IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 8, 2019, accepted September 26, 2019, date of publication October 3, 2019, date of current version October 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2945499

Characterizing Dynamic Load Balancing in Cloud Environments Using Virtual Machine Deployment Models

MISBAH LIAQAT[1,2], ANJUM NAVEED[2], RANA LIAQAT ALI[3], JUNAID SHUJA[4], AND KWANG-MAN KO[5]

[1]Department of Computer Science and Engineering, Air University Islamabad, Islamabad 44000, Pakistan
[2]Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[3]Department of Physics, COMSATS University Islamabad, Islamabad 44000, Pakistan
[4]Department of Computer Science, COMSATS University Abbottabad, Abbottabad 22060, Pakistan
[5]Department of Computer Engineering, Sangji University, Wonju 26339, South Korea

Corresponding authors: Misbah Liaqat (misbah.liaqat@mail.au.edu.pk) and Kwang-Man Ko (kkman@sangji.ac.kr)

This work was supported by the Basic Sciences Research Program (2017030223) Funded by the Ministry of Education and international cooperation program (2018090561, FY2018) managed by the National Research Foundation of Korea (NRF). We also extend our sincere appreciation to the Higher Education Commission (HEC) at COMSATS University Islamabad Pakistan for funding the NRPU/2017/R&D (Project No 10393).

ABSTRACT The ever growing computational demands of users call for efficient cloud resource management to avoid service-level agreement (SLA) violation. Virtualization co-locates multiple virtual machines (VMs) on a single physical server to share the underlying resources for efficient resource management. However, the decision about ''what'' and ''where'' to place workloads significantly impacts performance of hosted workloads. Existing cloud schedulers consider a single resource (RAM) to co-locate workloads that as a result lead to SLA violation due to non-optimal VM placement. To handle this issue, current study has updated nova scheduler to propose a multi-resource based VM placement approach to improve application performance in terms of central processing unit (CPU) utilization and execution time. Experimentally we have shown that our proposed method has lessened application execution time by 50% when compared with one of the well-known technique.

INDEX TERMS Date center, initial VM deployment, OpenStack, scheduling, virtualization.

I. INTRODUCTION

During the last one decade, due to extensively increasing demand for high-end computational servers, efficient cloud resource management has become a must to meet requirements for cloud providers [1], [2]. Virtualization configures and runs numerous workloads on a single physical server to attain high resource utilization for effective cloud resource management [3], [4]. However, aggressive workload co-location leads to resource over utilization that significantly impacts application performance in terms of SLA violation. Therefore, the decision about what and where to place workloads is very important as efficient workload distribution surges in application performance due to diminishing hotspots with data centers (DC). Alternatively, cloud resource underutilization significantly impacts return on investment (ROI) for cloud operators. Load balancing within cloud data centers fairly distributes a workload onto a set of physical servers to, (i) increase ROI, (ii) minimize the number of hotspots, (iii) reduce SLA violation, and (iv) minimize cloud operational cost.

Load balancing guarantees that all physical resources within cloud DC have a uniform workload. Existing load blanching schemes such as, Round Robin [5], Min-Min scheduling [6]–[8], Max-Min Algorithm [9], OpenStack Scheduler [10], Min-min Algorithm [11], and Improved Max-min [12], have considered static load balancing (single-resource) to co-locate VMs. All aforementioned schemes opted VM placement scheduler that overlooks queuing user requests, and a fair-share algorithm enabled resources provisioning within data centers. Furthermore, in OpenStack the design of existing nova scheduler is not optimized as it considers the RAM availability factors only to select servers for VM deployment. However, in static load balancing the absence of dynamic state of CPU load, during the scheduler's decision making leads to inappropriate workload distribution

The associate editor coordinating the review of this manuscript and approving it for publication was Christos Verikoukis.

on physical hosts. The inappropriate workload distribution surges application execution time. Therefore, to handle these issues, we have proposed a method that considers multiple resources during workload's co-location to improve the application execution time.

The main contribution to this study includes:

- Performance evaluation of existing OpenStack scheduler to present a case for dynamic load balancing through VM placement.
- Proposing and implementing a multi resource-based scheduler to minimize the deficiencies of OpenStack scheduler.
- Proposing and implementing a load analyzer algorithm to capture the current CPU utilization. Load analyzer effectively capture the rising trend of CPU utilization pattern to manage the VM placement based on current load.

The detailed structure of the paper is as follows. Section II briefly discusses the related work and resource allocation background. Section III presents, the proposed solution and core algorithms. Section IV elaborates the initial VM deployment model. Moreover, the experimental setup is discussed in section V. In addition, performance analysis of existing and proposed approaches is conducted in section VI. Finally, a comprehensive conclusion of this work is included in section VII along with future research directions.

II. RELATED WORK

During recent era, load balancing has become an active area of research due to multifarious applications in various computing domains such as, cloud computing [13], big data [14], [15] and grid computing [16]. For the sake of understanding, we have organized existing state-of-the-art in two categories. First category briefly debates on initial VM placement schemes which are closely related to our work. Alternatively, second category focuses on the VM placement based cloud schedulers.

Regarding the initial placement of VMs, researchers in [17] considered VM placement at the time of creation using discrete event-based Koala simulator. However, [17] lacks in considering the requirements of application during VM placement. The authors of [18] exploited black-box method to identify the capability of multi-tier application hosted in virtualized platforms. Similarly, authors of [19] employed same techniques as discussed in [18] to investigate different factors such as, deployment scenarios, real-time scheduler's decisions, and types of workload, to identify the parameters influencing VMs co-location. On the other hand, researchers in [20] considered the relationship among three categories of workload based on several virtual network configuration strategies in terms of the number of VMs, vCPUs usage per VM, and memory size for each VM. However, this work overlooked the impact of statistical methodology to the concluded results.

A Joint-VM provisioning approach can be seen in [21] which consolidate and provide provision of multiple VMs based on the estimate for their aggregated capacity needs. The statistical multiplexing technique is utilized among the work load patterns of multiple VMs. Alternatively, backward speculative placement (BSP) technique considered the VM's resource usage history curve to decide VMs relocation and placement. Though, BSP only considered the number of CPUs while it overlooks the effects of load on physical hosts. In addition, OpenStack components work together according to the cloud needs and communicate with each other using the RPC and RabbitMQ protocol [22]. In DC, OpenStack compute service (nova), assists administrators to deploy one or multiple hypervisors to virtualize the underlying resources. It handles the communication with local hypervisor to ensure VM initiation and termination along with the performance metrics and queries to VM load indicators [23].

In second category, various aspects and evaluation of cloud schedulers are extensively presented. Some of the aspects dealing with the data processing flow which can be seen in [24], and [25]. In a survey [26], the author presented the meta-scheduler analysis. Furthermore, a ''model for instantiating dynamically virtual machine in relation to the current job characteristic'' is described in [27] by utilizing CloudSim. In Cloud context, the scheduler remains of interest without considerations of virtualization and its overheads [28] and [29]. The Openstack scheduler performance can be enhanced as per developer's view point discussed in [30] and [31]. Detailed comparisons of Openstack features with the other open source solutions such as OpenNebola and VMware are presented in [32]–[34], and [35] in detail. Some of the performance issues are identified by the authors of [36] and [37] in comparison with the previous version of Openstack, named as Cactus. A VM consolidation strategy that is implemented on OpenStack cloud by considering SLA violation is presented in [23] and [38]. Further, a dynamic VM consolidation strategy is proposed in [39] that chose the least active physical server from where the VM is migrated in order to reduce the utilization. Moreover, authors in [10] explain that in OpenStack, the number of CPU used and the amount of RAM play a vital role in initial assignment of VMs in different types of scheduler (filter, chance and simple).

The Openstack cloud is classified based on three types of schedulers such as Chance, Filter, and Simple [33]. In Chance scheduler, the available nodes are randomly chosen for placement regardless of its characteristics whereas the Simple scheduler identifies the available node with the least load to deploy first VM. In contrast, filter scheduler maps the nova-api calls to appropriate component. The filter scheduler consists of two functions the filtering and weighting to compute genetically as shown in Fig. 1. A set of compute servers having the capability to run a given VM is selected by the filter function [10], [40]. On the contrary, a cost function ranks the filtered set of servers with respect to their suitability. At the deployment of VM, filter scheduler creates a disk image for VM and calls the hypervisor for VM boot [41], [42]. This call includes the parameters described as numbers of cores, the amount of RAM needed and type of vCPU along

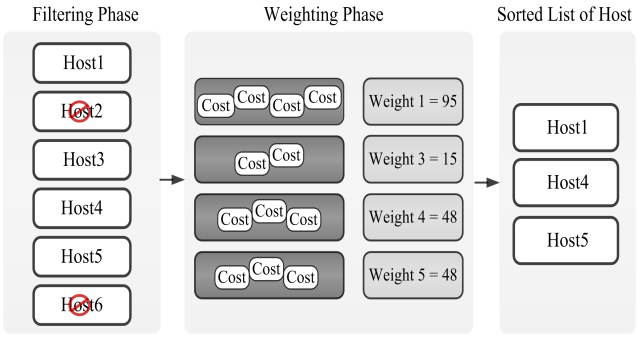


FIGURE 1. Filter scheduler.

with local CPU allocation policy and the hard disk image to boot from.

Based on literature review it is perceived that cloud schedulers do not consider the CPU utilization at the deployment of VM. Our proposed work is different from all aforementioned VM placement schemes as we have proposed multi-resource objective based scheme. We empowered Nova scheduler to consider the RAM capacity and number of vCPUs in addition with CPU utilization (CPU load) for initial placement to minimize VM migration.

III. PROPOSED SOLUTION

Fig. 2 presents an overview of our proposed VM deployment architecture in OpenStack cloud. In the said figure, OpenStack cloud controller represents the physical host that runs the API components, schedulers, and compute servers. Each compute server is deployed with the OpenStack compute competent. The functionality of a request handler is apprehended in OpenStack scheduler and API components. The request handler accepts the VM placement requests, as long as CPU demand remains below the cloud capacity (i.e. Utilization of CPU does not exceed the defined threshold. Otherwise, the request is rejected. The placement architecture is split in two engines such as, Global Decision Engine (GDE), and Local Monitoring Engine (LME). The Load Filter is running in the controller which resides within GDE. The GDE performs the initial placement of VM based on decision making parameters collected through the LME. LME shows the Compute Load and Load Analyzer components. Compute Load performs computations based on CPU states. In contrast, Load Analyzer collect the aggregated results of CPU utilization from each compute server and transfer it to LME.

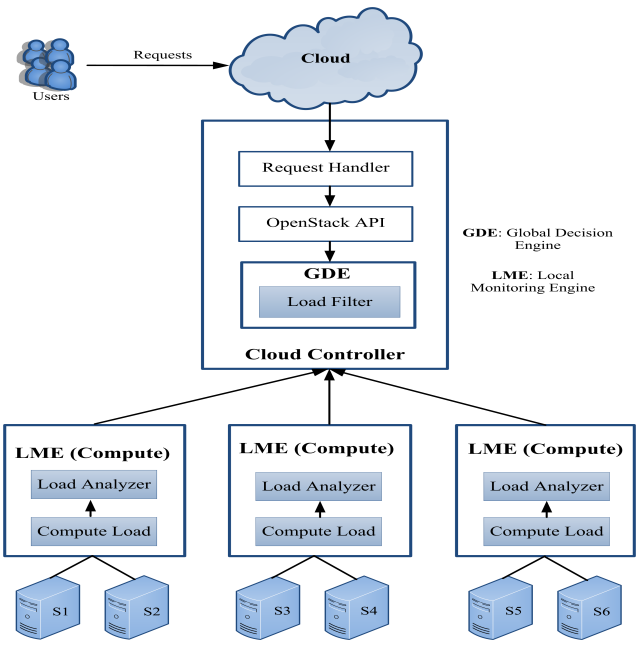


FIGURE 2. VM deployment architecture.

In this study, some details about the proposed algorithms which are used in our implementation are explained. Moreover, Table 1 represents the symbols and their description.

TABLE 1. Algorithm's symbols and their description.

Symbol(s)	*Description*
ul	User Load
sl	System Load
nl	Nice state
ncl	Idle state
SP	System's previous load
SR	System's resent load
AvgLoad	Average Load
N	Total number of physical hosts
n	specific physical hist
s	Host state
p	Filter properties
TU_i	requested instance type from users
i	i is instance type
car	CPU allocation ratio
d	Database
TvCPUs	Total number of vCPUs

A. COMPUTE LOAD

Compute Load (CL) calculates the average load and updates it in a local database. CL exploits current and previous CPU utilization states to compute average load on CPU. Initial phase comprising (line 3-6) calculates CPU utilizations based on user state, system state, nice state, and idle state. Moreover, subsequent stage (line 7-11) refers to average load finding process. Average load is computed based on the ratio of CPU used to total CPU capacity as presented in Algorithm 1.

B. LOAD ANALYZER

Load Analyzer (LA) computes the load using the Algorithm 1 for all physical servers and shares it with GDE as shown in Fig. 2. LA (Algorithm 2), transfer the CPU load by establishing a one-to-one communication link between GDE and Load Analyzer for every compute node (line 2-3).

Algorithm 1 Compute Load

1: tsecond ← 30
2: **while** (1) **do**
3: ul ← getUserProcessLoad()
4: sl ← getSystemProcessLoad()
5: nl ← getnice()
6: ncl ← getNotUsedCPUpercentage()
7: SR_1 ← {ul, sl, nl }
8: SR_2 ← {ul, sl, nl, ncl}
9: SP_1 ← {ul, sl, nl }
10: SP_2 ← {ul, sl, nl, ncl}
11:

$$AvgLoad \leftarrow \frac{\sum_{I \epsilon SR_1}^{length(SR_1)} Cost\ (i) - \sum_{j \epsilon SP_1}^{length(SP_1)} Cost\ (j)}{\sum_{I \epsilon SR_2}^{length(SR_2)} Cost\ (i) - \sum_{I \epsilon SP_2}^{length(SP_2)} Cost\ (j)}$$

12: Wait(tsecond)
13: Load-DB-update(AvgLoad)
14: Goto step 2
15: **end while**

Algorithm 2 Load Analyzer

1: **for** each node $n \in N$ **do**
2: Loadinfo$< n$, val $>$ ← ComputeLoad(n)
3: send-LoadInfo-GDE(ComputeLoad(n))
4: **end for**

C. LOAD FILTER

Load Filter (LF) decides whether a particular compute server is feasible to host a VM or not. Load Filter is based on four steps. LF starts with a verification process to see that the user requested instance fulfills the deployment criteria or not (see line 2-5). In the second step, LF investigates available resources based on vCPU's current utilization level (see line 6-11). During step-3, LF gathers the load information across each physical server from GDE as transmitted by Algorithm 2 (line 12). Moreover, LF sets flag as TRUE if it finds a compute node with, (a) minimum load, or (b) vCPUs used are maximum but the targeted server has minimum CPU utilization (line 14-18).

IV. SYSTEM MODEL OF VM ALLOCATION ALGORITHM

In this model we want to deploy the VM considering the load as a factor while its initial placement to the physical machine. For the better understating of the model all the constants and variables are listed in Table 2. Based on the mixed-integer linear programming, system problem is designed in the proposed model. The objective function of cloud deployment model is to maximize the number of VMs on the physical hosts as shown in equation (1).

Objective function:

$$Maximize \sum_{i=1}^{M} \sum_{j=1}^{N} X_{ij} \tag{1}$$

Algorithm 3 Load Filter

Require: self, s, p, TU_i, i, car, d, N, threshold
1: **procedure** HostPasses(self, s, p)
2: $Type_i$ ← p.get(i) ▷ i is instance type
3: **if** $Type_i \doteq TU_i$ **then**
4: **return** True
5: **end if**
6: vCPU ← *getVCPU()*
7: car ← *self .getCPUAllocationRatio*(s, p)
8: TvCPU ← $s.TvCPU * car$
9: **if** Tvcpu > 0 **then**
10: vCPU ← TvCPU
11: **end if**
12: l ← $\forall_{n \in N}$ LoadInfo-GDE-DB(n)
13: MinLoad $= \lceil (s.vcpus - used) \rceil * l$
14: **if** l $>$ threshold **then**
15: **return** Tvcpu - $s.vcpus - used >\doteq TU_i$.reqvcpu
16: **else**
17: **return** Tvcpu - (s.vcpus$-used *$ MinLoad) $>\doteq TU_i$.reqvcpu)
18: **end if**
19: **end procedure**

TABLE 2. Notations and description.

Notation	Description
N	the request's size as a number of requested VMs
M	total number of physical machines (PM) in data center
$Vcpu_i$	shows the requested core(s) of VM_i
X_{ij}	bivalent variable which is representing the VM_i assignment to the PM j
S_j	a variable which represents that PM j is used or not
$Vcpu_j$	indicates the maximum number of virtual cores of server j
L_i	denotes the load value of VM_i
$L_{T.max}$	denotes the maximum threshold value of load of server j
R_i	denoted as the required amount of RAM of VM i
OR_j	described as the RAM required for he overprovisioning on servers j
R_j	represented as the total amount of RAM of PM j

Subject to:

The event of VM placement method is summarized by formulating the objective function with all the conditions and constraints into the following equations.

$$X_{ij} = \begin{cases} 1, & \text{if the VM}_\text{i} \text{ is placed in server j;} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

$$S_j = \begin{cases} 1, & \text{if the server is used;} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

$$\sum_{j=1}^{N} X_{ij} \leq 1, \quad \forall i \in M \tag{4}$$

The number of linear constraints based on the optimization concept reflects the capacity limit of PMs, which subjects the

obvious facts that a PM can only host the number of VMs based on its remaining resources. In contrast, a single VM can only be deployed to one PM. In constraint equation (2) X_{ij} is explained as the deployment of VM_i on the physical server j. Furthermore, the value of $X_{ij} = 1$ when VM is deployed; otherwise, $X_{ij} = 0$, where, $\forall i \in M$ and $\forall j \in N$. In addition, the constraint (3) denotes that one VM will be deploy only on a single physical host which shows that the $S_j = 1$ for that VM. Moreover, cloud providers have to fulfill the request within the prescribed quota or SLA as explained in equation (2-3).

$$\sum_{i=1}^{M} Vcpu_i * X_{ij} \leq Vcpu_{j.\max} S_j - Vcpu_{j.used}, \quad \forall j = 1..., N \tag{5}$$

Each server has a number of limited cores (vCPUs). $Vcpu_j.max$ that cannot be exceeded when hosting or serving the VMs based on the remaining resources as presented in equation (5). For PMs authenticating the condition $Vcpu_j.max \geq Vcpu_{j.used}$ and $\text{Vcpu}_{used} \neq 0$, the total number of used PMs are lower bounded by $\lceil \frac{\sum_{j=1}^{N} Vcpu_{j.used}}{Vcpu_{j.\max}} \rceil$. This bound adds the following inequality to the model as shown in equation (6).

$$\sum_{j=1}^{N} S_j = \lceil \frac{\sum_{j=1}^{N} Vcpu_{j.used}}{Vcpu_{j.\max}} \rceil \tag{6}$$

$$\sum_{i=1}^{M} R_i X_{ij} \leq RS_j * ORS_j, \quad \forall j \in N \tag{7}$$

$$\sum_{i=1}^{M} L_i * X_{ij} \leq S_j L_{T\max \cdot j}, \quad \forall j \in N \tag{8}$$

The constraint in equation (7) denotes that RAM which is required to deploy the total number of VMs on each physical server should be less than or equal to the total amount of RAM of each physical host. Moreover, the constraints in equation (8) presents the VM deployment considering the load factor. L_i is the load value of virtual machine i on the physical machine X_{ij} that should be minimum or equal to the server capacity considering the value of threshold $L_{Tmax\,j}$ defined for the maximum load on each physical machine. In next section, results are conducted for random and static load factors in order to perceive the VM deployment behavior based on CPU utilization with respect to existing and proposed work.

V. EXPERIMENTAL SETUP

This section briefly explains our experimental setup to perform analysis. We considered a control environment to deploy VMs using OpenStack scheduler. Moreover, we have used the traditional computers rather than expensive and powerful machines (small industrial cloud replica). For the analysis, we deployed our small OpenStack cloud infrastructure comprising of four physical machines connected through a flat-DHCP networking module. The configured server are heterogeneous in terms of their resource capacity (RAM). Among the physical hosts one node is set responsible to act as a controller node with distinguished resource specifications in terms of Xeon(R), Intel(R), CPU E5620 with 2.40GHz, 32 GB RAM, and QEMU hypervisor. Besides, for the rest of three compute nodes physical servers have 16 GB RAM but forth compute node is configured on the controller node. For simplicity each compute node is termed as Edge1, Edge2, Edge3, Edge4 and EEdge1, EEdge2, EEdge3, and EEdge4 for existing and proposed technique, respectively. Moreover, Edge1 and EEdge1 are also represented as controller nodes as both have the maximum RAM specification among all other nodes. Furthermore, the capacities of physical servers are mentioned in Table 3.

TABLE 3. Capacity and physical server specification.

$Capacity$	$PhysicalServer(s)$
CPU type	Xeon(R)
Core(s) per socket	4
Thread(s) per socket	2
CPU(s)	8
CPU Freq, (GHz)	2.40
Architecture	$x86-64$
Kernel	3.11.0-26-generic
L1 Cache (KB)	128
L2 Cache (MB)	1
L3 Cache (MB)	12

A. LOAD DISTRIBUTION BEHAVIOR STUDY

To analyze load distribution among physical machines, we have characterized the load as static and dynamic. To generate static and random load we have considered a CPU intensive application that executes the multi-core Python application to increase CPU load. In both cases, VMs are created using same parameters such as, 2GB RAM, 1GB Disk Space, and 2vCPUs. To analyze static load generation case, equal numbers of VMs are deployed on each physical server. Moreover, we have used same CPU intensive application for each vCPU to generate a workload. In contrast, for dynamic load, we have developed an application that randomly generates a workload of different capacity. For simplicity and randomness, load generator's generated load is mapped between 0 and 2 random values. The number 0 specify that no CPU intensive application is executed on VM, whereas, values including 1 and 2 states that only 1 and 2 cores are fully utilized.

The core motivation of this study is to perceive the behavior of scheduler at the time of VM deployment or when it is shielded from any external influence. By using the initial set of experiments in next sections, performance of each compute node is evaluated and VMs are launched to analyze the CPU utilization based on the load factors.

VI. PERFORMANCE ANALYSIS OF EXISTING AND PROPOSED APPROACHES

For the better understanding of results, VM deployment with the placement sequence of existing study is presented

in Fig. 3 with respect to CPU utilization on each physical machine. In VM deployment model using the existing OpenStack scheduler, VMs are differentiated using distinct color schemes. The light-purple color triangle shows that first eight VMs are deployed on Edge1.

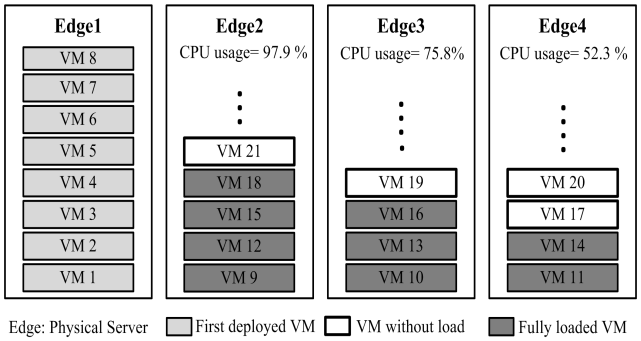


FIGURE 3. VM deployment using OpenStack cloud scheduler.

The VMs with dark gray-color are representing that VM is fully loaded by executing the CPU intensive application. Alternatively, white-color based VMs represents that VMs are deployed without load. Based on CPU utilization factor, Edge2 represents the current CPU usage capacity, which is 97.9% while four VMs are hosted on it at time "t". Alternatively, Edge3 and Edge4 represent 75.8% and 52.3% CPU usage statistics at time "t". Considering Fig. 3, based on non-optimized OpenStack scheduler, when 21st VM is deployed it is placed on the highly loaded physical host based on spread technique criteria in order to balance the RAM.

For the evaluation of proposed and existing OpenStack scheduler, we have presented the two scenarios in this section based on static and random load.

A. STATIC LOAD BASED VM DEPLOYMENT AND CPU UTILIZATION

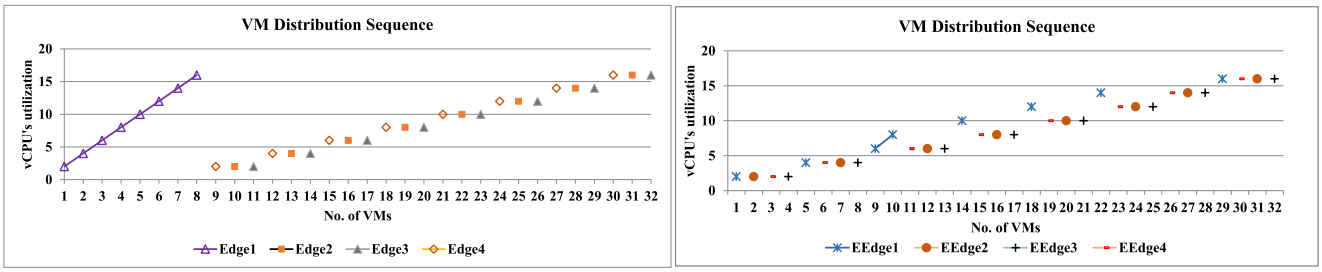
In our first experiment, we have modeled the relation between number of VMs deployment (x-axis represented as No. of VMs) and virtual core's utilization (y-axis) for all physical hosts. Each VM is deployed exactly using the 2 vCPUs so at y-axis the maximum value is 16 for each host. Therefore, total sixteen VMs are deployed on a single compute node while based on total number of VMs and there are thirty two VMs in total as presented on x-axis. In addition, Fig. 4a presents the VM distribution while using the existing OpenStack scheduler. In contrast, Fig. 4b presents the VM distribution based on proposed schedular. In order to differentiate the existing and proposed study the physical hosts are named as Edge and EEdge (extended Edge) based on existing and proposed scheduler and different color patterns are used to explain the single VM deployment on a specific host as shown in Fig. 4.

It is observed in Fig. 4a that during the launching of VMs, first eight VMs are created on Edge1 due to availability of sufficient RAM capacity. From ninth to onward all VMs are launched on Edge2, Edge3, and Edge4 (physical hosts) in a sequence because of spread technique evenly distributes the VMs in order to manage the RAM. In addition, in Fig. 4b VMs are not deployed based on RAM criteria. This figure shows that VMs are placed based on CPU utilization factor as explained in Fig. 5. At the deployment of seventh and eighth VM both VMs are deployed on EEdge3 host which shows that VM sequence is not same as represented in Fig. 4a. In addition, Fig. 5 is mapped based on the number of VM deployment as presented on x-axis in Fig. 4a and Fig. 4b and CPU utilization across that deployment on y-axis. Performance degradation is highlighted in Fig. 5 when multiple co-located VMs compile extensive computational tasks. The performance of each single VM is measures by running a CPU intensive application to impose load upon a particular vCPU. We have observed an increasing vCPU load of 25%, 50%, 75%, and 99% when the number of VMs are increased on every single physical server such as Edge and EEdge nodes. For the existing scheduler the graph is showing that upto point 8 at x-axis VMs are continuously deployed on Edge1 (as indicated with triangle) which increase the CPU utilization factor upto 99%. Moreover, the value from 9 to 32 at x-axis is showing that there is a specific deployment sequence which is followed for other compute nodes with the increased CPU load value. Furthermore, for the proposed scheduler VMs are deployed on the different EEdge nodes and shows the equal range of CPU utilization for each physical host.

In existing technique for the deployment of sixth, seventh, and eighth VM only one host Edge1 is selected which shows the 75.4%, 79.5% and 86.0% CPU utilization. In comparison of existing scheduler, EEdge4, EEdge2, EEdge3, and EEdge1 are selected for the deployment of same VMs with the CPU load 50%, 50.3%, and 50.5%, respectively. In this scenario, equal load is generated on deployed VMs in order to generate same CPU load as discussed in section V-A. Based on Fig. 5, it is observed that Edge1 has maximum RAM, therefore in existing technique the deployment sequence is dependent on RAM while the proposed study shows that VMs are deployed based of CPU utilization factor as first eight VMs are not directly deployed to EEdge1 and for other VMs the sequence of physical host is not repeated as presented for existing technique in Fig. 4a.

B. DYNAMIC LOAD BASED VM DISTRIBUTION CONSIDERING CPU UTILIZATION AS A FACTOR

In second experiment, VMs are deployed using OpenStack existing and proposed scheduler and number of applications are executed on each VM while considering the random load generator function. Each VM is deployed using the 2 vCPUs and load is generated based on random function values. In said Fig. 6, number of core utilization across the VM deployment sequence is represented to highlight that on which edge node VMs are deployed at sequence interval T as shown in and Fig. 6b. Moreover, Fig. 6 shows the relationship between core CPU utilization and VM deployment sequence on four physical servers including Edge1, Edge2, Edge3, and Edge4 for existing technique and EEdge1, EEdge2, EEdge3,



(a) Existing Scheduler **(b)** Proposed Scheduler

FIGURE 4. Static load distribution behavior of existing and proposed techniques.

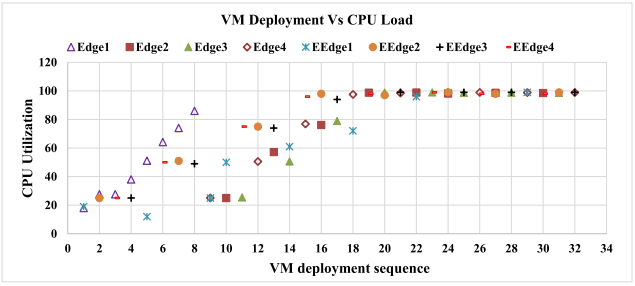


FIGURE 5. VM distribution vs CPU load.

and EEdge4 for proposed technique. In Fig. 6a at x-axis number of VMs are plotted with respect to CPU utilization behavior for each physical server. Further, based on proposed method, first four VMs among a sequence of VMs are deployed on EEdge4, EEdge2, EEdge3, and EEdge1 as shown in Fig. 6b. This happened due to random load generation, which causes each VM to have dissimilar CPU utilization. In the said figure, second VM on EEdge2 presents minimum CPU utilization as no execution profile is running on it because of the lowest load generated by the random function generator.

Based on deployment sequence in Fig. 6b, the CPU utilization is minimum for EEdge2 as compared to rest of the nodes. Therefore, seventh and eighth VMs are deployed on EEdge2 in order to balance the load factor. Furthermore, ninth VM that is second VM for EEdge3 shows the maximum CPU utilization as it executes four different execution profiles. Therefor, after third, seventh, and eighth VM's deployment, EEdge2 again leverages minimum load due to random load generation. As a result, tenth VM is also deployed on EEdge2. Based on comparison VM deployment sequence of proposed algorithms is not same as in existing algorithm presented in Fig. 6a.

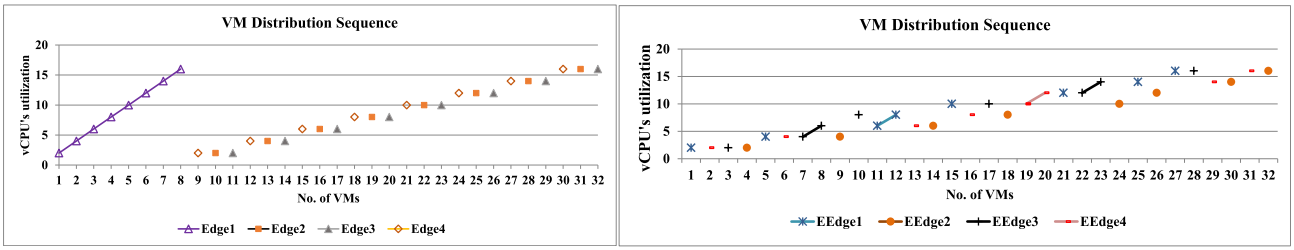
In Fig. 4b, VM deployment sequence is plotted across CPU usage at x-axis and y-axis. Based on Existing scheduler, first eight VMs are deployed on Edge1, which increases the aggregate CPU usage capacity to 95.90%. At the time of deployment of the ninth VM, the load on the edge2 reaches to 25.2%. For the Edge3 and Edge4, the load rises to 39.1% and 25.6%, respectively. When twelfth, thirteenth, and fourteenth VMs are deployed on Edge2, Edge3, and Edge4 reaches to their peak CPU load. At the deployment of fifteen VM based on placement criteria, VM is deployed on the maximum loaded host without considering the CPU utilization as shown in Fig. 7 and its deployment sequence is shown in Fig. 6a. For the proposed scheme Fig.7 represents that when initial VM is deployed on EEdge1, the CPU utilization is 5%. After first VM placement, the other compute hosts are loaded with minimum load compared to EEdge1. However, the second, third, and forth VMs are placed on EEdge4, EEdge3, and EEdge2, respectively. After the deployment of first four VMs, GDE (Fig. 2) perceived that EEdge1 has a minimum load. Therefore, it chooses EEdge1 to place next incoming VM. Moreover, at the time of ninth and tenth VM, the CPU utilization is 23% and 45%, respectively.

Based on experimental outcomes (static and dynamic load based sceneries) it is perceived that OpenStack scheduler does not observe the CPU utilization. Moreover, the number of CPUs associated with VMs and the available RAM are the major contributing factors which affects the decision making of a particular physical host. However, there is a need to balance the CPU load at the deployment of VMs in order to efficiently utilize the OpenStack cloud resources as presented in the proposed scheduler.

C. EXECUTION TIME ANALYSIS

In order to analyze the existing and proposed schedulerś performance we have analyzed the application execution time of each host while the deployment of VMs with repect to static and dynamic load factor as explained in Fig. 4 and Fig. 6.

Based on static load distribution, as mapped in Fig. 4a, and Fig. 4b application execution time is plotted in order to check the performance of existing and proposed schedulers named as filter scheduler and extended scheduler. In order to understand the results we have plotted the graph of each physical node where Filter_Scheduler Edge2, Filter_Scheduler Edge3, and Filter_Scheduler Edge4 represents the results of Edge2, Edge3 and Edge4 using the existing OpenStack scheduler. In contrast Extended_Scheduler Edge2, Extended_Scheduler Edge3, and Extended_Scheduler Edge4 shows the result while considering our proposed solution for nodes EEdge2, EEdge3, and EEdge4. Fig. 8 shows that existing and extended



(a) Existing Scheduler (b) Proposed Scheduler

FIGURE 6. Dynamic load distribution behavior of existing and proposed techniques.

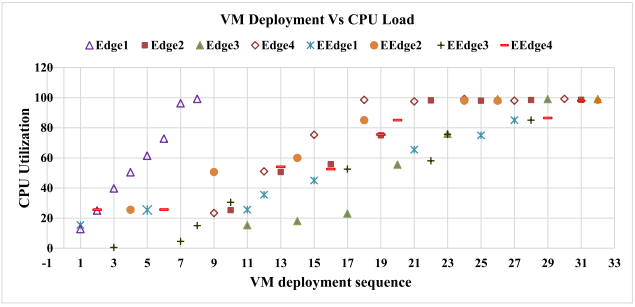


FIGURE 7. VM distribution vs CPU load.

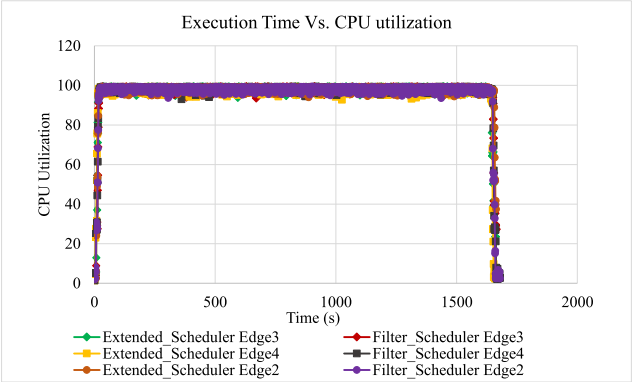


FIGURE 8. Execution time based on static load.

schedulers behavior is same when the load distribution nature depends on static load. As every single machine is deployed using the 2vCPUs and each vCPU is running with the compute intensive application which fully utilize the CPU resources. Therefore, for existing and proposed study each physical host shows the 100% utilization of CPU resources when all VMs are deployed on them and take same time for the completion of the tasks as running inside the VMs.

Fig. 9 shows the execution time when the VMs are deployed using the dynamic load generation function as shown in Fig. 6a, and Fig. 6b. In addition, Fig. 9 is reflecting the minimum execution time for each compute node when the load is distributed based on dynamic load distribution (Fig. 6a, and Fig. 6b). In comparison of existing scheduler, our proposed scheduler enhanced the performance

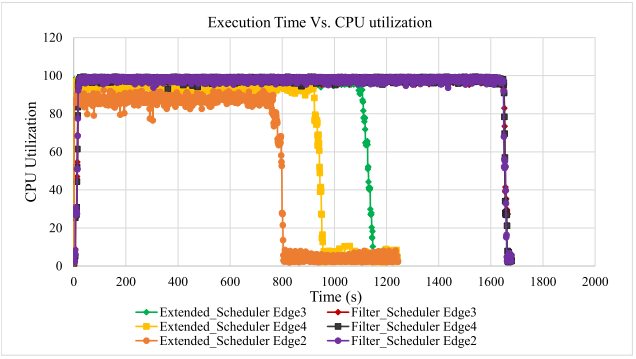


FIGURE 9. Execution time based on dynamic load.

of Edge2, Edge3, and Edge4 up-to 50%, 33%, and 44%, respectively. Based on result's analysis, it is concluded that Extended_Scheduler efficiently utilize the CPU resources.

VII. CONCLUSION AND FUTURE DIRECTIONS

In this paper, our focus is to critically analyze the OpenStack's scheduler for VM placement in a controlled environment in perspective of small-scale private cloud. In order to enable this experiment, we presented the lightweight extension of OpenStack compute service, which is based on three modules: a) Request Handler, b) Global Decision Engine, and c) Local Monitoring Engine. This extension allowed us to compare our experimentation results with the OpenStack benchmark Nova scheduler according to load distribution principles. Finally, our proposed VM deployment architecture evaluation shows the effectiveness of our implementation in achieving the load balancing objectives. The findings from this research can be summarized as follows: a) the decision making of existing schedulers for the initial placement of VMs to the physical servers did not consider the CPU utilization. b) The number of CPUs and RAM capacity at the time of VM deployment are the most important parameters in all types of schedulers. c) The application execution time is minimum when initial VM deployment considers the CPU utilization factor. Using our existing set of experiments, we concluded that considering the load as a metric for VM deployment is more feasible compared to existing metric of non-utilized RAM. Our current research covers the

computational resources. Though, in future we will extend our work considering the network and storage as a resource. Further, we can consider the optimized DC solutions for emerging blockchain workloads through cloud in order to provide the interconnected and truly transparent network. Moreover, Internet of Things (IoT) requires cloud computing paradigm to respond the resource requirements whereas load balancing plays a vital role to achieve the resource efficiency during the execution of IoT applications.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generat. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] Y. Jararweh, M. Jarrah, M. Kharbutli, Z. Alshara, M. N. Alsaleh, and M. Al-Ayyoub, "CloudExp: A comprehensive cloud computing experimental framework," *Simul. Model. Practice Theory*, vol. 49, pp. 180–192, Dec. 2014.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Dec. 2003.

[4] J. Espadas, A. Molina, G. Jiménez, M. Molina, R. Ramírez, and D. Concha, "A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 273–286, Jan. 2013.

[5] A. K. Sidhu and S. Kinger, "Analysis of load balancing techniques in cloud computing," *Int. J. Comput. Technol.*, vol. 4, no. 2, pp. 737–741, 2013.

[6] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing," in *Proc. Nat. Conf. Parallel Comput. Technol. (PARCOMPTECH)*, Feb. 2013, pp. 1–8.

[7] S.-C. Wang, K.-Q. Yan, W.-P. Liao, and S.-S. Wang, "Towards a load balancing in a three-level cloud computing network," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.*, vol. 1, Jul. 2010, pp. 108–113.

[8] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," in *Proc. 3rd IEEE/IFIP Int. Conf. Central Asia Internet*, Sep. 2007, pp. 1–7.

[9] O. M. Elzeki, M. Z. Reshad, and M. A. Elsoud, "Improved max-min algorithm in cloud computing," *Int. J. Comput. Appl.*, vol. 50, no. 12, pp. 22–27, 2012.

[10] O. Litvinski and A. Gherbi, "Openstack scheduler evaluation using design of experiment approach," in *Proc. 16th IEEE Int. Symp. Object/Compon./Service-Oriented Real-Time Distrib. Comput. (ISORC)*, Jun. 2013, pp. 1–7.

[11] G. Patel, R. Mehta, and U. Bhoi, "Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing," *Procedia Comput. Sci.*, vol. 57, pp. 545–553, Jan. 2015.

[12] G. Ming and H. Li, "An improved algorithm based on max-min for cloud task scheduling," in *Recent Advances in Computer Science and Information Engineering*. Berlin, Germany: Springer, 2012, pp. 217–223.

[13] M. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud," *Future Gener. Comput. Syst.*, vol. 81, pp. 156–165, Apr. 2018.

[14] M. H. U. Rehman, E. Ahmed, I. Yaqoob, I. A. T. Hashem, M. Imran, and S. Ahmad, "Big data analytics in industrial IoT using a concentric computing model," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 37–43, Feb. 2018.

[15] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT edge computing with lightweight virtualization," *IEEE Netw.*, vol. 32, no. 1, pp. 102–111, Jan./Feb. 2018.

[16] S. Zahoor, N. Javaid, A. Khan, B. Ruqia, F. J. Muhammad, and M. Zahid, "A cloud-fog-based smart grid model for efficient resource utilization," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 1154–1160.

[17] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-placement algorithms for on-demand clouds," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Nov./Dec. 2011, pp. 91–98.

[18] W. Iqbal, M. N. Dailey, and D. Carrera, "Black-box approach to capacity identification for multi-tier applications hosted on virtualized platforms," in *Proc. Int. Conf. Cloud Service Comput.*, Dec. 2011, pp. 111–117.

[19] G. Kousiouris, T. Cucinotta, and T. Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *J. Syst. Softw.*, vol. 84, no. 8, pp. 1270–1291, 2011.

[20] Q. Wang and C. A. Varela, "Impact of cloud computing virtualization strategies on workloads' performance," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2011, pp. 130–137.

[21] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via VM multiplexing," in *Proc. 7th Int. Conf. Autonomic Comput.*, 2010, pp. 11–20.

[22] A. Beloglazov and R. Buyya, "OpenStack neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 5, pp. 1310–1333, 2015.

[23] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: A real case based on OpenStack Cloud," *Future Gener. Comput. Syst.*, vol. 32, pp. 118–127, Mar. 2014.

[24] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis, "Schedule optimization for data processing flows on the cloud," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 289–300.

[25] L. T. X. Phan, Z. Zhang, Q. Zheng, B. T. Loo, and I. Lee, "An empirical analysis of scheduling techniques for real-time cloud-based data processing," in *Proc. IEEE Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Dec. 2011, pp. 1–8.

[26] S. Sotiriadis, N. Bessis, F. Xhafa, and N. Antonopoulos, "Cloud virtual machine scheduling: Modelling the cloud virtual machine instantiation," in *Proc. 6th Int. Conf. Complex, Intell., Softw. Intensive Syst.*, Jul. 2012, pp. 233–240.

[27] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *J. Syst. Archit.*, vol. 60, no. 9, pp. 726–740, 2014.

[28] S. J. Mullender, "Predictable cloud computing," *Bell Labs Tech. J.*, vol. 17, no. 2, pp. 25–39, Sep. 2012.

[29] A. Kačeniauskas, R. Pacevič, V. Starikovičius, A. Maknickas, M. Staškūnienė, and G. Davidavičius, "Development of cloud services for patient-specific simulations of blood flows through aortic valves," *Adv. Eng. Softw.*, vol. 103, pp. 57–64, Jan. 2017.

[30] A. L. Garcia, L. Zangrando, M. Sgaravatto, V. Llorens, S. Vallero, V. Zaccolo, S. Bagnasco, S. Taneja, S. D. Pra, D. Salomoni, and G. Donvito, "Improved Cloud resource allocation: How INDIGO-DataCloud is overcoming the current limitations in cloud schedulers," 2017, *arXiv:1707.06403*. [Online]. Available: https://arxiv.org/abs/1707.06403

[31] G. Einziger, M. Goldstein, and Y. Sa'ar, "Faster placement of virtual machines through adaptive caching," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Apr./May 2019, pp. 2458–2466.

[32] D.-N. Le, B. Seth, and S. Dalal, "A hybrid approach of secret sharing with fragmentation and encryption in cloud environment for securing outsourced medical database: A revolutionary approach," *J. Cyber Secur. Mobility*, vol. 7, no. 4, pp. 379–408, 2018.

[33] V. N. Van, N. Q. Long, G. N. Nguyen, and D.-N. Le, "A performance analysis of openstack open-source solution for IaaS cloud computing," in *Proc. 2nd Int. Conf. Comput. Commun. Technol.* New Delhi, India: Springer, 2016, pp. 141–150.

[34] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," in *Proc. 9th Int. Conf. Fuzzy Syst. Knowl. Discovery*, May 2012, pp. 2457–2461.

[35] G. V. Laszewski, J. Diaz, F. Wang, and G. C. Fox, "Comparison of multiple cloud frameworks," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 734–741.

[36] M. Liaqat, S. Ninoriya, J. Shuja, R. W. Ahmad, and A. Gani, "Virtual machine migration enabled cloud resource management: A challenging task," 2016, *arXiv:1601.03854*. [Online]. Available: https://arxiv.org/abs/1601.03854

[37] B. Hu and H. Yu, "Research of scheduling strategy on OpenStack," in *Proc. Int. Conf. Cloud Comput. Big Data*, Dec. 2013, pp. 191–196.

[38] A.-M. Ammar, J. Luo, Z. Tang, and O. Wajdy, "Intra-balance virtual machine placement for effective reduction in energy consumption and SLA violation," *IEEE Access*, vol. 7, pp. 72387–72402, 2019.

[39] X. Xiao, Y. Xia, F. Zeng, W. Zheng, X. Sun, Q. Peng, Y. Guo, and X. Luo, "A novel coalitional game-theoretic approach for energy-aware dynamic VM consolidation in heterogeneous cloud datacenters," in *Proc. Int. Conf. Web Services*. Cham, Switzerland: Springer, 2019, pp. 95–109.
[40] M. Liaqat, V. Chang, A. Gani, S. H. A. Hamid, M. Toseef, U. Shoaib, and R. L. Ali, "Federated cloud resource management: Review and discussion," *J. Netw. Comput. Appl.*, vol. 77, pp. 87–105, Jan. 2017.
[41] O. Litvinski and A. Gherbi, "Experimental evaluation of OpenStack compute scheduler," *Procedia Comput. Sci.*, vol. 19, pp. 116–123, Jan. 2013.
[42] S. Latif, S. M. Gilani, R. L. Ali, M. Liaqat, and K.-M. Ko, "Distributed meta-brokering P2P overlay for scheduling in cloud federation," *Electronics*, vol. 8, no. 8, p. 852, 2019.

MISBAH LIAQAT received the B.S. degree (Hons.) in computer science from COMSATS University, Pakistan, in 2013, and the Ph.D. degree from the University of Malaya, Kuala Lumpur, Malaysia, in 2017. She was associated as a BSP RA with the Center for Mobile Cloud Computing Research (C4MCCR), University of Malaya, during her Ph.D. degree. She is currently an Assistant Professor with Air University Islamabad, Pakistan. Her research interests include cloud scheduling, cloud resource management, mobile cloud computing, sensor cloud, VM migration, the Internet of Things, and wireless sensor networks.

ANJUM NAVEED received the bachelor's degree (Hons.) in engineering (software engineering) from the National University of Sciences and Technology, Pakistan, and the Ph.D. degree from the University of New South Wales, Australia, in 2009. He is currently a Postdoctoral Researcher with the University of Malaya (UM), Malaysia. His topic was Channel Assignment in MR-MC Wireless Mesh Networks. Since 2009, he has been an Assistant Professor with the National University of Sciences and Technology. He has worked on a number of research as well as research and development projects. He has completed research projects worth USD 400K. He has also completed industrial projects worth approximately USD 200K. During his stay at NUST, Pakistan, he has supervised two Ph.D. students and 14 master's students. He is currently supervising three Ph.D. and five master's students. His research interests include resource allocation in mobile clouds, VM migration in mobile clouds, network performance analysis and optimization, and interference analysis in wireless networks.

RANA LIAQAT ALI received the M.Sc. degree in electronics from Quaid-I-Azam University, in 1999, the M.S. degree in electrical engineering from Air University Islamabad, Pakistan, in 2006, and the Ph.D. degree from the Department of Electrical Engineering, COMSATS, Islamabad, in 2013, with a research work at Lancaster University (InfoLab21), U.K. He was a System Engineer with Panasonic, in 1999. After that, he joined Air University Islamabad. He is currently an Assistant Professor with the Department of Physics (Electronics), CIIT, Islamabad. He is also with DigiSys and RF Systems Research groups, COMSATS. His research interests include adaptive systems, including array processing, beamforming, Mic arrays and Smart arrays, wireless networks, and cloud computing.

JUNAID SHUJA received the M.S. degree from the COMSATS Institute of Information Technology (CIIT), Abbottabad, in 2012, and the Ph.D. degree from the University of Malaya, in 2017. He is currently an Assistant Professor with CIIT. His primary research interests include ARM emulation and SIMD instruction cross-platform execution and other research interests include data center energy efficiency, sustainable cloud computing, ARM and GPU-based servers for energy efficient cloud computing, and cloud computing in general.

KWANG-MAN KO received the B.S. degree from Wonkwang University, in 1991, and the M.S. and Ph.D. degrees in computer engineering from Dongguk University, in 1993 and 1998, respectively. He was a Professor with Kwangju Women's University, from March 1998 to August 2001. He joined the faculty in September 2001. He is currently a Professor with the School of Computer and Information Engineering, Sangji University. His current research interest includes Energy-oriented Architecture Description Language (EoADL). He constructs a programmable architecture and generates an energy-oriented SDK through the structural, behavioral, and energy consumption management information in embedded systems, and SoC design areas.