Optimal Decision Making for Big Data Processing at Edge-Cloud Environment: An SDN Perspective

Gagangeet Singh Aujla, *Student Member, IEEE*, Neeraj Kumar, *Senior Member, IEEE*, Albert Y. Zomaya, *Fellow, IEEE*, and Rajiv Ranjan, *Senior Member, IEEE*

***Abstract*—With the evolution of Internet and extensive usage of smart devices for computing and storage, cloud computing has become popular. It provides seamless services such as e-commerce, e-health, e-banking, etc., to the end users. These services are hosted on massive geodistributed data centers (DCs), which may be managed by different service providers. For faster response time, such a data explosion creates the need to expand DCs. So, to ease the load on DCs, some of the applications may be executed on the edge devices near to the proximity of the end users. However, such a multiedge-cloud environment involves huge data migrations across the underlying network infrastructure, which may generate long migration delay and cost. Hence, in this paper, an efficient workload slicing scheme is proposed for handling data-intensive applications in multiedge-cloud environment using software-defined networks (SDN). To handle the inter-DC migrations efficiently, an SDN-based control scheme is presented, which provides energy-aware network traffic flow scheduling. Finally, a multileader multifollower Stackelberg game is proposed to provide cost-effective inter-DC migrations. The efficacy of the proposed scheme is evaluated on Google workload traces using various parameters. The results obtained show the effectiveness of the proposed scheme.**

***Index Terms*—Cloud data centers, edge computing, energy efficiency, software-defined networks (SDNs), Stackelberg game.**

Nomenclature

n, m, f	Number of cloud DCs, edge DCs, job type.
X, Y	Delay-tolerant, delay-sensitive workloads.
F	Job with requirements: $a_{\text{type}}, b_{\text{req}}, c_{\text{req}}$.
a_{type}	Type of application.
$b_{\text{req}}, c_{\text{req}}$	Communication and computing requirement.
$\alpha(t)$	Arrival rate at time slot t.
$Q_i^f(t+1)$	Size of queue for type f jobs at time (t + 1).
$Q_i^f(t)$	Size of queue for type f jobs at time (t).
$Q_i^{pr}(t)$	Present size of queue at the ith DC at time (t).
$\lambda_i^f(t)$	Number of type f jobs routed at the ith DC.
S_i	Number of servers allocated.
μ_i	Processing speed of each server.
SLA_p^v	SLA violations of the pth server.
t_p^{thr}	Threshold utilization time.
t_p^{act}	Total active time of the server.
D_p^{mig}	Performance degradation due to migration.
$t_i^{\text{res}}, t_{\max}^{\text{res}}$	Response time and desirable response time.
D_i^{comm}, D_i	Communication and overall delays.
$D_i^{\text{mig}}, D_i^{\text{proc}}$	Migration and processing delays.
d_{net}	Delay incurred due to underlying networks.
D_i^{edge}	Delay incurred for handling jobs at edge devices.
v_i, a_i	Service and arrival rate at the ith edge device.
E_i, E_i^p	Energy consumption of the ith DC and the pth server of the ith DC.
E_i^c, E_i^o	Energy consumed for cooling and other activities of the ith DC.
E_i^{net}	Network energy consumption of the ith DC.
$E_{\text{sw}}^{\text{net}}, E_{\text{port}}^{\text{net}}$	Energy consumption of switches and ports.
E_{idl}^p	Energy consumption of the idle pth server.
$E_{\max}^p$	Maximum energy consumption of the pth server.
U_i^p	Utilization of the pth server of the ith DC.
$R^p(t)$	Resources consumed at time t at the kth server.
$R_{\max}^p$	Resources consumed at time t at the kth server.
x_i^{edge}	Job requests handled by edge devices.
E_i^{edge}	Energy consumed by the ith edge device.
a_i, b_i, c_i	Predefined parameters for edge devices.
C_i^f	Cost for handling type f job at the ith DC.
C_i^{tot}	Cost for handling type f job after migration.
C_i^{comp}	Cost related to computing at the ith DC.
C_i^{comm}	Cost related to at the ith DC.
C_i^{eng}	Cost related to energy at the ith DC.
C_i^{pen}	Cost related to SLA violations at the ith DC.
$C_{i\to k}^{\text{mig}}$	Migration cost.
P_i, M_i, S_i	Processor, memory, storage required.

Manuscript received April 2, 2017; revised July 18, 2017; accepted August 3, 2017. Date of publication August 11, 2017; date of current version February 1, 2018. Paper no. TII-17-0663. *(Corresponding author: Neeraj Kumar.)*

G. S. Aujla and N. Kumar are with the Department of Computer Science and Engineering, Thapar University, Patiala 147004, India (e-mail: gagi_aujla82@yahoo.com; neeraj.kumar@thapar.edu).

A. Y. Zomaya is with the School of Information Technologies, J12, University of Sydney, Sydney, NSW 2006, Australia (e-mail: albert.zomaya@sydney.edu.au).

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

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.2017.2738841

ρ, ρ_e	Price coefficient for resources and energy.
C_i^L, C_i^{IDC}	Local and Inter-DC communication cost.
C_i^{band}	Communication cost.
b_{net}	Bandwidth cost coefficient.
b_i^{comm}	Bandwidth requirement for communication.
E_i^f	Energy required to handle type *f* job.
$Y_i^f(t)$	Number of migrating jobs at time *t*.
C_p^{slav}	SLA violation cost per unit time.
T_p^{slav}	Duration of SLA violation for the pth processor.
U_i, U_k	Utility function of the ith and kth DC.
R_f	Revenue received for handling type f job.
R_{mig}	Revenue received for hosting f job migrated.
$\hat{\mho}_{ijk}^{\text{map}}, U_{ijk}$	Utility map function of *ijk* pair.
ηav_i	Delay of the network with new load.
τ_i^{av}	Throughput with new load.
$d_{\frac{i\to k}{j}}$	Distance from the ith to the kth DC at *j*.
z_{ijk}	Decision variable.

I. Introduction

CLOUD computing (CC) is one of the most powerful technologies to provide shared pool of resources such as servers, storage, and networks to the end users. Such resources are hosted at massive data centers (DCs) located geographically across the globe [1]. In recent years, data-intensive applications such as e-health, e-commerce, and e-banking have generated a huge volume of heterogeneous data, which vary with time [2]. To handle such massive data streams generated from these applications, the existing DC's infrastructure has been expanded in recent times. As per a recent survey [3], nearly 12 million servers are deployed in almost 3 million DCs in order to handle the online activities across the U.S. only. Moreover, with the advent of Internet of things (IoT), the big data generated from different applications have increased exponentially, which creates a need to design new effective solutions for improvement of the existing network infrastructure. So, such data explosion has created the demand for big data processing using large-scale geodistributed DCs.

Recent developments in the CC sector has provided a multicloud environment, which provides multiple cloud services through single heterogeneous computing architecture. Such a multicloud environment provides low latency, high data rate, and nondisruptive services with respect to big data processing to the end users [2]. In this direction, large cloud service providers (CSPs) such as Google, Microsoft, and Amazon have also stepped into big data processing using large-scale DCs located at various geographic locations [4]. To manage this huge amount of data, Google introduced the MapReduce framework supported by 13 DCs spread in eight countries across four continents [5]. Similarly, Netflix utilize Amazons EC2 infrastructure distributed across 11 regions over the globe to deploy their services [6]. Several architectures such as Spark and Storm have also been developed using the data-flow concept for improving big data processing [4].

For efficient processing of big data, a huge amount of data need to be transferred across geodistributed DCs using the underlying networks. However, such a movement of huge amount of data across DCs may incur large cost. For example, 706-GB/day inter-DC traffic is generated in BigBench, which involves a large amount of operational cost [7]. With continuous growth in size of big data generated by various sources, the need of migrating datasets across DCs for processing also increases. In this situation, the performance of underlying networks may become worst due to heavy traffic generated. Moreover, this may also generate high migration delay, network costs, and service-level agreement (SLA) violations to the CSPs. Several CSPs have deployed efficient data migration technologies in recent years. For example, Effingo has been deployed by Google to handle the large-scale data migration in its DCs [2].

Jayalath *et al.* [8] highlighted the impact of distributing computation for big data processing across large set of nodes. Similarly, Li *et al.* [6] presented an optimization problem by considering data movement and task placement to minimize the inter-DC traffic along with guaranteeing job completion with in a predefined time. Yu *et al.* [9] highlighted that the advent of IoT has leveraged the need of serving the requests of mobile devices in closer proximity of the users using geodistributed DCs. Yassine *et al.* [10] proposed a multirate bandwidth-on-demand scheme for inter-DC communications in order to offer reliable multimedia services. After analyzing the above-discussed proposals, it is evident that providing services closer to the end user can provide low-latency services for end users.

In this context, a latest technology that provides localized computing, storage, and processing services to end users is known as edge/fog computing. The ubiquitous nature of edge computing is critical for handling wide range of IoT-based real-time and latency-sensitive applications. Deng *et al.* [11] proposed a workload allocation scheme for a fog-cloud scenario. The authors put an emphasis on the fact that the cooperation between cloud and fog may help to achieve desired quality of service (QoS) and energy efficiency. Jalali *et al.* [12] presented a comprehensive analysis of CC and edge computing. The authors stressed on keeping the data closer to the end user in order to achieve lower latency. However, in the case of inefficient usage of the network resources, the energy consumption may increase. To resolve this issue, Borylo *et al.* [13] proposed a dynamic resource allocation scheme for an energy-aware cloud-fog interplay. The authors focused that optimal interplay between fog and cloud DCs using a software-defined network (SDN) can provide benefits such as energy efficiency and high QoS.

One of the major challenges for the underlying network is the inter-DC migration overhead due to high velocity of data movements across different DCs. In this direction, Lu *et al.* [2] proposed a dynamic anycast model using elastic optical inter-DC networks for data migration and backup. Gharbaoni *et al.* [1] presented an anycast-based approach to select a destination server for migrating virtual machines (VMs) by considering the actual load on inter-DC connections and VM data transfer requirements. Wang *et al.* [14] discussed the impact of inter-DC migration on performance of underlying DC networks. Gu *et al.* [4] highlighted that inter-DC traffic in big data processing constitutes large portion of DC traffic and thereby incurs a huge amount of operational cost. Chen *et al.* [7] presented a

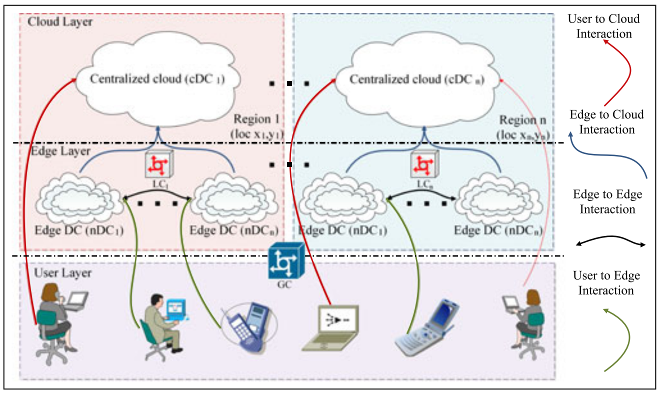


Fig. 1. System model.

workflow allocation graph, which considers the price diversity across geodistributed DCs to achieve cost minimization for big data processing. From the above proposals, it is evident that the performance of the underlying networks is an important parameter to achieve low-latency inter-DC migrations. So, to handle large data movements across different DCs, the SDN can be an attractive choice to manage the underlying networks resources. In this direction, Blenk *et al.* [15] presented SDN architecture for cost-effective and flexible control of communication networks. Xu *et al.* [16] proposed a bandwidth-aware energy-efficient routing algorithm using an SDN to improve network performance. Wang *et al.* [17] utilized an SDN to define the QoS and energy-aware flow path for network management.

A. Contribution

Based upon the above discussion, the major contributions of this work are given as follows.

1) A workload slicing scheme for handling data-intensive jobs in multiedge-cloud environment is presented.
2) An SDN-based controller is designed to provide an energy-aware flow-scheduling scheme with access of virtualized network resources.
3) A multileader multifollower Stackelberg game is formulated for providing optimal inter-DC migrations.

II. System Model

Fig. 1 shows the system model comprising of a multiedge-cloud environment having *n* cloud and *m* edge geodistributed DCs located in a region. The cloud DCs are large-scale infrastructure that consists of huge computing, storage, and network resources. However, the edge DCs consist of nano-DCs and edge devices (EDs). The proposed system model comprises of two controllers: 1) global controller (GC), and 2) local controller (LC). The GC is responsible for handling the workload classification and scheduling in multiedge-cloud environment and the LC handle the inter-DC migrations.

A. Workload Model

Consider a workload (*W*) comprising of *F* type of jobs to be processed in multiedge-cloud environment. A job is described as $F : (a_{\text{type}}, b_{\text{req}}, c_{\text{req}})$, where a_{type}, b_{req}, and c_{req} denote application type, communication, and computational requirements, respectively. At time *t*, type *f* jobs are modeled using poisson distribution with an arrival rate of $(\alpha(t))$. The type *f* jobs scheduled at the ith DC follow the queues dynamics [18] as follows:

$$Q_i^f(t+1) = \max\left[Q_i^f(t) - Q_i^{pr}(t)\right] + \lambda_i^f(t) \tag{1}$$

where $\lambda_i^f(t)$ is the number of type *f* jobs routed to the ith DC.

B. QoS Model

SLA is the most important requirement during handling incoming workload in multiedge-cloud environment. If the resources required to process the workload exceed the available capacity of resources with a DC, then a violation of SLA occurs. The SLA violations are computed on the basis of the time for which the pth server is experiencing threshold level of utilization (t_p^{thr}), total active time (t_p^{act}), and performance degradation (D_p^{mig}) due to migration. The SLA violations (SLA_p^v) of the pth server of ith DC is given as follows [19]:

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

Now, the performance degradation (D_p^{mig}) due to migration is defined similar to [19] given as follows:

$$D_p^{\text{mig}} = \frac{1}{W}\sum_{w=1}^{W}\frac{\varrho_w^{dg}}{\varrho_w^{cp}} \tag{3}$$

where ϱ_w^{dg} and ϱ_w^{cp} denote estimate of performance degradation due to migration and resources requested for migration.

Moreover, low delay and high response time are the most desired requirements of end users. In this context, the response time (t_i^{res}) for handling an incoming job is illustrated as follows:

$$t_i^{\text{res}} = \frac{1}{\mu_i \times S_i - Q_i^{pr}(t)} + \frac{1}{\mu_i} + D_i^{\text{comm}} \tag{4}$$

where S_i is the total number of servers allocated, μ_i is the processing speed of each server, and D_i^{comm} denotes the delay incurred for communication from source to the allocated DC.

The delay incurred for communication (D_i^{comm}) from source to the allocated DC is given as follows:

$$D_i^{\text{comm}} = d_{\text{net}}\lambda_i^f(t) \tag{5}$$

where d_{net} is delay incurred due to an underlying network.

The overall delay incurred in processing an incoming job request comprises of response time, migration (D_i^{mig}), and processing (D_i^{proc}) delays. So, the delay (D_i) is given as follows:

$$D_i = D_i^{\text{mig}} + D_i^{\text{proc}} + t_i^{\text{res}}. \tag{6}$$

In order to meet the SLA requirements, sometimes, workload is migrated from one DC to another that may incur additional delay. The delay incurred during inter-DC migration ($D_{i\to k}^{\text{mig}}$) from the ith DC to the kth DC is given as follows:

$$D_{i\to k}^{\text{mig}} = d_{\text{net}}\alpha(t). \tag{7}$$

Now, in case an edge DC or devices is handling the job, the delay incurred (D_i^{edge}) is defined using the M/M/1 queuing model and is given as follows:

$$D_i^{\text{edge}} = \frac{1}{v_i - a_i} \tag{8}$$

where v_i and a_i denote the service rate and the arrival rate of jobs, respectively.

C. Energy Model

The energy consumption of a DC comprises of energy consumed by processors (E_i^p), network resources (E_i^{net}), cooling (E_i^c), and other infrastructure (E_i^o). So, the energy consumption of the ith DC is given as follows:

$$E_i = \sum_p E_i^p + E_i^{\text{net}} + E_i^c + E_i^o. \tag{9}$$

Now, the energy consumption of a processor depends directly on the amount of utilization (U_i^p) and is given as follows:

$$E_i^p = E_{\text{idl}}^p + (E_{\max}^p - E_{\text{idl}}^p)\, U_i^p \tag{10}$$

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

The level of utilization of the pth server of the ith DC depends on the amount of resources consumed ($R^p(t)$) at time t and maximum capacity of processor ($R_{\max}^p$) and is given as follows:

$$U_i^p = \left(\frac{R^p(t)}{R_{\max}^p}\right) \times 100. \tag{11}$$

A major chunk of energy consumption of DCs depends on the network infrastructure. The network devices consume energy on the basis of fixed energy consumption ($E_{\text{sw}}^{\text{net}}$) and dynamic energy consumption ($E_{\text{port}}^{\text{net}}$). So, the energy consumption of network devices in the ith DC is given as follows:

$$E_i^{\text{net}} = E_{sw}^{\text{net}} + E_{\text{port}}^{\text{net}}. \tag{12}$$

The energy consumed by the network infrastructure in a DC depends upon the working time of the network devices

$$E_{\text{dc}}^n = \sum_{q \in S} E_q \times T_q + \sum_{r \in Pq} E_r^q \times T_r^q \tag{13}$$

where S and P_q are set of switches and ports in switch q; E_q, T_q, E_r^q, and T_r^q are the fixed power consumed by the qth switch, working time of the qth switch, dynamic power consumed by the rth port of the qth switch, and working time of the rth port of the qth switch.

Now, expanding (13) as per anticipated traffic, it becomes

$$E_{dc}^n = \sum_{q \in S} E_q \times \frac{\tau_q}{b_c \Theta_q |Pq|} + \sum_{r \in Pq} E_r^q \times \frac{\tau_r^q}{b_c \Theta_r^q} \tag{14}$$

where τ_q is the aggregate traffic traversing through switch q, τ_r^q is the aggregate traffic traversing through port *r* of switch q, Θ_q is the average occupancy ratio of switch q, and Θ_r^q is the average occupancy ratio of port *r* of switch q for the working time.

Now, if the EDs are handling the job requests (x_i^{edge}), then the energy consumed by the ith ED is given as follows:

$$E_i^{\text{edge}} = \left(a_i (x_i^{\text{edge}})^2 + b_i x_i^{\text{edge}} + c_i\right) \times t \tag{15}$$

where $a_m > 0$ and $b_m, c_m \geq 0$ are the predefined parameters.

D. Cost Model

The operational cost (C_i^f) for handling type *f* job at the ith DC comprises of different subcosts and is given as follows:

$$C_i^f = C_i^{\text{comp}} + C_i^{\text{comm}} + C_i^{\text{eng}} + C_i^{\text{pen}} \tag{16}$$

where C_i^{comp}, C_i^{comm}, C_i^{eng}, and C_i^{pen} are the costs incurred on computing resources, communication infrastructure, energy, and SLA violations, respectively.

In some cases, migration of job from the ith DC to the kth DC occurs. Hence, a migration cost ($C_{i \to k}^{\text{mig}}$) is also incurred. After considering this fact, the total cost (C_i^{tot}) incurred by a DC while handling *f* type of jobs is given as follows:

$$C_i^{\text{tot}} = C_i^f + C_{i \to k}^{\text{mig}}. \tag{17}$$

The cost on computing resources allocated to handle a job depends on processor (P_i), storage (S_i), and memory (M_i) required for a specific time (t_i). The cost for allocating various computing resources to the allocated job is given as follows:

$$C_i^{\text{comp}} = (\rho P_i + \rho M_i + \rho S_i) \times t_i \tag{18}$$

where ρ is the variable price coefficient for different resources.

The cost incurred for communication of data involves two types: 1) local communication (C_i^L) and 2) inter-DC communication (C_i^{IDC}), and is shown as follows:

$$C_i^{\text{comm}} = C_i^L + C_i^{\text{IDC}}. \tag{19}$$

Moreover, the above communication cost depends on the bandwidth requirements of the end user and is given as follows:

$$C_i^{\text{band}} = \sum_{j,k} b_{\text{net}} \lambda_i^f(t) b_{\text{req}} \tag{20}$$

where b_{net} is the bandwidth cost coefficient.

The cost of energy (E_i^f) required to execute type *f* jobs at the ith DC is given as follows:

$$C_i^{\text{eng}} = \rho_e E_i^f \tag{21}$$

where ρ_e is the price coefficient charged for per unit energy.

The cost for migrating type *f* jobs from the ith DC to the kth DC over flow path *j* is given as follows:

$$C_{i \to k}^{\text{mig}} = \sum_{j,k} b_{\text{net}} Y_i^f(t) b_{\text{req}} \tag{22}$$

where Y_i^f is the number of migrating type *f* jobs.

Sometimes, SLA violations may occur. Hence, the service provider has to bear a penalty (C_i^{pen}) as given below [19]:

$$C_i^{\text{pen}} = \sum_p [C_p^{\text{slav}}\, T_p^{\text{slav}}] \tag{23}$$

where C_p^{slav} is the cost of SLA violation per unit time and T_p^{slav} is the duration of violation for the pth processor of the ith DC.

III. Problem Formulation

In order to select an appropriate DC for migration in multiedge-cloud environment, the entities that play a vital role are source DC (*i*), flow path (*j*), and destination DC (*k*). Now, multiple choices exist for migrating data from the ith DC to the kth DC on the basis of *j* flow paths. The mapping ($\hat{\mho}^{\text{map}}_{i,j,k}$) of these entities is shown as follows:

$$\hat{\mho}^{\text{map}}_{i,j,k} = \sum_{i=1}^{n} \begin{bmatrix} 1,1,1 & 1,2,1 & . & . & 1,j,1 \\ 1,1,2 & 1,2,2 & . & . & 1,j,2 \\ . & . & . & . & . \\ . & . & . & . & . \\ 1,1,k & 1,2,k & . & . & 1,j,k \end{bmatrix}. \tag{24}$$

For this purpose, a combined utility is defined as follows:

$$U_{ijk} = \frac{b_{\text{req}} \times \eta_i^{av}}{(n+1) \times \tau_i^{\text{av}}} \times \frac{1}{d_{\frac{i \to k}{j}}} \tag{25}$$

where η_i^{av} and τ_i^{av} are the average anticipated throughput and delay of the network after including the new load. $d_{\frac{i \to k}{j}}$ is the distance from the ith to kth DC through *j*.

To select the optimal mapping from the above-discussed matrix, a decision variable ($z_{ijk}, \forall t$) is defined as follows:

$$z_{ijk} = \begin{cases} 1, & \text{for} \quad U_{ijk} > U^*_{ijk} \\ 0, & \text{for} \quad \text{otherwise} \end{cases} \tag{26}$$

where ijk^* is the set of pairs other than *ijk*.

Hence, the objective function is formulated using integer linear programming and is given as follows:

$$\max \left[\sum_{j=1}^{j_n} (\mho_{1j_11}) z_{1j_11} + \mho_{1j_22} z_{1j_22} + ... + \mho_{1j_nk} z_{1j_nk} \right] \tag{27}$$

subject to the following constraints:

$$z_{ijk} \in [0,1] \tag{28}$$

$$U_i(k) > U_i(k^*) \tag{29}$$

$$U_k(t) > U_k(t-1) \tag{30}$$

$$0 < \sum_f Q_i^{pr}(t) c_{\text{req}} \leq S_i \tag{31}$$

$$t_i^{\text{res}} \leq t_{\max}^{\text{res}} \tag{32}$$

$$C^{\text{mig}}_{i \to k} < C_i^{\text{pen}} \tag{33}$$

$$d_{(\frac{i \to k}{j})} < d_{(\frac{i \to k}{j})^*} \tag{34}$$

where $U_i(k)$ is the utility of the ith DC with respect to the kth DC, $U_i(k^*)$ is the utility of the ith DC with respect to DCs other than *k*, $U_k(t)$ and $U_k(t+1)$ are utilities of the kth DC at time *t* and t + 1, respectively, $t_{\max}^{\text{res}}$ is the maximum desirable response time, and $d_{(\frac{i \to k}{j})^*}$ denotes the distance between all pairs other than the ith to kth DC through flow path *j*.

IV. Proposed Scheme

The proposed scheme is divided into three phases. The algorithms for these phases are described as follows:

A. Workload Slicing Scheme for Multiedge-Cloud Environment

In this scheme, input workload (W) is sliced into two categories: delay-sensitive (X) and delay-tolerant (Y) workloads. Now, X is based on real-time applications that require maximum response. Moreover, Y is a workload with maximum completion time and requires high computing resources. But, it have to be completed before a predefined deadline. Now, X is high-priority workload, and it is scheduled before Y. The workload (Y) requires high computing resources and is routed directly to geodistributed cloud DCs. But, the real-time workload (X) is subdivided into two parts as shown below. One part of the workload (X^e) is scheduled to available edge DCs. However, there may be some workload (X^c), which requires high computing resources that may not be available at edge DCs. Such a workload slice is routed to cloud DCs. The architecture of the workload slicing scheme is shown in Fig. 2. Algorithm 1 shows the working of the proposed slicing scheme using prioritized preemptive round robin (PPRR) similar to [20] to schedule the jobs at DCs or EDs.

B. SDN-Based Controller

In the proposed SDN framework, the underlying network infrastructure is decoupled from the controller. Contrary to the traditional networks, all the forwarding devices (FDs) such as-routers, gateways, and switches in the SDN can flexibly adapt to new functionalities and network policies. The communication infrastructure in the SDN follows the open flow protocol [21], [22]. Fig. 3 shows the SDN architecture consisting of three decoupled planes: data, control, and application, which are described as follows:

1) Data Plane: The data plane consist of FDs that as per forwarding decisions taken by the SDN controller. Such decisions are configured into FDs using data-control plane interface. FDs contain a set of flow tables and group tables that are linked to each other by a pipeline [21]. The flow table follows the instruction set provided by the SDN controller. The instruction set consists of the matching rule, priority, action, and statistics. The working of data plane is described as follows.

Step 1: The source DCs that need to migrate the job to another DC send a request. The request of DC is received by a scheduler and queued for further processing.

Step 2: The scheduler matches the input requirements with rules prescribed by the SDN controller through an instruction set. The matching rule consists of flow id, source IP address, source MAC address, virtual LAN address, port number, and transport protocols [21]. On the basis of matching rules, an appropriate action is decided. The possible actions by FDs consist forwarding, modifying, discarding, replicating, etc.

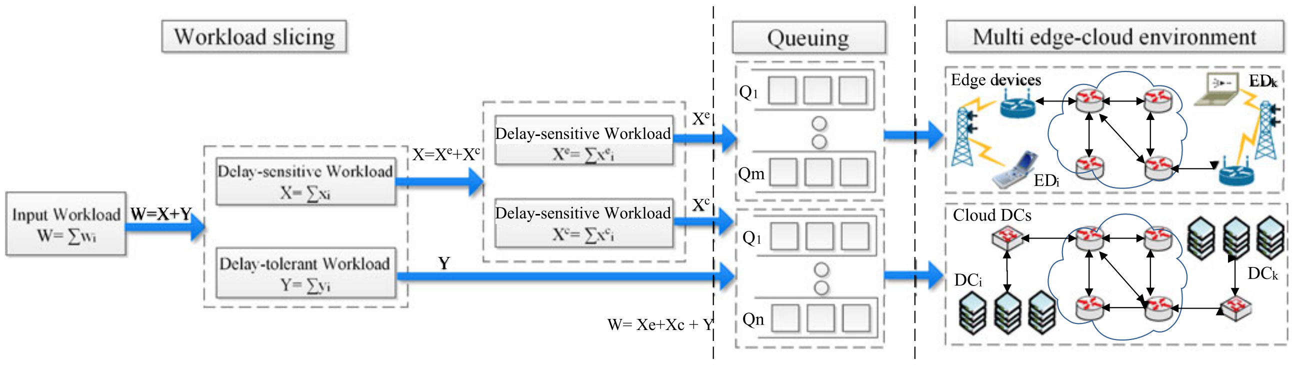
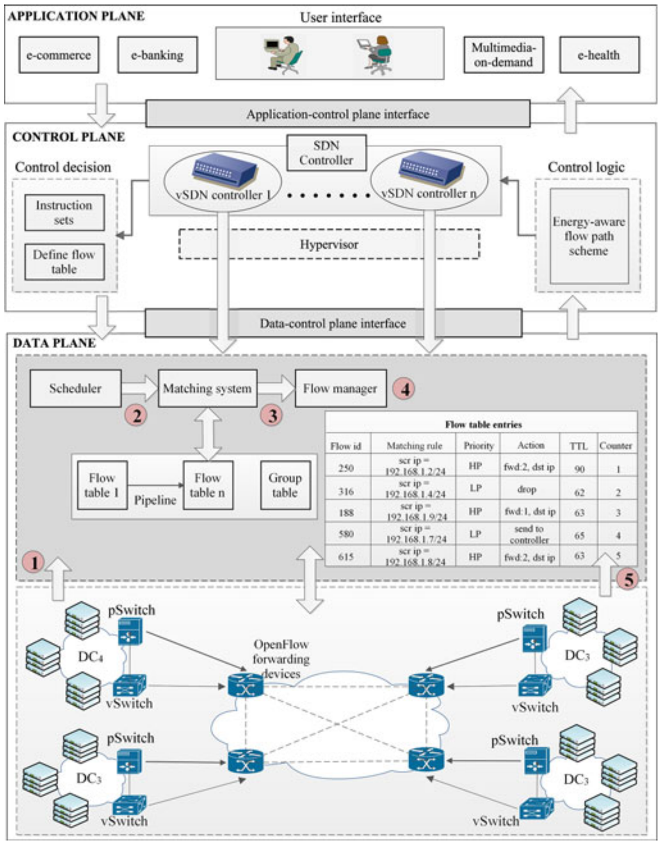

Workload slicing
Queuing
Multi edge-cloud environment
Input Workload W= ∑wᵢ
W=X+Y
Delay-sensitive Workload X= ∑xᵢ
Delay-tolerant Workload Y= ∑yᵢ
X=Xᵉ+Xᶜ
Delay-sensitive Workload Xᵉ= ∑xᵉᵢ
Delay-sensitive Workload Xᶜ= ∑xᶜᵢ
Xᵉ
Xᶜ
Y
W= Xe+Xc + Y
Q₁
Qm
Q₁
Qn
Edge devices
EDₖ
EDᵢ
Cloud DCs
DCᵢ
DCₖ

Fig. 2. Workload slicing scheme.

```
Algorithm 1: Workload slicing and scheduling algorithm.
  Input: Workload $W$
  Output: Cloud DC or ED
  1:  Slice workload $W$ into $X$ and $Y$
  2:  if $W = Y$ then
  3:      Check for type of jobs
  4:      Compute $F : (a_{\text{type}}, b_{\text{req}}, c_{\text{req}})$
  5:      if $F : (a_{\text{type}}, b_{\text{req}}, c_{\text{req}})$ are available with $DC_i$ then
  6:          Add workload $\rightarrow Q_N : (Q_1, Q_2, ...., Q_n)$
  7:          Select flow path using Algorithm II
  8:          Schedule job $F \rightarrow$ DC$_i$ $\rightarrow$ PPRR
  9:      else
  10:         Schedule job $\rightarrow$ DC$_{i^*}$ $\rightarrow$ PPRR     $\triangleright\ i \notin i^*$
  11:     end if
  12: else
  13:     Check for available EDs
  14:     Map $X$ with available EDs
  15:     if Required resources are available with $ED_i$ then
  16:         Add workload $\rightarrow Q_M : (Q_1, Q_2, ...., Q_m)$
  17:         Select flow path using Algorithm II
  18:         Schedule job $\rightarrow$ ED$_i$ $\rightarrow$ PPRR
  19:     else
  20:         Add workload $\rightarrow Q_N : (Q_1, Q_2, ...., Q_n)$
  21:         Schedule job $\rightarrow$ cloud DCs
  22:     end if
  23: end if
```



Flow table entries

Flow id	Matching rule	Priority	Action	TTL	Counter
250	scr ip = 192.168.1.2/24	HP	fwd:2, dst ip	90	1
316	scr ip = 192.168.1.4/24	LP	drop	62	2
188	scr ip = 192.168.1.9/24	HP	fwd:1, dst ip	63	3
580	scr ip = 192.168.1.7/24	LP	send to controller	65	4
615	scr ip = 192.168.1.8/24	HP	fwd:2, dst ip	63	5

Fig. 3. Architecture of the SDN-based control scheme.

Step 3: Once an action is decided, the request is forwarded to the flow manager. This is followed by the selection of an appropriate flow table to complete the action.

Step 4: Once the appropriate flow table is selected, the packets are migrated to the destination DC using it.

Step 5: This step involves feedback to verify the reliability of the flow path. This is done by using statistics that contains a counter for reporting to the controller.

2) Control Plane: The control plane is the decision making plane that works on the basis of a control logic. Using the control logic, the SDN controller forwards the programming and logic instructions into an instruction set. The SDN controller is a centralized entity that handles the network traffic dynamically. But, with an increase in the network traffic, the physical controller gets overloaded. So, the efficiency of the physical controller is degraded with respect to latency, bandwidth, and resilience. One of the major issues that occur in a large-scale network is the resilience, i.e., in case the primary physical controller gets fail, then the entire network fails.

Hence, in order to resolve these issues, the concept of virtual SDN (vSDN) network is used. The vSDN network allows the network slicing of a large physical network into multiple virtual networks. In this concept, the controller creates a virtual network infrastructure, which can be utilized to schedule the flow when physical resources are exhausted. However, the virtual network resources are a slice of physical resources only. Using the network hypervisor, the instances of the physical network are created as multiple virtual networks. The network hypervisor is installed at the network operating system (NOS), which acts as an intermediate layer between the vSDN network and the underlying physical SDN network, hence allowing to exploit parallelism by running multiple NOSs on the vSDN network. The vSDN provides flexibility to the software programmer to easily program and run their vSDN network via openflow protocols and interfaces. The vSDN network consists of a set of multiple virtual controllers and virtual switches of a single physical SDN network connected via a hypervisors. So, by extending the physical network into multiple vSDN networks, manifold benefits such as high resource utilization, load balancing, remote programming, cost saving, and low overhead are achieved. In order to handle multiple jobs, the controller adjusts the load of the incoming jobs as per available resources using a load balancing rate (Υ) as follows [21]:

$$\Upsilon = \frac{1/j \times \sum_0^i L_i}{L_{\max}} \tag{35}$$

where $L_{\max}$ is the maximum load a controller/switch can bear.

The load balancing rate lies between 0 and 1. If the value of Υ is close to 1, then it means that the load is evenly distributed. However, if the value of Υ is low, then it means that the load is not evenly distributed and the controller needs to migrate the load using the offload manager.

The incoming traffic flow (*f*) is categorized with respect to its status: 1) active (f^a), 2) queued (f^q), and 3) suspended (f^s) flows. Now, *f* is queued in the appropriate queue. The status of flow is active only if a valid flow path (*j*) exists. The traffic flow that is to be scheduled is added to a specific queue. A flow is said to be active only if a valid path without any other flows exists for it. As soon as the flow reaches the top of the queue, it becomes active. However, a flow is said to be suspended if no valid path exists for it. In this case, the controller reconfigures the flow tables in order to provide a valid flow path for the suspended flow. Once a valid flow path is available, then it is added to an appropriate queue for scheduling.

An energy-aware flow-scheduling algorithm is presented to provide a control logic to the SDN controller for taking decisions related to flow scheduling. In order to make the flow-scheduling process energy efficient, ports on an inactive link are put into the sleep mode. Moreover, when all ports of a specific switch are in the sleep mode, then the concerned switch is also put into the sleep mode. This action is performed to minimize the energy consumption of unused ports and switches [16]. In order to synchronize the shifting of switch into the sleep mode, a decision variable ($\Psi_{\text{syn}}, \forall t$) is defined as follows:

$$\Psi_{\text{syn}} = \begin{cases} 1, & \text{for active} \\ 0, & \text{for idle.} \end{cases} \tag{36}$$

If ($\Psi_{\text{syn}} = 0$), then the switch shifts to the sleep mode. For this purpose, a threshold time (t_{thr}) is considered. The value of Ψ_{syn} becomes 0 only if the switch is idle for threshold time (t_{thr}). The switch shifts back to the active mode if the value of Ψ_{syn} is 1.

A job (*f*) having size (s_f) with a deadline time (t_f^r), release time (t_f^r), and guaranteed flow rate (r_f) is to route from DC$_i$ to DC$_k$. The guaranteed flow rate (r_f) is given as follows:

$$r_f = \frac{s_p}{T_p^d - T_p^r}. \tag{37}$$

The flow path (*j*) on which the incoming flow (*f*) would be routed should be selected in such a way that the utilization of network resources are maintained in an optimal manner. The proposed algorithm must adhere to minimal energy consumption and guaranteed data rate while selecting a flow path (*j*) for flow (*f*). The proposed flow-scheduling Algorithm 2 is as follows.

Algorithm 2: Energy-aware flow-scheduling algorithm.

```
Input: $f$, $s_f$, $t_f^d$, $t_f^r$, $G$, $f^a$, $f^q$, and $f^s$
Output: $path\ p$, $r_f$
1:  Calculate guaranteed flow rate ($r_f$) using Eq. (37)
2:  $j \leftarrow FindPath$(G, $f^a$, $f^q$, $f^s$, $f$, $g_f$)
3:  if valid path exists then
4:      if physical path $j$ exists then
5:          Schedule $f_p$ over $p$
6:          for Each flow path J do
7:              Divide J into flow sets $f_{\text{set}}$ with no shared links
8:              for $f_{\text{set}} \in$ J do
9:                  Calculate $t_{\text{act}} = activetime(f_{\text{set}})$
10:                 Compute energy consumption using Eq. 14
11:                 if ($t_{\text{act}}$ is minimum) then
12:                     $f^q \leftarrow f^q + f$
13:                     Schedule $f$
14:                 end if
15:             end for
16:         end for
17:     else
18:         Check for virtual path $j^v$
19:         if ($j^v$ exists) then
20:             Schedule $f$
21:         end if
22:     end if
23: else
24:     Suspend $f_p$ till a valid path is available
25:     $f^s \leftarrow f^s + f$
26:     Report to controller
27:     Controller rebuilds flow table for valid $j$
28:     Repeat steps 1–16
29: end if
30: if flow $f$ finishes then
31:     Update $f^a \leftarrow f^a - f$
32:     Move next flow in queue to the top
33: else
34:     Repeat steps 1–16
35: end if
```

In the proposed algorithm, the guaranteed flow rate (g_p) is computed for the flow (f) (line 1). Now, valid paths are searched in the flow table with respect to the network topology (G, f^a, f^q, f^s, f), and g_f (line 2). If a valid physical paths exists, then a link that consumes minimal energy is scheduled for *f*. To achieve this, each available flow path *J* is divided into a set of

TABLE I
CONDITIONS FOR INTER-DC MIGRATION

Case No.	Decision	Bandwidth	Computing resources
Case 1	True	✓	✓
Case 2	True (*)	x	✓
Case 3	False	✓	x
Case 4	False	x	x

(*) True only if virtual network resources are available.

flows (f_{set}). After this, the active time (t^{act}) is computed for each element of f_{set}. In the next step, the energy consumed by each element of the flow set is calculated. Now, the incoming flow is scheduled to flow element with least active time and energy consumption. At this instant, the flow is on the top of queue and its status is active (lines 3–15). However, if no physical flow path exists, then virtual flow path (j^v) is checked. If j^v exists, then the f is scheduled over it (lines 16–20). But, there may be a case when no valid flow path exists, then in such a case, the incoming flow is suspended and the added to appropriate queue. The issue is reported to the controller, which rebuilds a valid flow path and the incoming flow is scheduled again (lines 21–26). After scheduling the flow, it is removed from the queue of active flows and next flow in the queue is shifted to active status (lines 27–33).

3) Application Plane: The purpose of application plane is to interact with various end-user applications and provide feedback to the controller through an application-control plane interface. Various end-user applications such as e-commerce, e-banking, multimedia-on-demand, etc., reside in this plane.

C. Stackelberg Game for Inter-DC Migration

In the proposed scheme, inter-DC migration is valid for three cases in edge-cloud environment such as: cloud to cloud, edge to cloud, and edge to edge. In order to participate in the migration, the conditions shown in Table I may exist.

1) Stackelberg Game: The Stackelberg game is a strategic game in economics and is popular as a special case of noncooperative games. It is two-period hierarchical game, in which the players are classified as a leader and followers [20]. Both the players in the game compete for the quantity, and the leader is sometimes called as the market leader. This is said because the leader avails the benefit of initiating the game. By doing so, the leader can enforce his moves on followers. But, the leader must be aware *ex ante* that the follower observes its actions. Generally, the leader has the power of commitment to its actions. On the other hand, the leader must know that the Stackelberg follower has no means of commitment to any of its actions. So, the leader's best response is to play follower's action [20]. Hence, in this way, both leaders and followers reach an equilibrium state in order to maximize their payoffs. The Stackelberg game has manifold advantages such as:

1) optimal choice in a distributed environment;
2) handles the economical aspects;
3) sequential movement of player;
4) competitive behavior.

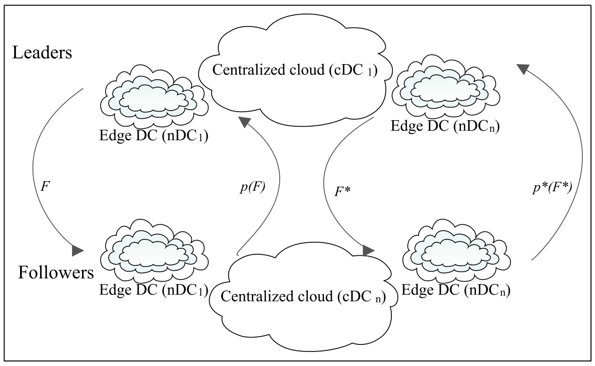


Fig. 4. Stackelberg game model.

2) Why Stackelberg Game?: In this work, the Stackelberg is the most suitable choice for handling various aspects related to inter-DC migration. A lot of similarities exist between the addressed problem and the Stackelberg game. In inter-DC migration, two players (source and destination DC/EDs) play their moves to reach an optimal solution. The source DC/EDs act as leaders and announce their resource requirements to destination DCs/EDs who act as followers. The game proceeds in a distributed edge-cloud environment, where DCs or EDs are geolocated. Moreover, the equilibrium between both the players is dependent on the economical factors. Both the players act in a sequential manner to compete with other DCs/EDs. Hence, with so many similarities, the Stackelberg game is most suitable for handling the issue of inter-DC migrations.

3) Game Model: In order to handle inter-DC migrations, a multileader multifollower Stackelberg game is presented. The proposed game model comprises of the following entities.

a) *Players: i* source DCs/ED (multiple leaders) and *k* destination DCs/ED (multiple followers).
b) *Strategy/action:* For the leaders, the strategy for selecting a host for handling $W : (a_{\text{type}}, b_{\text{req}}, c_{\text{req}})$ is $S_l = (f_1, f_2 \ldots, F_i)$, where $i \in I$. For followers, the strategy to finalize the price (P_k^{mig}) for hosting the migrating jobs is $S_f = (p_1, p_2 \ldots, p_k)$, where $k \in K$.
c) *Payoff:* The players finalize their decisions with respect to the payoffs they receive. For this purpose, different utility functions of leaders and followers are defined. In these utility functions, the terms price, cost, and revenue are used. Price, cost, and revenue represents the amount charged to sell a product, the amount incurred to manufacture of that product, and the amount that a producer receives on selling its product, respectively.
The utility function for the ith DC that requires to migrate data or job to the kth DC is given as follows:

$$U_i = \left[\sum_{f=1}^{F} R_f\right] - \sum_{f=1}^{F} [C_i^{\text{tot}}]. \tag{38}$$

The utility function of the kth DC selected to handle the migrated data from the ith DC is given as follows:

$$U_k = \left[\sum_{i} R_{\text{mig}}\right] - \sum_{j=1}^{J} [C_i^f]. \tag{39}$$

Algorithm 3: Stackelberg game for inter-DC migration.

```
Input: DC$_i$, $b_{\text{req}}$, $c_{\text{req}}$
Output: flowpath $j$, $ijk$ pair, $DC_k$
1: for ($i = 1; i \leq n; i$++) do
2:    $F : (a_{\text{type}}, b_{\text{req}}, c_{\text{req}}) \to DC_i$ ▷ Leader move
3:    for ($k = 1; k \leq n; k$++) do
4:       Check ($c_{\text{req}}$) ▷ Follower move
5:       if $C_{\text{req}}$ is available then
6:          Compute utility $U_k$
7:          if $U_k(t) > U_k(t-1)$ then
8:             Accept migration and announce price
9:          else
10:            Reject migration
11:         end if
12:      else
13:         Reject migration
14:      end if
15:   end for
16:   for ($k = 1; k \leq n; k$++) do ▷ Leader move
17:      Compute $U_i(k)$
18:      if $U_i(k) > U_i(k^*)$ then
19:         Add DC$_k$ in the queue above DC$_{k+1}$
20:      end if
21:      Select flow path j using Algorithm II
22:      Map all available ijk pairs
23:      Compute $U_{ijk}$
24:      if $U_{ijk} > U^*_{ijk}$) then
25:         Set decision variable $z_{ijk} == 1$
26:         Select ijk pair and send consent to the $k$th DC
27:         if the $k$th DC conforms then ▷ Follower move
28:            Migrate workload
29:         else
30:            Select next pair and repeat step
31:         end if
32:      end if
33:   end for
34: end for
```

4) Proposed Stackelberg-Game-Based Algorithm: The working of the Stackelberg game model is shown in Fig. 4. Using this model, a multileader multifollower Stackelberg game is formulated for selecting optimal destination DC for migration. In this regard, Algorithm 3 is designed to show the working of the proposed Stackelberg game.

In this algorithm, multileaders (i DCs) initiate the game by requesting all the available DCs for migration of job (line 1). Now, for all available followers (n DCs or m EDs), check for computing resources required. If the computing resources are available, then compute utility (U_k). If U_k at time t is more than the U_k at previous time slot, then accept the migration request and announce the price. Otherwise, the request is rejected by follower DCs (lines 2–15). In the next move, the leader (DC_i) computes its utility (U_i) for each of the k DCs that have accepted the migration request (lines 16 and 17). If the utility ($U_i(k)$) of DC$_i$ with respect to the kth is more than ($U_i(k^*)$ of each of the DC other than the kth DC, then add DC$_k$ in the queue above DC$_{k+1}$ (lines 18–20). Now, select flow path j for k DCs using Algorithm 2 (line 21). Now, map all *ijk* pairs. Compute utility (U_{ijk}) for all *ijk* pairs (lines 22 and 23). If ($U_{ijk} > U^*_{ijk}$)), then set decision variable (z_{ijk}) to 1. Otherwise, set the value of decision variable next available pair to 1 (lines 24 and 25). Now, select the *ijk* pair and send consent to the kth DC. Once the kth DC confirms the deal, then migrate workload. Otherwise, select the next pair (lines 26–34).

V. Results and Discussion

The proposed scheme is evaluated using a workload trace of 1000 jobs released by Google [23] and simulated using three scenarios: only cloud DCs, EDs, and the proposed edge-cloud interplay. The incoming workload requires some amount of resources such as CPU, memory, and storage. The resources required to serve the incoming job requests are shown in Fig. 5(a). Initially, the workload is classified into two categories: delay-sensitive and delay-tolerant jobs. The level of priority for various jobs on the basis of delay sensitivity is shown in Fig. 5(b). Now, the workload is scheduled to cloud DCs and EDs on the basis of the classification. The delay-sensitive jobs are provisioned using EDs and delay-tolerant jobs are handled by cloud DCs. However, some of the delay-sensitive jobs may also require high computing resources that are not available with EDs. Such jobs are provisioned using cloud DCs. The slicing of jobs between cloud DCs and EDs is shown in Fig. 5(c).

Some amount of energy is utilized to handle the jobs allocated to edge-cloud environment. The energy consumed by cloud DCs and EDs to serve the sliced jobs is shown in Fig. 5(d). The multiedge-cloud environment plays an important impact on the energy consumption of DCs. Fig. 5(e) shows the comparison of energy consumed by the proposed multiedge-cloud environment with other two scenarios. The energy consumed by multiedge-cloud DCs is lower as compared to the scenario when only cloud DC or EDs are used. Moreover, the proposed multiedge-cloud environment proves to be a better platform in terms of SLA violations also. Fig. 5(f) shows the SLA violations incurred for serving the incoming jobs. The SLA violations witnessed for the proposed environment are negligible as compared to other scenarios. The major reason for better performance of the edge-cloud environment is that the workload is sliced and scheduled to the host that is best suited to provide the required resources and QoS. In case of only DCs or EDs scenarios, there is not other option available to schedule the workload. There is either cloud DCs or EDs to handle incoming workload. But, in the edge-cloud environment, the workload is classified among cloud DCs and EDs, thereby reducing the load on resources. So, the energy consumption reduces as the loads on the resources are reduced. Moreover, with multiple options available for handling workloads, the SLAs are easily met.

The proposed scheme uses SDNs as underlying DC networks. The use of the proposed energy-aware flow-scheduling scheme for SDNs reduces the energy consumption with respect to underlying networks. Fig. 5(g) shows that the energy consumed by the proposed flow-scheduling scheme for SDNs consumed lesser energy as compared to traditional networks. Now, when the required computing resources are not available with the hosting DC or EDs, then the jobs are migrated to another DC or ED

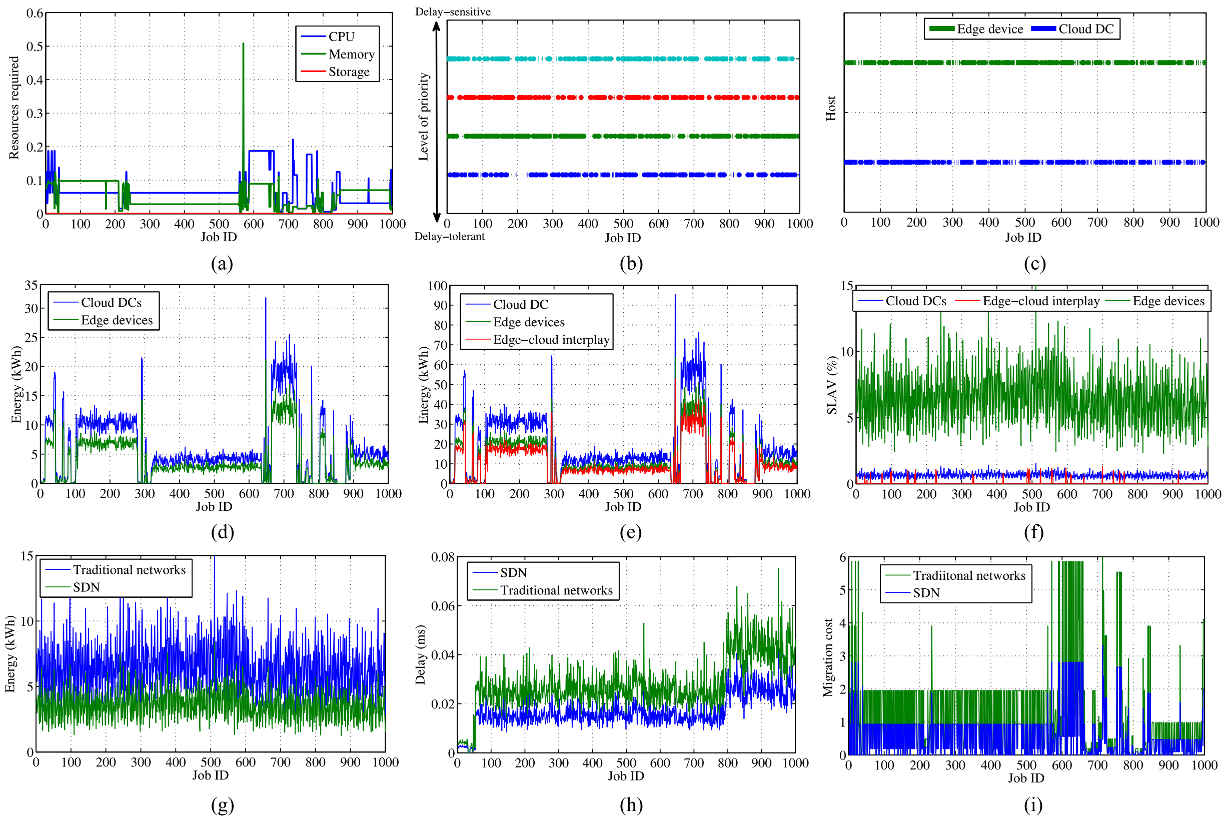


Fig. 5. Results obtained for single time slot. (a) Resources required for handling incoming jobs. (b) Priority of each job. (c) Workload slicing at edge and cloud DCs. (d) Energy consumed for handling each job. (e) Comparison of energy consumption. (f) SLA violations. (g) Network energy consumption. (h) Migration delay. (i) Migration cost.

so as to meet SLA. In that case, additional delay and cost are involved due to migration. However, an appropriate selection of destination DC or ED that can serve the migrated job is an important task. The proposed multileader multifollower Stackelberg game selects the appropriate DC or ED, where the job could be migrated with profit to both source and destination DCs or EDs, apart from this, the underlying networks and the flow path that serves the backbone of such a migration. The dynamic flow path selection not only avoid additional delay, but also reduce migration cost. In this context, Fig. 5(h) shows the migration delay incurred while migrating the jobs from source DC or ED to destination DC or ED. The results depict a lower delay for SDNs as compared to traditional networks. Moreover, the use of SDNs has a strong impact on the reduction of migration cost due to its dynamic and flexible nature. Fig. 5(i) shows that the migration cost for SDNs is much less than the cost involved when traditional networks are used.

A. Case Study

For inter-DC migrations, a Stackelberg game is formulated to select the destination host. The DCs/EDs are selected on the basis of a combined utility (U_{ijk}). But, the individual utilities of leaders and followers must show an increase with respect to previous instant. For deep analysis, a game with one leader and nine followers (four cloud DCs and five EDs) is formulated. The value of decision variable (z_{ijk}) is shown in Table II.

TABLE II
SELECTION OF DESTINATION HOST FOR JOB MIGRATION

Host	DC_1	DC_2	DC_3	DC_4	ED_1	ED_2	ED_3	ED_4	ED_5
DC_{l1}	1	0	0	0	0	0	0	0	0
DC_{l2}	0	0	0	0	0	0	1	0	0
DC_{l3}	0	1	0	0	0	0	0	0	0
DC_{l4}	0	0	0	1	0	0	0	0	0

Note: Each cell shows value of decision variable (z_{ijk}).

The destination DC or ED is selected if the value of z_{ijk} is equal to 1. The value of z_{ijk} is equal to 1 only if the combined utility (U_{ijk}) is maximum and the utilities of leader and follower increase with respect to previous instance. Now, for the first leader (DC_{l1}), the value of z_{ijk} is equal to 1 for the follower (DC_1), as it required high computing resources which were not available with any other follower. Now, let us consider a case when resources required are small, but low latency is required. In this case, for the leader (DC_{l2}), the value of z_{ijk} is equal to 1 for follower (ED_3), as it serves the resource as well as latency requirements of the leader. Similarly, DC_{l3} and DC_{l3} select different destination DCs/EDs as per their resource and SLA requirements. So, the proposed game acts as an optimal decision maker for destination host and flow path selection.

B. Evaluation for a 12-h Scenario

After analyzing the proposed scheme for single time slot, it is evaluated for a longer time period (12 h). The results obtained

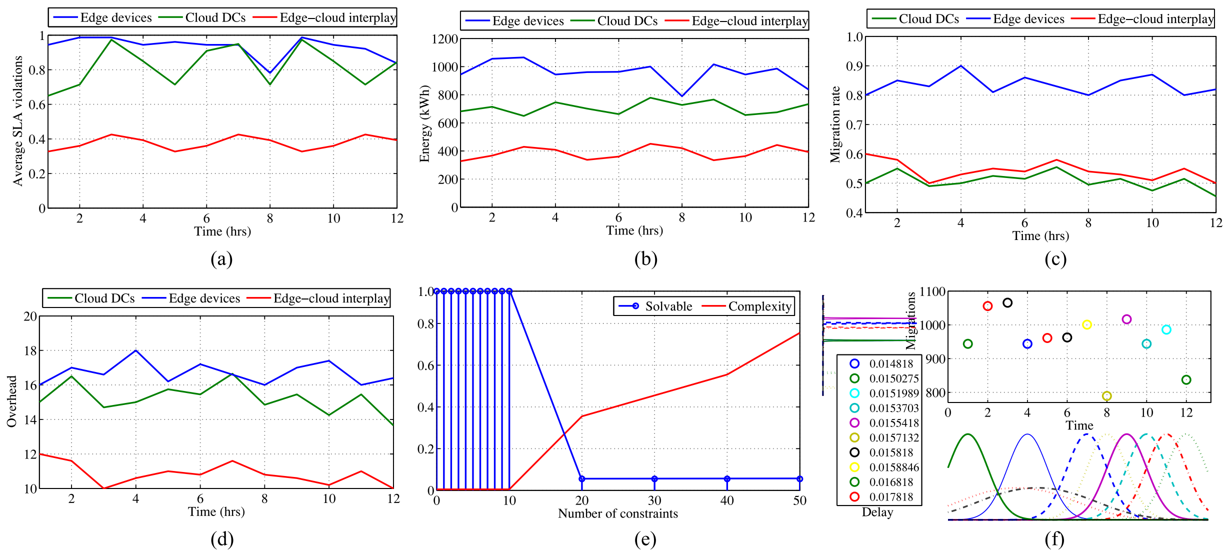


Fig. 6. Results obtained for the 12-h scenario. (a) Average SLA violations. (b) Energy consumed. (c) Migration rate. (d) Average overhead. (e) Complexity analysis. (f) Edge-to-edge analysis.

clearly show that the edge-cloud environment has a clear lead over other two cases. Fig. 6(a) shows that the average SLA violations for the proposed environment are lower than the other two cases. Moreover, the proposed scheme consumes lesser energy as compared to other cases as shown in Fig. 6(b). Finally, in Fig. 6(c), the migration rate for the three cases is compared. The results show that the migration rate in EDs is more than the other two cases due to limitation of resources in EDs. The cloud DCs show the lowest migration rate, but the proposed environment is almost equal to it. Finally, Fig. 6(d) shows the average overhead for all three cases. The result shows that the edge-cloud environment ends up in the lowest overhead. Hence, the results obtained indicate that the proposed edge-cloud environment is better than other cases in terms of energy consumption, SLA violations, migration rate, and overhead.

C. Complexity Analysis

Now, the complexity analysis of the proposed integer linear programming (ILP) problem is performed. Generally, the ILP problems are NP-hard, but this is not true for every problem. The present problem is a simpler case and can be easily solved with respect to present set of constraints. Fig. 6(e) shows the complexity and solvability variation of the proposed problem with respect to the number of constraints. It clearly shows that the proposed problem is solvable till ten constraints, but after that, its complexity increases.

D. Evaluation of Edge-to-Edge Migration

Finally, an analysis of edge-to-edge migrations is performed. Fig. 6(c) shows that the EDs being resource-limited show high migration rate. Also, the edge-to-edge migrations are analyzed with respect to migrations and delay. Fig. 6(f) shows the comparison of delay and migrations in the edge-to-edge environment. It is evident that the EDs act as a best compliment to cloud DCs. But, if considered individually, they incur higher energy consumption, migration rate, and SLA violations. However, the delay is lower for EDs as compared to cloud DCs.

VI. Conclusion

In this paper, a workload slicing scheme has been designed for handling big data applications in a multiedge-cloud environment. In this environment, the incoming job requests are sliced on the basis of priority and scheduled among EDs and cloud DCs. Moreover, an SDN controller is proposed for an energy-aware flow-scheduling scheme using virtualized networks. Finally, a multileader multifollower Stackelberg game is formulated to select an optimal DC or ED to host the migrated jobs. The proposed scheme has been evaluated on the basis of various parameters such as energy, delay, SLA violations, migration rate, and cost. The results obtained show that the proposed scheme minimizes the energy consumption of overall multiedge-cloud environment and underlying networks. Moreover, a reduced delay and cost for inter-DC migration is also achieved.

References

[1] M. Gharbaoui, B. Martini, and P. Castoldi, "Anycast-based optimizations for inter-data-center interconnections [invited]," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 4, no. 11, pp. 168–178, Nov. 2012.
[2] P. Lu, L. Zhang, X. Liu, J. Yao, and Z. Zhu, "Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks," *IEEE Netw.*, vol. 29, no. 5, pp. 36–42, Sep. 2015.
[3] 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* 14–08, 2014.
[4] L. Gu, D. Zeng, S. Guo, Y. Xiang, and J. Hu, "A general communication cost optimization framework for big data stream processing in geo-distributed data centers," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 19–29, Jan. 2016.
[5] L. Gu, D. Zeng, P. Li, and S. Guo, "Cost minimization for big data processing in geo-distributed data centers," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 3, pp. 314–323, Sep. 2014.
[6] P. Li *et al.*, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1785–1796, Jun. 2017, doi: 10.1109/TPDS.2016.2626285.

[7] W. Chen, I. Paik, and Z. Li, "Cost-aware streaming workflow allocation on geo-distributed data centers," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 256–271, Feb. 2017.

[8] C. Jayalath, J. Stephen, and P. Eugster, "From the cloud to the atmosphere: Running mapreduce across data centers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 74–87, Jan. 2014.

[9] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Trans. Emerg. Topics Comput.*, 2015, to be published, doi: 10.1109/TETC.2015.2479093.

[10] A. Yassine, A. A. N. Shirehjini, and S. Shirmohammadi, "Bandwidth on-demand for multimedia big data transfer across geo-distributed cloud data centers," *IEEE Trans. Cloud Comput.*, 2016, to be published, doi: 10.1109/TCC.2016.2617369.

[11] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016.

[12] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1728–1739, May 2016.

[13] P. Borylo, A. Lason, J. Rzasa, A. Szymanski, and A. Jajszczyk, "Energy-aware fog and cloud interplay supported by wide area software defined networking," in *Proc. IEEE Int. Conf. Commun.*, May 2016, pp. 1–7.

[14] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos, "A survey on data center networking for cloud computing," *Comput. Netw.*, vol. 91, pp. 528–547, 2015.

[15] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 655–685, First Quarter 2016.

[16] G. Xu, B. Dai, B. Huang, J. Yang, and S. Wen, "Bandwidth-aware energy efficient flow scheduling with SDN in data center networks," *Future Gener. Comput. Syst.*, vol. 68, pp. 163–174, 2017.

[17] S.-H. Wang, P. P. W. Huang, C. H. P. Wen, and L. C. Wang, "EQVMP: Energy-efficient and QoS-aware virtual machine placement for software defined datacenter networks," in *Proc. Int. Conf. Inf. Netw.*, Feb. 2014, pp. 220–225.

[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] 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.: Pract. Experience*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

[20] G. S. Aujla, M. Singh, N. Kumar, and A. Y. Zomaya, "Stackelberg game for energy-aware resource allocation to sustain data centers using RES," *IEEE Trans. Cloud Comput.*, 2017, to be published, doi: 10.1109/TCC.2017.2715817.

[21] G. S. Aujla, R. Chaudhary, N. Kumar, J. J. Rodriques, and A. Vinel, "Data offloading in 5G-enabled software-defined vehicular networks: A Stackelberg game-based approach," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 100–108, Aug. 2017.

[22] D. Chen, L. Wang, A.Y. Zomaya, M. Dou, J. Chen, Z. Deng, and S. Hariri, "Parallel Simulation of Complex Evacuation Scenarios with Adaptive Agent Models," in *Proc. IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, Mar. 2015, pp. 847–857.

[23] J. Wilkes, "More Google cluster data," Google research blog, Nov. 2011, [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html

Gagangeet Singh Aujla (S'15) received the B.Tech. and M.Tech. degrees in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2003 and 2013, respectively. He is working toward the Ph.D. degree with Thapar University, Patiala, India.

He has many research contributions in the area of smart grids, cloud computing, vehicular ad hoc networks, and sustainable computing.

Mr. Aujla is a member of the CSI and a life member of the ISTE.

Neeraj Kumar (M'16–SM'17) received the Ph.D. degree in computer science and engineering from Shri Mata Vaishno Devi University, Katra, India, in 2009.

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 University, Patiala, India. He has authored or coauthored more than 200 technical research papers in leading journals and conferences from the IEEE, Elsevier, Springer, and Wiley.

Albert Y. Zomaya (F'04) received the B.S. degree in electrical engineering from Kuwait University, and the Ph.D. degree in control engineering from Sheffield University, U.K., in 1987 and 1999 respectively. He is currently the Chair Professor of high-performance computing and networking with the School of Information Technologies, The University of Sydney, Sydney, Australia. He is also the Director of the Centre for Distributed and High Performance Computing, which was established in late 2009. He has authored or coauthored more than 1000 scientific papers and articles and has authored/coauthored/edited more than 20 books.

Prof. Zomaya is Editor-in-Chief of the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING AND ACM SUSTAINABLE COMPUTING. He is an Associate Editor for 22 leading journals. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS during 2011–2014. He received the IEEE Technical Committee on Parallel Processing Outstanding Service Award and, the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing in 2011, and the IEEE Computer Society Technical Achievement Award in 2014. He is a Chartered Engineer. He is a Fellow of the AAAS and the IET (UK).

Rajiv Ranjan (SM'15) received the Ph.D. degree in computer science and software engineering from the 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, Newcastle upon Tyne, U.K. He is also the Chair Professor with the School of Computer, China University of Geosciences, Beijing, China. Before this, he was a Julius Fellow (2013–2015), a Senior Research Scientist, and a Project Leader in the Digital Productivity and Services Flagship of CSIRO—Australian Government's 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.