It also easily outclasses the arbitrary VM distribution scheme because it may place VMs on previously overloaded compute nodes, which may lead to the performance deprivation of an overall system.

5. Conclusions

In this paper, a resource orchestration framework for resource allocation has been presented. The resource provisioning is performed by implementing an algorithm. Two novel modules have been proposed and validated for communication with ONOS SDN controller and OpenStack in order to orchestrate network and cloud resources. The performance of proposed MRMC algorithm is compared with two algorithms for the VM(s) selected on the basis of minimum CPU utilization as well as the amount of utilized RAM. The experiment results show that both the VM selection algorithms are outperformed by the proposed MRMC algorithm.

In addtion, a framework for dynamic VM migration in OpenStack clouds has been proposed in this paper. The proposed overload detection, VM selection, and VM allocation algorithm address the issue of dynamic VM migration. The experiment results show that the proposed algorithms outperform the algorithms that are considered for the sake of comparison. Avoiding single point of failure and scalabiity are important advantages of the proposed system in a distributed manner. Based on the distribured approach adopted, overload detection and VM selection algorithms can easily scale with the increased number of compute nodes because these algorithms are run independently on each compute node.

Despite substantial contributions of the paper in cloud resource management and provisioning, and VM migrations, there are a number of open challenges that require to be addressed for further advancing these areas. The overload detection algorithm proposed in this paper detects the overloaded compute node on the basis of CPU utilization only. However, there are factors like RAM, and bandwidth that may affect the compute node. In order to make the algorithms more efficient, these additional factors need to be considered.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2016R1D1A1B01016322).
This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the

ITRC(Information Technology Research Center) support program(IITP-2017-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion)

References

[1] B Jennings, R Stadler. "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, Jul 1;23(3):567-619, 2015. Article (CrossRef Link)
[2] Menezes, Evandro, et al. "CPU utilization measurement techniques for use in power management," *U.S. Patent* No. 6,845,456. 18 Jan. 2005.
[3] Gade, Anuradha, Bruce McMurdo, and Jeremy Stieglitz. "System and method for improving network resource utilization," *U.S. Patent Application* No. 11/154,204.
[4] M. Bichler, T. Setzer, and B. Speitkamp, "Capacity planning for virtualized servers," in *Proc. of Workshop on Information Technologies and Systems (WITS)*, Milwaukee, Wisconsin, USA, volume 1, 2006.
[5] G. Khanna, K. Beaty, G. Kar, and A. Kochut., "Application performance management in virtualized server environments," in *Proc. of Network Operations and Management Symposium*, 2006. NOMS 2006. 10th IEEE/IFIP, pages 373–381. IEEE, 2006. Article (CrossRef Link)
[6] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu., "A cost-sensitive adaptation engine for server consolidation of multitier applications," *Middleware 2009*, pages 163–183, 2009.
[7] Afaq Muhammad; Zubair Amjad, Wang-Cheol Song, "A Framework for Orchestration based on Live Migration of Virtual Machines," in *Proc. of KSII Conference* 2016 (Seoul, Korea), 17(2), 121-122, 2016,
[8] Liu, C., Mao, Y., Van der Merwe, J., and Fernandez, M. "Cloud Resource Orchestration: A Data-Centric Approach," in *Proc. of CIDR*, 2011.
[9] Van der Merwe, J., Ramakrishnan, K., Fairchild, M., Flavel, A., Houle, J., Lagar-Cavilla, H. A., and Mulligan, J., "Towards a ubiquitous cloud computing infrastructure," in *Proc. of LANMAN*, 2010. Article (CrossRef Link)
[10] Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. "Black-box and gray-box strategies for virtual machine migration," in *Proc. of NSDI*, 2007.
[11] Agarwal, S., Dunagan, J., Jain, N., Saroiu, S., Wolman, A., and Bhogan, H. "Volley: automated data placement for geo-distributed cloud services," in *Proc. of NSDI*, 2010.
[12] Wood, T., Gerber, A., Ramakrishnan, K., Shenoy, P., and der Merwe, J. V. "The case for enterprise-ready virtual private clouds," in *Proc. of HotCloud*, 2009.
[13] Peng, Junjie, et al. "Comparison of several cloud computing platforms," in *Proc. of Information Science and Engineering (ISISE), 2009 Second International Symposium on. IEEE*, 2009. Article (CrossRef Link)
[14] Wang, Lizhe, et al., "Cloud computing: methodology, systems, and applications," *CRC Press*, 2011.
[15] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 243–264, 2008.
[16] O. Sefraoui, M. Aissaoui, M. Eleuldj., "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications.* Jan 1;55(3), 2012.
[17] OpenStack Nova, 2011. Article (CrossRef Link).
[18] JungYul Choi, "Virtual Machine Placement Algorithm for Saving Energy and Avoiding Heat Islands in High-Density Cloud Computing Environment," *J. KICS*, vol. 41, no. 10, pp. 1233-1235, Oct. 2016.
[19] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. of 10th IFIP/IEEE Int. Symp. Integrated Netw. Management (IM)*, pp. 119-128, Munich, Germany, 2007. Article (CrossRef Link)
[20] M. R. Chowdhury, M. R. Mahmud, and M. R. Rashedur, "Study and performance analysis of various VM placement strategies," *IEEE/ACIS SNPD*, pp. 1-6, Jun. 2015.

[21] M. R. Chowdhury, M. R. Mahmud, and M. R. Rashedur, "Implementation and performance analysis of various VM placement strategies in CloudSim," *J. Cloud Computing*, vol. 4, no. 20, Dec. 2015.
[22] DN Gde, Q Nguyen-Van, TV Duc, N Nguyen-Sinh, PJ Alvin, K Kim, D Choi. "Design of service abstraction model for enhancing network provision in future network," in *Proc. of Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*, IEEE, Oct 5 (pp. 1-4), 2016.
[23] Yazir, Yagiz Onat, et al., "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis," in *Proc. of Cloud Computing (CLOUD)*, 2010 *IEEE 3rd International Conference on. IEEE*, 2010.
[24] "Libvirt, the Virtualization API," 2017. Article (CrossRef Link)
[25] "lookbusy -- a synthetic load generator,"
[26] Abdelsamea, Amany, et al. "Virtual machine consolidation challenges: a review," *International Journal of Innovation and Applied Studies* 8.4, 1504, 2014.
[27] P Berde, M Gerola, J Hart, Y Higuchi, M Kobayashi, T Koide, B Lantz, B O'Connor, P Radoslavov, W Snow, G Parulkar. "ONOS: towards an open, distributed SDN OS," in *Proc. of Proceedings of the third workshop on Hot topics in software defined networking*, Aug 22, pp. 1-6, ACM, 2014.
[28] S. T. Selvi, C. Valliyammai, and V. N. Dhatchayani., "Resource Allocation Issues and Challenges in Cloud Computing," in *Proc. of 2014 International Conference on Recent Trends in Information Technology*, pages 1–6, Chennai, India, April, 2014.
[29] P. Salot. "A Survey of Various Scheduling Algorithm in Cloud Computing Environment," *International Journal of Research in Engineering and Technology*, 2(2):131– 135, February 2013.
[30] Bangui, Hind, et al. "Multi-Criteria Decision Analysis Methods in the Mobile Cloud Offloading Paradigm," *Journal of Sensor and Actuator Networks*, vol. 6, no. 4, pp. 25, 2017. Article (CrossRef Link)

Afaq Muhammad

He received PhD degree in Computer Eng. from Jeju National University, MS degree in Electrical Eng. with emphasis on Telecom from Blekinge Institute of Technology, Sweden, and BS degree in Electrical Eng. from University of Eng. and Technology, Peshawar, Pakistan in 2017, 2010, and 2007 respectively. Currently, he is working as an Assistant Professor in the Department of Computer Science and IT at Sarhad University of Science and IT, Pakistan. He also worked as a Postdoctoral Researcher at Network Convergence Lab, Jeju National University. Before starting his PhD, he worked as a Research Associate in the Faculty of Comp. Sci. and Eng. at GIK institute of Eng. Sciences and Technology, Pakistan, and as a Lecturer in the Department of Electrical Engineering at City University of Science and IT. His research interests are cloud computing, software defined networking, network function virtualization, wireless networks, and protocols.

Wang-Cheol Song

He received B.S. degree in Food Engineering and Electronics from Yonsei University, Seoul, Korea in 1986 and 1989, respectively. And M.S. and PhD in Electronics studies from Yonsei University, Seoul, Korea, in 1991 and 1995, respectively. Since 1996 he has been working at Jeju National University. His research interests include VANETs and MANETs, Software Defined Networks, network security, and network management.