Renewable Energy-Based Multi-Indexed Job Classification and Container Management Scheme for Sustainability of Cloud Data Centers

Neeraj Kumar, *Senior Member, IEEE*, Gagangeet Singh Aujla, *Member, IEEE*, Sahil Garg, *Member, IEEE*, Kuljeet Kaur, *Member, IEEE*, Rajiv Ranjan, *Senior Member, IEEE*, and Saurabh Kumar Garg, *Member, IEEE*

***Abstract*—Cloud computing has emerged as one of the most popular technologies of the modern era for providing on-demand services to the end users. Most of the computing tasks in cloud data centers are performed by geodistributed data centers which may consume a hefty amount of energy for their operations. However, the usage of renewable energy resources with appropriate server selection and consolidation can mitigate the energy related issues in cloud environment. Hence, in this paper, we propose a renewable energy-aware multi-indexed job classification and scheduling scheme using container as-a-service for data centers sustainability. In the proposed scheme, incoming workloads from different devices are transferred to the data center which has sufficient amount of renewable energy available with it. For this purpose, a renewable energy-based host selection and container consolidation scheme is also designed. The proposed scheme has been evaluated using Google workload traces. The results obtained prove 15%, 28%, and 10.55% higher energy savings in comparison to the existing schemes of its category.**

***Index Terms*—Container-as-a-service (CoaaS), data centers (DCs), energy management, renewable energy sources (RESs), sustainability.**

Manuscript received November 25, 2017; revised January 6, 2018; accepted January 24, 2018. Date of publication February 1, 2018; date of current version May 2, 2019. The work of G. S. Aujla and N. Kumar is supported by the Department of Science and Technology (DST), India, and Austrian Government through project sanctioned under bilateral cooperation agreement vide reference number INT/AUSTRIA/BMWF/P-03/2017. Paper no. TII-17-2715. *(Corresponding author: Neeraj Kumar.)*

N. Kumar is with Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala (Punjab) 147004, India (e-mail: neeraj.kumar@thapar.edu).

G. S. Aujla is with the Computer Science and Engineering Department, Chandigarh University, Mohali (Punjab), India (e-mail: gagi_aujla82@yahoo.com; gagangeet.aujla@ieee.org).

S. Garg and K. Kaur are with the École de Technologie Supérieure, Université du Québec, Montréal, Canada (e-mail: garg.sahil1990@gmail.com; kuljeet0389@gmail.com).

R. Ranjan is with the Department of Computer Science, Newcastle University, Newcastle upon Tyne NE1 7RU, U.K., and also with Chinese Academy of Sciences, Beijing 100049, China (e-mail: raj.ranjan@ncl.ac.uk).

S. K. Garg is with the Department of Computing and Information System, Faculty of Engineering and ICT, University of Tasmania, Hobart, TAS 7005, Australia (e-mail: Saurabh.Garg@utas.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2018.2800693

I. Introduction

WITH the rapid popularity of latest computing and communication paradigms such as cloud computing, Internet of Things (IoT), and software-defined networks (SDN), there is a steep increase in the design and deployment of various applications such as smart city and automatic cars using these environments across the globe [1]–[4]. Among these, cloud computing has emerged as one of the most revolutionary paradigms that provide on-demand customized services to cater to the requirements of wide range of end users in smart cities [5]. Cloud environments using virtualization technologies share same or different types of physical resources such as CPU, network, and memory to execute different applications as per the end user's demands in an isolated manner. In recent years, with an increase in the number of users and their applications, the demand of various types of cloud resources (processor, storage, etc) has increased rapidly. Such a huge demand of cloud resources has paved the way for creation of massive geodistributed Data Centers (DCs) across the globe [6]. For example, nearly 3 million DCs are hosting various online activities such as email, social media, and e-commerce using 12 million servers in US only [7]. However, with an exponential growth in DCs, the operational cost associated with routine functioning (such as resource management and energy consumption) of DCs is also increasing.

Energy consumption is one of the major contributors to the overall operational cost of DCs. Mostelic and Brandic [8] highlighted that the energy consumed by DCs increased from 70 to 330 billion kWh from 2000 to 2007. By 2020, this figure is expected to reach 1000 billion kWh [8]. According to a survey (2014) [8], beyond 2015, only 8.5% of total global DCs are capable enough to handle the user's requests with its existing infrastructure. Therefore, by 2020, 75% DCs are expected to expand to double the size as was in 2010. With such an expansion, it is expected that there would be manifold increase in the energy consumption of DCs. According to a report published in the New York Times (2012) [9], DCs deployed in the US will need more than 17 power plants to meet their energy requirements. Presently, DCs in the US consume energy equal to the generation capacity of 34 power plants of 500 MW.

Hence, in order to meet the growing energy demand of DCs, high use of fossil fuels results in an increase in the carbon emissions. As per Natural Resources Defense Council (NRDC) survey (US, 2013) [7], 97 MMT of carbon emissions were released from DCs only. By 2020, this figure is expected to reach 147 MMT. Therefore, such an alarming situation of energy consumption and carbon emissions associated with DCs poses a serious threat to the vision of green cities. Hence, to cope with this situation, cloud service providers (CSPs) are adopting energy-efficient techniques (such as Virtual Machine (VM) consolidation, migration, and workload prediction) to handle the routine activities of DCs [10]. By adopting energy-efficient techniques, cloud computing sector can witness manifold benefits such as reduction in energy consumption, operational cost, and carbon emissions. However, CSPs have to also simultaneously face a tough challenge of preserving service level of agreement (SLA), quality of service (QoS), and reliability of the cloud services. Therefore, to handle these issues, nowadays, a lightweight technology namely: *Container-as-a-Service* (*CoaaS*) is being adopted to design energy-efficient DCs [11].

A. Container-as-a-Service

CoaaS is an emerging technology in which the resources are kept inside a container from where different applications can access them as per their requirements. Generally, the resources in a VM are emulated over a hypervisor and a guest operating system (OS). However, *CoaaS* runs on a Docker engine rather than a hypervisor. Hence, they can also be utilized as an alternative to the hypervisor-based virtualization to execute time-critical applications. Container orchestrator, a run-time scheduler, is one of the components of container that handles various scheduling decisions as per service availability. In a *CoaaS* model, all the containers share single OS kernel with different levels of abstraction for all the applications. On contrary, the VMs require their own virtualized network, BIOS, CPU, and OS. Hence, PaaS and SaaS use containers instead of VMs [12].

Containers are smaller in size as VMs, so they provide a quick startup with an improved performance, and better compatibility. These can run either on the host or inside a VM [12]. Moreover, container-based virtualization can also support lightweight and energy-efficient migration within and outside VMs. This helps in live system updates using the underlying host OS thereby eliminating the requirement of hardware emulation. According to [13], containers migration consumes less energy with minimum resource wastage as compared to the entire VM. Moreover, containers use few number of OSs for higher level of application execution in contrast to the VMs [12]. Table I depict the comparison of containers and VMs.

After analysis of the above facts, it is evident containers can provide increased performance, higher efficiency, and lower energy dissipation through higher execution speed, quick startup and shared kernel instances. These manifold benefits make containers adequate for real-time applications and data collection. In other words, these light weight containers are better options than typical virtualization utilized in current cloud DCs.

Fig. 1 shows the *CoaaS* model deployed over infrastructure layer using docker engine. The docker contains all the components required for runtime task execution such as configuration files and databases. It enables an additional layer of abstraction and virtualization. It is used for Linux OS and leverages the resource isolation feature of the underlying kernel. This, in turn, permits containers to be independently executed within the kernel instance; thereby, eliminating the need to initiate and maintain VMs [12].

TABLE I
COMPARISON OF VM AND CONTAINER TECHNOLOGIES

Parameter	Containers	VMs
Virtualization type	OS-level virtualization	Completely virtualized
Quality assurance	High	Relatively low
User space	Separate	Common
Kernel state	Require booting	Operational
Start and stop time	Less than 50 ms	30–40 and 5–10 s
Startup overhead	Low	Relatively high
Memory usage	Low	High
CPU usage	Low	High
Migration speed	High	Low
Cost (overheads)	Low	High
Guest OS	Not required	Multiple guest OS
Resource sharing	YES (Kernel instances, OS, bare file system)	NO (own BIOS, CPU, OS and disk storage.)
Tools	Docker, CoreOS-RKT, OpenVZ, Sandboxie	Virtual Box, Xen, VM-Ware
Standardization	Runtime Specification v1.0 and Image Format Specification v1.0: OCI	Well standardized system

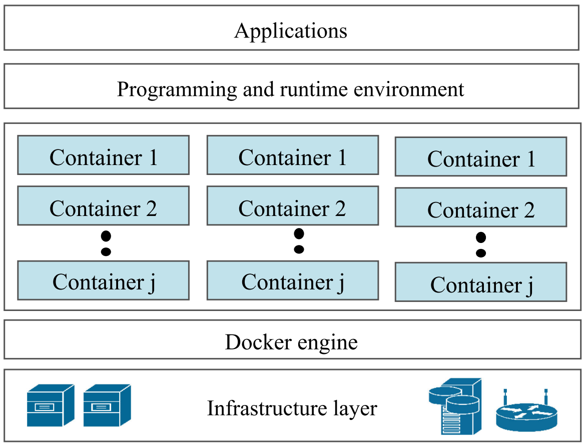

Fig. 1. CaaS model.

B. Sustainability of DCs

Sustainability is a way of reducing negative human impact on environment through various socio-ecological methods. For the survival of mankind and other organisms, healthy and sustainable ecosystems and environments are necessary. For this purpose, considerable efforts are being applied for the energy transition from fossil fuels to ecologically sustainable systems. Sustainable energy is one of the best ways to serve the present needs without compromising the ability of future generations to cope with their needs. Moreover, it is projected that by 2040, renewables can be expected to replace coal-based energy generation [14]. According to a recent survey [15], by 2020, carbon emissions associated with energy consumption of DCs are expected to be double as compared to 2011. With such an increase in the growth rate of carbon emissions, the environmental sustainability level can also degrade proportionately. Therefore, to cope with this challenge, the focus of DCs is slowly shifting from the mere usage of renewables toward sustainability of DCs, i.e., 100% renewable energy. For example, Apple tops the list of CSPs by achieving 100% clean energy index [15]. Moreover, the vision of green cities is incomplete without sustainable DCs. Hence, sustainable energy can provide manifold benefits such as-reduced carbon emissions, grid load, and operational cost along with participation in environmental commitment.

However, the intermittent nature of sustainable energy is one of the biggest challenge that needs to be handled effectively. For this purpose, the *CoaaS* model can be a viable method to deal with the irregularities of renewable energy sources (RES); thereby, achieving the vision of sustainable development.

C. Research Contributions of This Paper

To address the above challenges, the amalgamation of *CoaaS* model with sustainable energy to design efficient mechanism for sustainable DCs can be a viable solution. Moreover, the limited research proposals related to *CoaaS* [12], [13], [16], [17] and sustainable DCs [14], [18]–[21] make it necessary to take a leap ahead in this direction. Inspired by these facts, following contributions are presented:

1) A CoaaS model is used over physical servers in system architecture comprising of geodistributed DCs connected to RES. In this architecture, two types of controllers-global and local are used.
2) A multi-indexed job classification and scheduling scheme for DCs using containers is designed. To attain the sustainability, the global controller schedules the workloads to DCs with sufficient amount of renewable energy using renewable energy-aware host selection scheme.
3) In order to minimize the energy consumption of DCs, a container consolidation and migration scheme is designed for load balancing among various containers.

D. Organization

The rest of this paper is organized as follows. Section II presents the related work. Section III describes the system model and problem formulation. Section IV presents the proposed scheme. In Section V, performance evaluation of proposed technique using real workload is presented. Finally, we conclude the paper with key findings in Section VI.

II. Related Work

Many existing research works have explored energy related aspects of DCs. For example, Forestiero *et al.* [22] proposed a hierarchical approach for workload assignment and migration to reduce the energy cost of geodistributed DCs. In another work, Qiu *et al.* [23] proposed a scheme to minimize operational cost of geodistributed DCs by dynamic migration of user requests. All these proposals focused on workload scheduling and migration among geodistributed DCs with respect to spatial and temporal energy price variations. However, by doing so, the performance of cloud services is compromised. In this regard, Jin *et al.* [24] proposed an optimization scheme to analyze the effect of operational cost on the performance of cloud services hosted by geodistributed DCs. Chiang *et al.* [25] proposed an efficient green control algorithm to solve constrained optimization problem in order to make tradeoff between performance and energy saving policies. Dabbagh *et al.* [26] highlighted that VM placement and consolidation are promising solutions to design an energy-efficient DC. In this direction, many existing proposals explored these aspects to minimize the energy consumption of DCs. For example, Beloglazov *et al.* [27] proposed an adaptive heuristics for dynamic VM consolidation to reduce energy consumption while preserving SLA. Wang *et al.* [28] presented a virtual batching technique for server consolidation to achieve maximum energy conservation for DCs.

Recently some research work have used CoaaS to minimize the energy consumption of DCs. Piraghaj *et al.* [13] presented energy-efficient container consolidation scheme for cloud DC. In another work, Dai *et al.* [16] proposed a QoS-aware scheme for implementing CaaS to provide energy efficiency in DCs. Similarly, Gutierrez-Garcia *et al.* [17] presented a load management technique using VM migrations to save energy, but the authors highlighted that use of containers can be more beneficial and energy efficient. In another work, Kaur *et al.* [12] proposed an energy-efficient scheme using container migrations at edge devices. Hence, after analyzing these proposals, it is evident that containers can be useful to reduce energy consumption of DCs while ensuring SLAs. However, none of the above proposals have focused on the sustainability DCs using containers.

Some of the researchers have proposed techniques for partial utilization of sustainable energy for handling the energy consumption of DCs. For example, Yu *et al.* [29] designed an optimization technique for energy management to deal with power outages using RES and backup generators. Polverini *et al.* [30] proposed a thermal-aware scheduling of batch jobs in geodistributed DCs. Moving a step ahead, some of the recent works have proposed different methods for designing 100% sustainable DCs. For example, Guo *et al.* [18] proposed an energy and network-aware workload management scheme for sustainable DCs using thermal storage. Similarly, Aujla *et al.* [20] presented a Stackelberg game for renewable energy-aware resource allocation for the sustainability of DCs. In another work, Aujla *et al.* [21] presented a workload consolidation and scheduling scheme for designing sustainable DCs in edge-cloud environment. Similarly, Chen *et al.* [19] proposed a workload and energy management scheme for the sustainability of DCs using RES. Aujla *et al.* [14] presented energy management scheme for the sustainability of DCs using electric vehicles. However, none of these proposals have adopted *CoaaS* model to design sustainable DCs.

III. System Model

A geodistributed cloud environment having I DCs connected to RES and grid with energy storage system (ESS) is considered. Fig. 2 depicts the two hierarchical controllers, i.e., global and local deployed in this model. The global controller selects DCs where the incoming jobs can be scheduled in such a way that energy consumption is sustained using RES. The status of all the jobs being executed at a specific DC is regularly updated by this

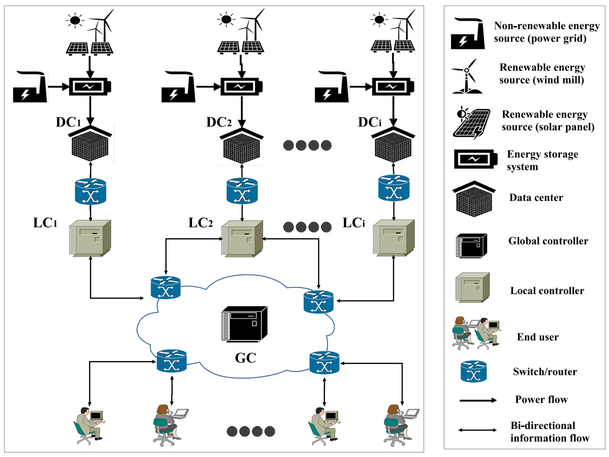
DC1
DC2
DCi
LC1
LC2
LCi
GC
Non-renewable energy source (power grid)
Renewable energy source (wind mill)
Renewable energy source (solar panel)
Energy storage system
Data center
Global controller
Local controller
End user
Switch/router
Power flow
Bi-directional information flow

Fig. 2. System architecture for geodistributed DCs.

controller. The local controllers handle the allocation of J containers ($\mathbb{C}$) deployed over P servers. Some of the preliminaries about the proposed model are illustrated as below.

A. Workload Model

Let us consider an incoming workload ω that consists of m number of κ type of jobs. This workload is classified and added to different queues for scheduling. The global controller decides the DC to which the job can be routed. At time t, the type κ jobs are modeled using Poisson distribution with an arrival rate $a(t)$. The type κ jobs scheduled at the ith DC follow the queues dynamics [18] as given below:

$$Q_i^\kappa(t+1) = \max[Q_i^\kappa(t) - Q_i(t)] + m_i^\kappa(t)$$
$$\text{s.t. } 0 < \sum_{\kappa=1}^{m} Q_i(t)\tilde{c_\kappa} \le s_p\delta_p \tag{1}$$

where $m_i^\kappa(t)$ is the number of type κ jobs routed to the ith DC, $\tilde{c_\kappa}$ is the required computing resources to handle type κ jobs, s_p is the total number of servers allocated, and δ_p is the processing speed of each server.

B. QoS Model

SLA violation occurs if the available capacity of resources at a DC is less than the resources required to process the workload. The SLA violations (SLA_p^v) of the pth server of the ith DC are given as [27]

$$\text{SLA}_p^v = \frac{1}{p}\sum_{p=1}^{P}\frac{t_p^{\text{thr}}}{t_p^{\text{act}}}D_p^{\text{mg}} \tag{2}$$

where t_p^{thr} is the time for which the pth server has threshold utilization level, t_p^{act} is total active time, and D_p^{mg} in the performance degradation due to migration.

Using SLA violations, a SLA metric for the jth container is defined as [13]

$$\text{SLA}_j^v = \sum_{p=1}^{P}\sum_{j=1}^{J}\frac{s_{\text{rq}}(\mathbb{C}_j, t_v) - s_{\text{al}}(\mathbb{C}_j t_v)}{s_{\text{rq}}(\mathbb{C}_j t_v)} \tag{3}$$

where $s_{\text{rq}}(\mathbb{C}_j, t_v)$ is server amount requested by the jth container on the pth server at time t_v and $s_{\text{al}}(\mathbb{C}_j, t_v)$ is server amount allocated by the jth container at time t_v.

In order to ensure SLAs, at some occasions, container migrations are required. The delay occurred due to these migrations (d_j^{mg}) for the jth container is given as

$$d_j^{\text{mg}} = d_{\text{nt}}\varphi_j(t) \tag{4}$$

where d_{nt} is delay offered by underlying network and φ_j is the migration rate.

C. Energy Consumption Model

The energy consumption of the ith DC is given as

$$E_i = \sum_p E_i^p + E_i^{\text{net}} + E_i^c \tag{5}$$

where E_i^p, E_i^{net}, and E_i^c are energy consumed by processors, network, and cooling, respectively.

Energy consumption of a server has linear relation with its CPU load. Therefore, the energy consumption of the pth server at the ith DC is given as

$$E_i^p = E_{\text{idl}}^p + (E_{\max}^p - E_{\text{idl}}^p)\, U_i^p \tag{6}$$

where E_{idl}^p is the energy consumed by the idle pth server, $E_{\max}^p$ is the maximum energy that the pth server can consume, and U_i^p is the level of utilization of the pth server.

The level of utilization of the pth server at the ith DC depends on the amount of resources consumed R^p at time t and maximum capacity of processor R_{mx}^p and is given as

$$U_i^p = \left(\frac{R^p(t)}{R_{\text{mx}}^p}\right) \times 100. \tag{7}$$

Similarly, the level of utilization of the jth container running on the pth server is given as

$$U_p^j = \left(\frac{R^j(t)}{R_{\text{mx}}^j}\right) \times 100 \tag{8}$$

where $R^j(t)$ denotes the resources consumed by the jth container at time t and R_{mx}^j is the maximum capacity of the container.

The energy consumption related to network infrastructure depends on: 1) fixed part ($E_{\text{sw}}^{\text{nt}}$): switches, fan, chasis, etc., and 2) dynamic part ($E_{\text{pr}}^{\text{nt}}$): active ports. Therefore, the network related energy consumption at the ith DC is given as

$$E_i^{\text{net}} = E_{\text{sw}}^{\text{nt}} + E_{\text{pr}}^{\text{nt}}. \tag{9}$$

Equation (9) can be extended on the basis of incoming flow passing through the ports as

$$E_i^{\text{net}} = E_{fx}^{\text{nt}} \times \sum_{\text{sw}}(t_f^{\text{en}} - t_f^{\text{st}}) + E_{dy}^{\text{nt}} \times \sum_{\text{sw}}(t_f^{\text{en}} - t_f^{\text{st}}) \tag{10}$$

where t_f^{en} and t_f^{st} denotes the end and start time of fth flow and E_{fx}^{nt} and E_{dy}^{nt} refers to the energy consumed by fixed part and dynamic part, respectively.

D. Energy Generation Model

In this paper, two types of sources of energy (renewable and grid) are considered to handle the energy demand of DCs. Renewable energy is utilized as a primary source of energy and the grid energy is utilized only in worst case. The energy generated (E_i^r) by RES is given as

$$E_i^r = E_i^{\mathrm{pv}} + E_i^{\mathrm{wn}} \tag{11}$$

where E_i^{pv} is the energy generated by PV panels and E_i^{wn} is the energy generated by wind turbines.

The energy E_i^{pv} generated by capturing Sunlight using PV panels is given as [20]

$$E_i^{\mathrm{pv}} = \left(1 - L_{\mathrm{pv}}^{\mathrm{tp}}\right) \eta\, S_i^{\mathrm{pv}} \cos(\alpha)\, \digamma \tag{12}$$

where $\digamma$ is solar radiations, η is conversion efficiency of panel, $\cos\alpha$ is radiation angle of sunlight, S_i^{pv} is size of panel, and L^{tp} is temperature exceedance loss.

The energy E_i^{wn} generated by a typical wind turbine is given as [20]

$$E_i^{\mathrm{wn}} = \frac{1}{2}\left[Ç\, \rho\, \hat{a}\, \nu^3\right] L^{\mathrm{tp}} \tag{13}$$

where Ç is the rotar efficiency, ρ is the air density, $\hat{a}$ is the area swept by rotar blades, and ν is the wind speed.

The energy generated by RES is stored in ESS connected to the DCs. The energy stored E_i^{ess} in ESS deployed at the ith DC is given as

$$E_i^{\mathrm{ess}}(t) = \left(E_i^{\mathrm{ess}}(t-1) + E_i^r - E_i\right) \iota_i^{\mathrm{ess}} \tag{14}$$

where ι_i^{ess} is the ESS's self-discharge rate.

E. Problem Formulation

Using above preliminaries, various cases are considered for defining the objective function.

1) Case 1: If $(E_i \le E_i^r)$, then the excess energy generated by the RES is stored in ESS and is used by the DCs in case of deficit. The excess energy E_i^{ex} stored in the ESS is given as

$$E_i^{\mathrm{ex}} = E_i^r - E_i. \tag{15}$$

2) Case 2: If $(E_i > E_i^r)$, then the deficit in the energy demand of DCs is fulfilled from the excess energy E_{ex} stored in the ESS. The energy deficit E_i^{df} is given as

$$E_i^{\mathrm{df}} = E_i - E_i^r. \tag{16}$$

In this case, there may be a subcase in which energy deficit E_i^{df} is not fulfilled by excess energy E_i^{ex} stored in ESS. In such a case, the energy is drawn from grid. The energy drawn from grid E_i^{gr} is given as

$$E_i^{\mathrm{gr}} = E_i^{\mathrm{df}} - E_i^{\mathrm{ex}} \tag{17}$$

However, in such a case, the workload is assigned to another DCs having sufficient amount of energy generated by the RES.

Keeping in view of all the above-discussed cases, the objective function is illustrated as

$$E_{\mathrm{obj}} = \text{Minimize} \sum_{t,i} \left(E_i(t) - E_i^r(t)\right) \tag{18}$$

subject to following constraints

$$E_i^r(t), E_i(t) > 0 : \forall i \tag{19}$$

$$t_i^{\mathrm{rs}} \le t_{\mathrm{mx}}^{\mathrm{rs}} \tag{20}$$

$$0 < E_i^{\mathrm{ex}} \le E_{\mathrm{ess}}^{\mathrm{mx}} \tag{21}$$

where $E_i(t)$, $t_{\mathrm{mx}}^{\mathrm{rs}}$, and $E_{\mathrm{ess}}^{\mathrm{mx}}$ denote response time, maximum desirable response time, and maximum storage capacity of ESS, respectively.

IV. Proposed Scheme

The proposed scheme consists of multiple steps each given in subsequent sections.

A. Multi-Indexed Classification and Scheduling Scheme

In this section, a multi-indexed workload classification and scheduling scheme is presented. The incoming workload is classified into three categories; 1) high priority, 2) high resource requirement, and 3) best effort. Initially, the controller classifies ω using priority index, load index, and availability index. Bloom filter is used to index the workload at the controller. The high priority workload is indexed using a priority index and the high resource requirement workload is classified into load index. The best effort workload is indexed into an availability index for serving it with the best possible availability of resources. All these indexes are formulated using Baye's theorem to find the conditional probability.

A priority index $\wp$ is used to classify the incoming job request requiring high priority as

$$\wp = P\left(\frac{c_j}{m}\right) = \frac{P(c_j)P(\frac{m}{c_j})}{P(m)} \tag{22}$$

where $P(c_j)$ and $P(\omega)$ depict the probabilities of the jth containers available capacity and incoming job requests, respectively.

Apart from the priority, the load index (ℓ) is defined to classify ω requiring high computing resources. Hence, ℓ is defined on the basis of load $(\imath_i)$ at the ith DC as

$$\ell = P\left(\frac{\imath_m}{\imath_i}\right) = \frac{P(\imath_m)P(\frac{\imath_i}{\imath_m})}{P(\imath_i)} \tag{23}$$

where $P(\imath_m)$ and $P(\imath_i)$ denote the mth job and the ith DC load probabilities, respectively.

Finally, the workload not classified in $\wp$ or ℓ is indexed using the availability index (α) as

$$\alpha = P\left(\frac{\Re_{rq}}{\Re_{av}}\right) = \frac{P(\Re_{rq})P(\frac{\Re_{av}}{\Re_{rq}})}{P(\Re_{av})} \tag{24}$$

where $P(\Re_{rq})$ and $P(\Re_{av})$ represents the probabilities of requested resources and available resources, respectively.

Once ω is classified, then it is scheduled using Algorithm 1. In this algorithm, ω is added to a $Q_i(t)$ at time *t*. For all elements in the queues, classifier is used to segregate ω on the basis of $\wp$, ℓ, and α (lines 1–3). If ω is classified in $\wp$, then it is added to a new queue (Q_{i1}) and thereby scheduled using Preemptive Prioritized Round Robin (PPRR) scheme (for details refer [20]) (lines 4–33). If ω is classified in ℓ, then it is added to a new queue (Q_{i2}) and it is also scheduled using PPRR scheme (lines 34–39).

Algorithm 1: Classification and Scheduling of Workloads.

Input: ω, available i DC list (aDCL)
Output: allocated DC_i

```
 1: for ($\omega$=1; $\omega \leq$ m; $\omega$++) do        ▷ $m \leftarrow$ incoming workloads
 2:     $\omega \leftarrow Q_i(t)$                          ▷ Add to queue
 3:     call.classifier($\omega$)
 4:     if $\omega \leftarrow \wp$ then
 5:         $Q_{i1} \leftarrow \omega$
 6:         call.PPRR($Q_{i1}$)                             ▷ Initiate PPRR
 7:         while $\omega \leftarrow Q_{i1}$ do
 8:             $\forall$ $\omega$
 9:             mutex==1
10:             TimeQuantum==3
11:             if mutex==1 then
12:                 mutex==0
13:                 for Requests in Queue do
14:                     BurstTime=BurstTime-TimeQuantumn
15:                     if BurstTime $\neq$ 0 then
16:                         allot resource $\leftarrow \Re_{rq}$
17:                         $\Re_{rq} \leftarrow$ updateQueue
18:                     end if
19:                     for time==0; time $\leftarrow$ BurstTime; time++ do
20:                         tempQueue $\leftarrow \Re_{rq}$
21:                         $tempQueue.sort(\Re_{rq})$
22:                         for Queue $\leftarrow \Re_{rq}$ and BurstTime $\neq$ 0 do
23:                             BurstTime=BurstTime-1;
24:                             allot resource $\leftarrow \Re_{rq}$
25:                             tempQueue $\leftarrow$ updateQueue
26:                             mutex=1;
27:                         end for
28:                     end for
29:                     $\Re_{rq} = \Re_{req} \rightarrow next$
30:                 end for
31:                 mutex=1
32:             end if
33:         end while
34:         if $\omega \leftarrow \ell$ then
35:             $Q_{i2} \leftarrow \omega$
36:             call.PPRR($Q_{i2}$)                         ▷ Initiate PPRR
37:             while $\omega \leftarrow Q_{i2}$ do
38:                 Repeat Step 8-32
39:             end while
40:             if $\omega \leftarrow \alpha$ then
41:                 $Q_{i3} \leftarrow \omega$
42:                 call.PPRR($Q_{i3}$)                     ▷ Initiate PPRR
43:                 while $\omega \leftarrow Q_{i3}$ do
44:                     Repeat Step 8-32
45:                 end while
46:             end if
47:         end if
48:     else
49:         $Q_{i4} \leftarrow \omega$
50:         call.FCFS($Q_{i4}$)
51:     end if
52:     aDCL.allocate($DC_i$) $\leftarrow \forall\omega$
53:     aDCL $\leftarrow$ update($\omega$ status)
54: end for
55: while ($Q_{in}$) do
56:     $Q_{i1} \leftarrow$updatepriorityqueue
57:     while ($Q_j$) do
58:         $Q_{i2} \leftarrow$updateloadqueue
59:         while ($Q_k$) do
60:             $Q_{i3} \leftarrow$updateavailabilityqueue
61:         end while
62:     end while
63: end while
64: while $\forall(Q_{in})$ do
65:     updateoutgoingquery $\leftarrow Q_{in}$
66: end while
```

Moreover, if ω is classified in α, then it is added to a new queue Q_{i3} and scheduled using PPRR scheme (lines 40–45). In this way, ω is classified differently on the basis of various indexes and added to the different queues. All queues are scheduled separately using PPRR scheme. However, the workloads which are not classified in any of the aforementioned queues are added to queue Q_{i4} and are scheduled using First Come First Serve (FCFS) (lines 48–51). Now, each workload is allocated to an appropriate DC listed in available DC list (aDCL). Once the DC are allocated, then the aDCL status is updated (lines 52–54). Finally, the respective queues are updated accordingly (55–66).

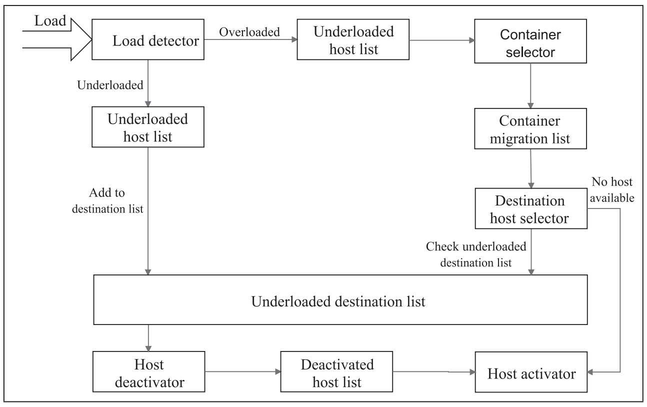

Fig. 3. Flowchart of *CoaaS* consolidation scheme.

B. Renewable Energy-Aware Host Selection Scheme

In this section, Algorithm 2 is presented for renewable energy-aware host selection. The objective of this algorithm is to utilize the resources optimally and to select the container, server, and DC in such a way that energy consumption is sustained using the energy generated by RES. In this algorithm, the subsequent steps are performed for all the servers deployed in available DCs. Initially, $\Re_{av}(p,i)$ available with the pth server of the ith DC are checked. If $(\Re_{rq}(i) \leq \Re_{av}(p,i))$, then the level of utilization U_i^p of each server is computed using (7). If (U_i^p) is equal to (U_{pi}^{id}), then the server is idle and so it is shutdown and added to list M_1 (1–8). Otherwise, for each active container in the pth server of the ith DC, the level of utilization (U_p^j) is computed using (8). If (U_p^j) is equal to (U_{jp}^{id}), then it is having no workload and can be deactivated and stored in list M_2 (lines 9–14). Rest of all the containers are active and are stored in list M_3 (lines 10–19). Initially, all the container for the pth server of the ith DC are allocated from list M_3. Now, if in case list M_3 is empty, then first the deactivated containers in list M_2 are activated and utilized. But, if still the demand is not fulfilled, then the servers in list M_1 are restarted and containers are deployed (lines 20–28).

Once, the resource availability is confirmed at DCs, then the renewable energy available at each DC is compared with the energy demand of DCs. For this purpose, E_i of ith DC is calculated using (5) and the energy generated by RES is calculated using (11) (29–30). Now, if $(E_i \leq E_i^r)$, then such DCs are added in $aDCL_1$. In this case, E_{ex} generated by the RES is calculated using (15) and stored in ESS (lines 31–33). However, if $(E_i > E_i^r)$, then E_{ex} stored in ESS is used to manage E_{df} calculated using (16). Such DCs are added in $aDCL_2$. But, if still the energy requirement is not fulfilled, then E_i^{gr} calculated using (17) is drawn from the grid (lines 34–40). DCs which are having sufficient renewable energy resources are sorted and added to aDCL and are selected to schedule various workloads by the global controller (line 41).

C. Container Consolidation and Migration Scheme

In container consolidation scheme, active load on a running host is consolidated on minimum number of servers to run the applications. This consolidation is performed to minimize the energy consumption of DCs. Fig. 3 shows the flow diagram of

Algorithm 2: Renewable Energy-Aware Host Selection.

```
Input: $\Re_{rq}(i)$, $\Re_{av}(p,i)$, $U_i^{mx}$
Output: $aDCL$, $\mathtt{C}_j$
1:  for (i = 1; i ≤ I; i++) do          ▷ $I \leftarrow$ Number of DCs
2:    for (p = 1; p ≤ P; p++) do          ▷ $P \leftarrow$ Number of servers
3:      Check available resources ($\Re_{av}(p,i)$)
4:      if ($\Re_{rq}(i) < \Re_{av}(j,i)$) then
5:        Compute ($U_i^p$) using (7)
6:        if ($U_i^p == U_{pi}^{id}$) then
7:          Shutdown $\leftarrow$ $k$th server ($S_k$)
8:          Store $\leftarrow M_1$
9:        else
10:         for (j = 1; j ≤ J; j++) do          ▷ $J \leftarrow$ Number of container
11:           Compute ($U_p^j$) using (8)
12:           if ($U_p^j == U_{jp}^{id}$) then
13:             Shutdown $\leftarrow$ $j$th container ($\mathtt{C}_j$)
14:             Store $\leftarrow M_2$
15:           else
16:             Store $\leftarrow M_3$
17:           end if
18:         end for
19:       end if
20:     end if
21:     if ($M_3$ is empty) then
22:       Activate container ($\mathtt{C}_j$) $\leftarrow M_2$
23:       Assign jobs $\leftarrow M_2$
24:     else
25:       Activate new servers $\leftarrow M_1$
26:       Deploy container ($\mathtt{C}_j$) $\leftarrow$ server in $M_1$
27:     end if
28:   end for
29:   Calculate $E_i$ using (5)
30:   Calculate $E_i^r$ using (11)
31:   if ($E_i \leq E_i^r$) then
32:     Add DCs in $aDCL_1$
33:     Store $E_i^{ex}$ calculated using (15) in ESS
34:   else
35:     Draw $E_i^{df}$ calculated (16) is drawn from $E_i^{ex}$ stored in ESS
36:     Add DCs in $aDCL_2$
37:     if ($E_i^{df} < E_i^{ex}$) then
38:       Draw energy ($E_i^{gr}$) using (17) from grid
39:     end if
40:   end if
41:   Sort selected DCs in descending order in aDCL
42: end for
```

the proposed consolidation scheme. Various steps of consolidation scheme are discussed as below.

1) Load detector: The current load of a running host is detected and classified as overloaded or underloaded on the basis of Overload(OL)/Underload(UL) threshold.

2) Underload host list (UHL): All the hosts detected as underloaded are added into this list.

3) Underloaded destination list (UDL): The underloaded hosts are added to this list. If any of the host in this list is used to migrate the containers from overloaded hosts, then such hosts remain active. The host that are not utilized are sent to host deactivator (HD).

4) HD: The underloaded hosts which are not utilized are deactivated.

5) Deactivated host list (DHL): All the underloaded hosts which are deactivated are added to this list.

6) Overload host list (OHL): All the hosts classified as overloaded are added to this list.

7) Container selector: If a running host is overloaded, then the active containers to be migrated are selected.

8) Container migration list (CML): This list contains all the containers identified for migration from an overloaded host to another underloaded or new hosts.

9) Destination host selector (DHS): In this step, destination hosts where the containers in migration list can be shifted are selected from UDL. However, if no host is available in the list, then a new host is activated from DHL.

10) Host activator: If DHS is not able to find any underloaded host, then a host is activated from DHL.

For synchronization of containers located within the same or different machines with respect to data updates, a variable $\alpha_{jd_ad_b}$ is considered. Here, d_a is the data located on the host server and d_b denotes the data located server where $\mathtt{C}_j$ is to be migrated. All the machines synchronize with the controller and a task is completed only when all the data updates are done. The following condition needs to be satisfied:

$$\sum_{j=1} j <= J = J \times \alpha_{jd_ad_b}. \tag{25}$$

Algorithm 3 shows working of the consolidation scheme.

V. Results and Discussions

The proposed scheme is evaluated using extensive simulations for three geodistributed DCs having 200 heterogeneous servers using Google workload traces [31]. In simulation setup, a startup and deployment delay of 0.5 s and 100 s, respectively, is considered for each container. Table II shows the server and container configurations. Network bandwidth is assumed to be 1 GB/s and 300 KB/s for servers and containers.

A. Sustainability of DCs

The impact of proposed scheme on the sustainability of DCs is evaluated on the basis of two scenarios: 1) 12 h, and 2) 24 h, which are discussed as below.

1) 12-h scenario: In this scenario, the proposed scheme is compared with Random Host Selection (RHS) and First Fit Host Selection (FFHS) scheme. The renewable energy traces used in the 12-h scenario are scaled up version of [18]. Using the container consolidation scheme, the energy consumption of DCs reduces to a great extent. Fig. 4 (a) shows the energy consumption of DCs. FFHS scheme consumes highest amount of energy to handle the incoming workload. The proposed scheme helps us to utilize DC resources in an optimal manner which results in maximum utilization of resources. By doing so, a large

Algorithm 3: Container Consolidation Scheme.

```
Input: Host $S_k$, $\mathtt{C}_j$, $R_{rq}$
Output: Destination host ($S_k^{des}$)
 1: for (p = 1; p ≤ P; p++) do        ▷ $P \leftarrow$ Number of servers
 2:   Compute ($U_i^p$) using (7)
 3:   if ($U_i^p < U_{ul}^{thr}$) then
 4:     Add $S_k \rightarrow$ UHL
 5:     Add $S_k \rightarrow$ UDL
 6:   else if ($U_i^p > U_{ol}^{thr}$) then
 7:     Add $S_k \rightarrow$ OHL
 8:     Select $\mathtt{C}_j$
 9:     Add $\mathtt{C}_j \rightarrow$ CML
10:     Pass control $\rightarrow$ DHS
11:     Check $\rightarrow$ UDL
12:     Check $\rightarrow R_{rq}$
13:     if $S_k^{des}$ in UDL then
14:       Migrate $\mathtt{C}_j \rightarrow S_k^{des}$
15:       if $\alpha_{jd_ad_b}$ == 0 then
16:         Update $\mathtt{C}_j$
17:       else
18:         No update required
19:       end if
20:     else
21:       Activate $S_k^{new} \rightarrow$ DHL
22:     end if
23:   else
24:     Host is neither overloaded nor underloaded
25:   end if
26:   Deactivate unallocated hosts in UDL
27:   Add deactivated hosts $\rightarrow$ DHL
28: end for
```

TABLE II
SERVER AND CONTAINER CONFIGURATIONS

Server Configuration					Container Configuration		
Server type	CPU	Memory (GB)	E_{id}^p (kW)	E_{max}^p (kW)	Container Type	CPU MIPS	Memory (GB)
1	4 cores	64	100	150	1	4658	128
2	8 cores	128	120	200	2	9320	256
3	16 core	256	150	250	3	18636	512

number of servers are deactivated to save energy. The proposed scheme saves 15% and 28% energy as compared to RHS and FFHS. Energy generated by RES (solar and wind) is shown in Fig. 4(b). Using the energy generated by RES, the demand of DCs is fulfilled.

At initial time slot, energy generated by solar panels is low when no Sunlight is available, so it creates a deficit of energy at DCs. Such energy deficit is fulfilled by drawing energy from grid. This is because initially, the ESS is also having no energy stored in it. Fig. 4(c) shows the mapping of energy consumption of DCs and energy generation by RES. It is evident from results that more than 80% of the scenario witnessed excess energy generation that is stored in ESS. This excess energy stored in ESS is used to manage the energy deficit whenever it arises. Fig. 4(c) clearly shows that after initial time slot, only two occurrences of energy deficit exists (at 0500 h and 1100 h to 1200 h). At 0500 h, the energy stored in ESS is used to power DCs. Similarly, at other time slot with energy deficit, the required energy is supplied by ESS. Fig. 4(d) shows the cases of energy deficit and excess with respect to the mapping of consumption and generation of energy at DCs.

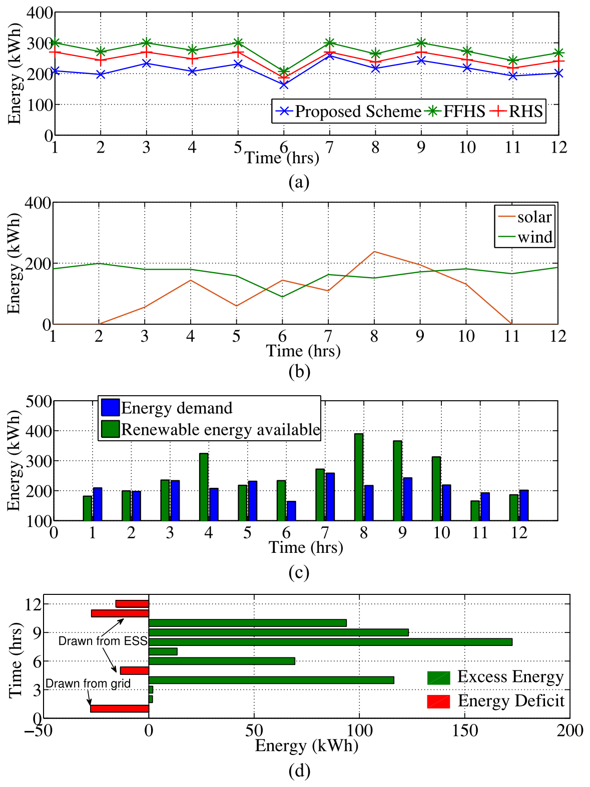

Fig. 4. Evaluation results for sustainability (12-h scenario). (a) Energy consumption. (b) Energy generation. (c) Mapping of energy consumption and generation. (d) Excess and deficit of energy.

2) 24-h Scenario: In this scenario, the proposed scheme is compared with Stackelberg game based resource allocation for sustainable DCs presented in [20]. The renewable energy traces used in the 24-h scenario are scaled up version of [20]. Fig. 5 (a) shows the energy consumed by the DCs using the proposed scheme. It clearly depicts that the energy consumed using proposed scheme is 10.55% lower in comparison to the existing scheme in [20]. Fig. 5(b) shows the energy generated by RES (solar and wind). It is evident form the figure that the renewable energy available is high in initial hours (0800 to 2000 h). However, the renewable energy availability is lower in the later hours (2001 to 0759 h). However, using the proposed scheme, the energy consumption of DCs is effectively sustained with the available renewable energy using ESS. The excess energy available in initial hours is stored in the ESS. The energy stored in ESS is used for handling the routine activities of the DCs in later hours when generation is at lower end. Therefore, it is evident from the results that the proposed scheme behaves effectively in 24-h scenario also.

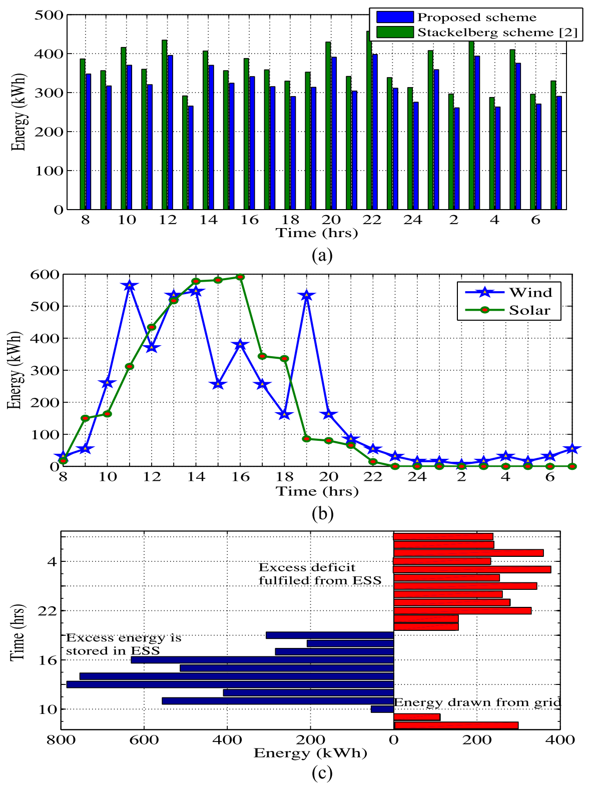

Fig. 5. Evaluation results for sustainability (24-h scenario). (a) Energy consumption. (b) Energy generation. (c) Mapping of Energy consumption and generation.

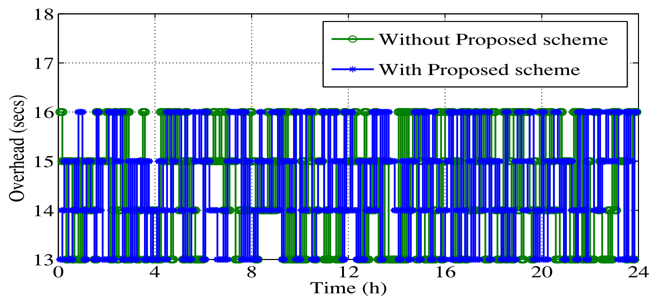

Fig. 6. Overhead.

B. Overhead

Although the *CoaaS* model exhibits negligible startup, migration overheads. However, there is some overhead due to the network flow. Fig. 6 shows the overhead generated using the proposed scheme. It is evident that the proposed scheme exhibits lower overhead in contrast to other schemes.

C. Impact of OV/UL on DCs

The above discussion clearly shows that the container consolation scheme helps us to reduce the energy consumption of DCs. By doing so, it becomes easy for the global controller to sustain energy consumption of DCs using RES. However, the OL and UL threshold value has a significant impact on the energy consumption, SLA violation, and container migration rate. In this section, the impact of change in OL and UL threshold is analyzed. For this purpose, the proposed scheme (A), RHS (B), and FFHS (C) are compared. The OL threshold is considered as 80%, 90%, and 100%. Similarly, UL threshold is considered as 40%, 50%, and 60%.

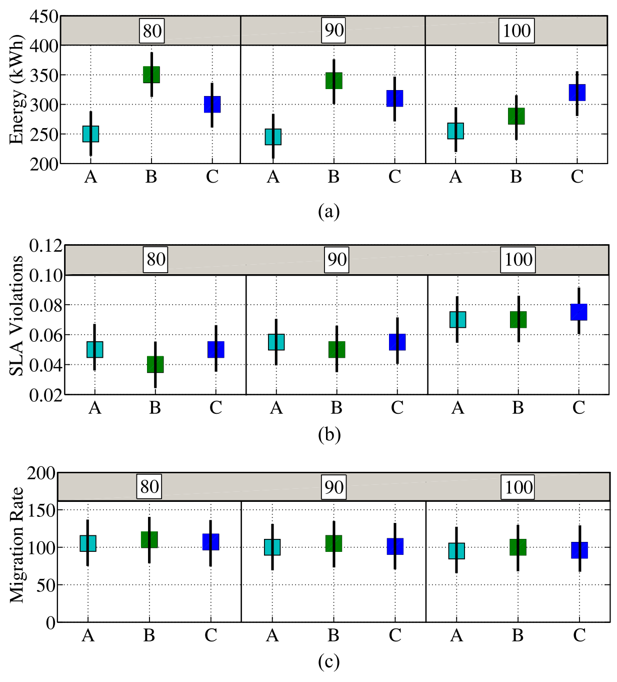

Fig. 7. Evaluation results with respect to OL threshold. (a) Energy consumption versus OL threshold. (b) SLA violations versus OL threshold. (c) Container migration rate versus OL threshold.

1) Variation of OL: In this analysis, energy consumption of proposed scheme (A) is lower than other two schemes (B and C). Fig. 7(a) shows that the energy consumption for proposed scheme for 80% and 90% OL threshold is lower than 100% OL threshold. Moreover, the SLA violations of the proposed scheme is lower than 0.05 for 80% and 90% OL threshold as shown in Fig. 7, but it shows an increase in SLA violations when OL threshold is considered as 100%. Therefore, it is evident that OL threshold is not suitable at 100%. Fig. 7(c) shows that by increasing the OL threshold, the migration rate for all three scheme decreases. This is because less number of hosts are identified as overloaded. Moreover, the decrease in container migration rate results in the lower number of new container creation. The above discussion shows that the proposed scheme consumes less energy and ensures viable SLAs. Moreover, 80% and 90% OL thresholds prove to be viable and 100% OL thresholds perform worst for all schemes.

2) Variation of UL: In this analysis, energy consumption of proposed scheme (A) is lower than other two schemes (B and C). Fig. 8(a) shows that the energy consumption for proposed scheme for 40% and 60% UL threshold is lower than 50% OL threshold. Moreover, the SLA violations of the proposed scheme are below 0.05 for all UL threshold as shown in Fig. 8(b). Moreover, Fig. 8(c) shows that by increasing the UL threshold, the migration rate for all three scheme increases. This is because more number of hosts are identified as underloaded. The above discussion shows that the proposed scheme consumes less energy and ensures SLAs.

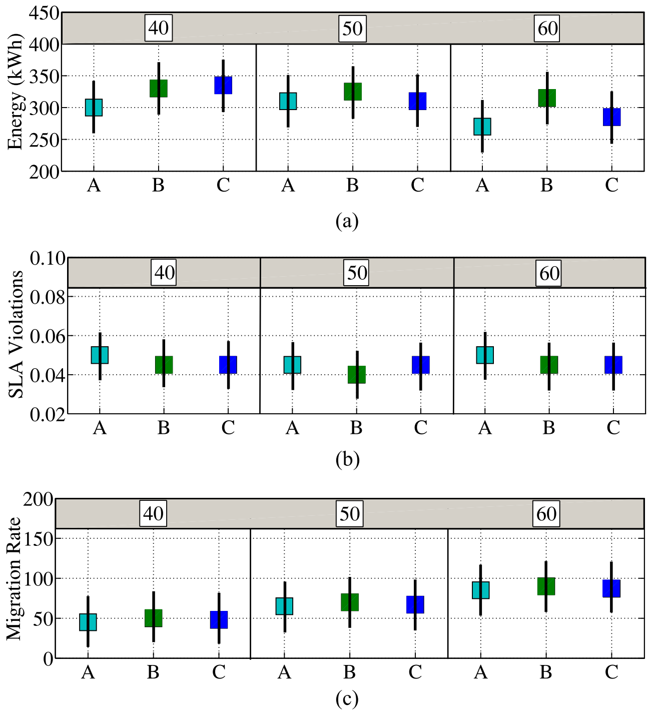

Fig. 8. Evaluation results with respect to UL threshold. (a) Energy consumption versus UL threshold. (b) SLA violations versus UL threshold. (c) Container migration rate versus UL threshold.

VI. Conclusion

In this paper, a *CoaaS* model has been deployed in a geodistributed cloud environment connected to RES through ESS. The major objective of the paper is to sustain the energy consumption of DCs using RES. To achieve this objective, a renewable energy-aware multi-indexed job classification and scheduling approach is designed. The proposed scheme allocates the incoming workload to those DCs which have sufficient amount of renewable energy to handle the incoming jobs. A global controller is used to handle all the incoming workload, classify it according to priority, load, and availability indexes and finally schedule it to DCs selected using renewable energy-aware host selection scheme, but due to intermittent nature of RES, there may be case when energy deficit occurs. In such a case, the energy stored in ESS is utilized. Moreover, a container consolidation scheme is designed to minimize the energy consumption of physical hosts. The proposed scheme has been evaluated using Google workload traces for two scenarios; 12 h and 24 h. The results obtained show that the proposed scheme successfully achieves its objective of sustaining energy consumption of DCs using RES. Moreover, the effect of OL and UL thresholds on the container consolidation scheme is also analyzed.

References

[1] Z. Deng, W. Han, L. Wang, R. Ranjan, A. Y. Zomaya, and W. Jie, "An efficient online direction-preserving compression approach for trajectory streaming data," *Future Gener. Comput. Syst.*, vol. 68, pp. 150–162, 2017.

[2] L. Wang, W. Song, and P. Liu, "Link the remote sensing big data to the image features via wavelet transformation," *Cluster Comput.*, vol. 19, no. 2, pp. 793–810, 2016.

[3] L. Wang, J. Zhang, P. Liu, K.-K. R. Choo, and F. Huang, "Spectral-spatial multi-feature-based deep learning for hyperspectral remote sensing image classification," *Soft Comput.*, vol. 21, no. 1, pp. 213–221, 2017.

[4] L. Wang, K. Lu, P. Liu, R. Ranjan, and L. Chen, "IK-SVD: Dictionary learning for spatial big data via incremental atom update," *Comput. Sci. Eng.*, vol. 16, no. 4, pp. 41–52, 2014.

[5] L. Wang, Y. Ma, J. Yan, V. Chang, and A. Y. Zomaya, "Pipscloud: High performance cloud computing for remote sensing big data management and processing," *Future Gener. Comput. Syst.*, vol. 78, pp. 353–368, 2018.

[6] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Rajan, "Optimal decision making for big data processing at edge-cloud environment: An SDN perspective," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 778–789, Feb. 2018, doi: 10.1109/TII.2017.2738841.

[7] J. Whitney and P. Delforge, "Data center efficiency assessment–scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers," *Rep. IP NRDC Anthesis*, pp. 14–08, 2014.

[8] T. Mastelic and I. Brandic, "Recent trends in energy-efficient cloud computing," *IEEE Cloud Comput.*, vol. 2, no. 1, pp. 40–47, Jan. 2015.

[9] B. Walsh, *The Surprisingly Large Energy Footprint of the Digital Economy*, 2013. [Online]. Available: http://science.time.com/2013/08/14/power-drain-the-digital-cloud-is-using-more-energy-than-you-think/

[10] A. Khosravi, S. K. Garg, and R. Buyya, "Energy and carbon-efficient placement of virtual machines in distributed cloud data centers," in *Proc. Eur. Conf. Parallel Process.*, 2013, pp. 317–328.

[11] R. Morabito, "Power consumption of virtualization technologies: An empirical investigation," in *Proc. 2015 IEEE/ACM 8th Int. Conf Utility Cloud Comput.*, 2015, pp. 522–527.

[12] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.

[13] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service," in *Proc. IEEE World Congr. Services*, Jun. 2015, pp. 31–38.

[14] G. S. Aujla and N. Kumar, "SDN-based energy management scheme for sustainability of data centers: An analysis on renewable energy sources and electric vehicles participation," *J. Parallel Distrib. Comput.*, vol. 117, pp. 228–245, Jul. 2018.

[15] Global e-Sustainability Initiative, "GeSI smarter 2020: The role of ICT in driving a sustainable future," Global e-Sustainability Initiative, Brussels, Belgium, 2012.

[16] X. Dai, Y. Wang, J. M. Wang, and B. Bensaou, "Energy-efficient planning of QoS-constrained virtual-cluster embedding in data centres," in *Proc. IEEE 4th Int. Conf. Cloud Netw.*, Oct. 2015, pp. 267–272.

[17] J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines," *IEEE Trans. Services Comput.*, vol. 8, no. 6, pp. 916–929, Nov. 2015.

[18] Y. Guo, Y. Gong, Y. Fang, P. P. Khargonekar, and X. Geng, "Energy and network aware workload management for sustainable data centers with thermal storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2030–2042, Aug. 2014.

[19] T. Chen, Y. Zhang, X. Wang, and G. B. Giannakis, "Robust workload and energy management for sustainable data centers," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 651–664, Mar. 2016.

[20] G. S. Aujla, M. Singh, N. Kumar, and A. Zomaya, "Stackelberg game for energy-aware resource allocation to sustain data centers using RES," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2017.2715817.

[21] G. S. Aujla and N. Kumar, "MEnSuS: An efficient scheme for energy management with sustainability of cloud data centers in edge-cloud environment," *Future Gener. Comput. Syst.*, vol. 86, pp. 1279–1300, Sep. 2018.

[22] A. Forestiero, C. Mastroianni, M. Meo, G. Papuzzo, and M. Sheikhalishahi, "Hierarchical approach for efficient workload management in geo-distributed data centers," *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 1, pp. 97–111, Mar. 2017.

[23] X. Qiu, H. Li, C. Wu, Z. Li, and F. C. M. Lau, "Cost-minimizing dynamic migration of content distribution services into hybrid clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3330–3345, Dec. 2015.

[24] X. Jin, F. Zhang, L. Wang, S. Hu, B. Zhou, and Z. Liu, "Joint optimization of operational cost and performance interference in cloud data centers," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 697–711, Oct.–Dec. 2017, doi: 10.1109/TCC.2015.2449839.

[25] Y. J. Chiang, Y. C. Ouyang, and C. H. Hsu, "An efficient green control algorithm in cloud computing for cost optimization," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 145–155, Apr. 2015.

[26] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Netw.*, vol. 29, no. 2, pp. 56–61, Mar. 2015.

[27] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Comput., Practice Exp.*, vol. 24, no. 13, pp. 1397–1420, 2012.

[28] Y. Wang and X. Wang, "Virtual batching: Request batching for server energy conservation in virtualized data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1695–1705, Aug. 2013.

[29] L. Yu, T. Jiang, and Y. Cao, "Energy cost minimization for distributed internet data centers in smart microgrids considering power outages," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 120–130, Jan. 2015.

[30] M. Polverini, A. Cianfrani, S. Ren, and A. V. Vasilakos, "Thermal-aware scheduling of batch jobs in geographically distributed data centers," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 71–84, Jan. 2014.

[31] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schema," Google Inc., Mountain View, CA, USA, Tech. Rep., Nov. 2011, revised 2012.03.20. [Online]. Available: http://code.google.com/p/googleclusterdata/wiki/TraceVersion2

Neeraj Kumar (M'16–SM'17) received the Ph.D. degree in computer science and engineering from Shri Mata Vaishno Devi University, Katra (J&K), India.

He was a Postdoctoral Research Fellow with Coventry University, Coventry, U.K. He is currently an Associate Professor with the Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology (Deemed to be University), Patiala, India. He has published more than 250 technical research papers in leading journals and conferences from IEEE, Elsevier, Springer, Wiley, etc.

Dr. Kumar research findings are published in top cited journals such as IEEE TKDE, IEEE TIE, IEEE TDSC, IEEE TIFS, IEEE TCE, IEEE TII, IEEE TVT, IEEE ITS, IEEE NETWORKS, IEEE COMMUNICATIONS, IEEE WC, IEEE IoTJ, IEEE SJ, FGCS, JNCA, *Information Sciences, Computer Networks*, and the *Journal of Parallel and Distributed Computing* and ComCom. He has guided many research scholars leading to Ph.D. and M.E./M.Tech. His research is supported by funding from UGC, DST, CSIR, and TCS. He is an Associate Technical Editor of IEEE COMMUNICATION MAGAZINE. He is an Associate Editor of *IJCS*, Wiley, *JNCA*, Elsevier, and *Security & Communication*, Wiley.

Gagangeet Singh Aujla (S'15–M'18) received the B.Tech. and M.Tech. degrees from Punjab Technical University, Jalandhar, Punjab, India, in 2003 and 2013, respectively, and the Ph.D. degree from Thapar Institute of Engineering and Technology (deemed to be university), Patiala, Punjab, India, in 2018, all in computer science and engineering.

He is currently an Associate Professor with the Computer Science and Engineering Department, Chandigarh University, Mohali, India. Prior to this, he was a Research Associate with Indo-Austria Joint project funded by Indian and Austrian Governments. He was also a Project Fellow in a research project funded by HSCST, India. Before that, he was an Assistant Professor with the Department of Computer Science and Engineering, Chandigarh Engineering College, Mohali, India. He has many research contributions in the area of smart grid, cloud computing, edge computing, vehicular networks, software defined networks, security and cryptography.

Dr. Aujla's research findings are published in top cited journals such as the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE INTERNET OF THINGS JOURNAL, the IEEE COMMUNICATION MAGAZINE, the IEEE NETWORK MAGAZINE, the IEEE CE MAGAZINE, *Future Generation Computer Systems, Information Sciences, Computer Networks*, and the *Journal of Parallel and Distributed Computing* and top-tier conferences such as ACM MobiHoc, IEEE Globecom, IEEE ICC, IEEE WiMob, etc. He is the recipient of the 2018 IEEE TCSC outstanding dissertation award at Guangzhou China. He also received outstanding achievements and appreciation awards at Chandigarh Engineering College, India. He is a member of the IEEE, ACM and ISTE.

Sahil Garg (S'16–M'19) received the B.Tech. degree from Maharishi Markandeshwar University, Mullana, Ambala, India, in 2012, the M.Tech. degree from Punjab Technical University, Jalandhar, India, in 2014, and the Ph.D. degree from Thapar Institute of Engineering and Technology (deemed to be university), Patiala, Punjab, India, in 2018, all in computer science and engineering.

He is currently a Postdoctrol Research Fellow with the École de Technologie Supérieure, Université du Québec, Montréal, Canada. Prior to this, he was a Junior Research Fellow with the Department of Information and Electronics Technology, India. His research interests include machine learning, big data analytics, knowledge discovery, game theory, and vehicular ad-hoc networks. Some of his research articles are published in top-tier journals such as IEEE TM, IEEE Network, IEEE TII, IEEE ComMag, IEEE IoTJ, IEEE CEMag, FGCS and CAEE. He is member of the IEEE and ACM. He received the Best Paper Award in IEEE ICC 2018.

Kuljeet Kaur (S'15–M'19) received B.Tech. degree in computer science from Punjab Technical University, Kapurthala, India, in 2011, the M.E. degree in information security from Thapar University, Patiala, India, in 2015, and the Ph.D. degree in computer science and engineering from Thapar Institute of Engineering and Technology (deemed to be university), Patiala, Punjab, India, in 2018.

She is currently a Postdoctrol Research Fellow with the École de Technologie Supérieure, Université du Québec, Montréal, Canada. Prior to this, she has twao years industrial experience at Wipro Infotech, India. Her main research interests include radio-frequency identification, cloud/edge computing, smart grid, and vehicle-to-grid.

Dr. Kaur obtained a TCS fellowship and has been DST Inspire fellow in the past. She has authored or coauthored a number of research articles in top tier journals (IEEE WCM, IEEE TII, IEEE TM, IEEE TVT, IEEE TSG, IEEE SJ, IEEE CM, IEEE TPDS, IEEE PS, Springer PPNA, etc.) of repute. He received the Best Paper Award in IEEE ICC 2018.

Rajiv Ranjan (SM'16) received the Ph.D. degree from the Department of Computer Science and Software Engineering, University of Melbourne, Melbourne, Australia, in 2009.

He is a Reader (equivalent to nondistinguished Full Professor in the North American System) in computing science with Newcastle University, U.K. Before moving to Newcastle University, he was Julius Fellow (2013–2015), a Senior Research Scientist, and a Project Leader in the Digital Productivity and Services Flagship of Commonwealth Scientific and Industrial Research Organization (CSIRO C Australian Governments Premier Research Agency). Prior to that, he was a Senior Research Associate (Lecturer level B) with the School of Computer Science and Engineering, University of New South Wales.

Saurabh Kumar Garg (M'15) received the Ph.D. degree in computer science and engineering from the University of Melbourne, Melbourne, Australia, in 2010.

He is currently a Lecturer in the Department of Computing and Information Systems, University of Tasmania, Hobart, Tasmania. He has published more than 40 papers in highly cited journals and conferences with H-index 24. His doctoral thesis focused on devising novel and innovative market-oriented metascheduling mechanisms for distributed systems under conditions of concurrent and conflicting resource demand. He has gained about three years of experience in the Industrial Research while working at IBM Research Australia and India.