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

Gagangeet Singh Aujla, Neeraj Kumar, Albert Y. Zomaya, Rajiv Ranjan

Institutions: Thapar University, University of Sydney, Newcastle University

Topics: Cloud computing, Edge device, Big data, Service provider and Server

Related papers:

- Data Offloading in 5G-Enabled Software-Defined Vehicular Networks: A Stackelberg-Game-Based Approach

- MEnSuS: An efficient scheme for energy management with sustainability of cloud data centers in edge–cloud environment

- Edge Computing in the Industrial Internet of Things Environment: Software-Defined-Networks-Based Edge-Cloud Interplay

- SDN-based energy management scheme for sustainability of data centers: An analysis on renewable energy sources and electric vehicles participation

- Optimized Big Data Management across Multi-Cloud Data Centers: Software-Defined-Network-Based Analysis

Share this paper: 👍 🐦 in ✉

View more about this paper here: https://typeset.io/papers/optimal-decision-making-for-big-data-processing-at-edge-4dd3tst730

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

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 geo-distributed 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 multi edge-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 multi edge-cloud environment using software defined networks (SDN). To handle the inter-DC migrations efficiently, a SDN-based control scheme is presented which provides energy-aware network traffic flow scheduling. Finally, a multi-leader multi-follower 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 shows the effectiveness of the proposed scheme.

Index Terms—Energy-efficiency, edge computing, cloud data centers, software-defined networks, Stackelberg game.

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, e-banking have generated a huge volume of heterogeneous data which varies with time [2]. To handle such massive data streams generated from these applications, existing DCs 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 on-line activities across US only. Moreover, with the advent of internet of things (IoT), the big data generated from different applications has 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 geo-distributed DCs.

Recent developments in CC sector has provided a multi-cloud environment which provides multiple cloud services through single heterogeneous computing architecture. Such multi-cloud environment provides low latency, high data rate, and non-disruptive services with respect to big data processing to the end users [2]. In this direction, large cloud service providers 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 8 countries across 4 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 needs to be transfered across geo-distributed DCs using the underlying networks. However, such 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 data sets 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 SLA violations to the cloud service providers (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 closer the proximity of the users using geo-distributed DCs. Yassine *et al.* [10] proposed a multi-rate 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 fog-cloud scenario. The authors put an emphasis on the fact that the cooperation between cloud and fog may help to achieve desired QoS and energy efficiency. Jalali *et al.* [12] presented a comprehensive analysis of CC and edge computing. Authors stressed on keeping the data closer to the end-user in order to achieve lower latency. However, in 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 energy-aware cloud-fog interplay. The authors focused that optimal interplay between fog and cloud DCs using 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 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 workflow allocation graph which considers the price diversity across geo-distributed 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, 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 SDN to improve network performance. Wang *et al.* [17] utilized 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 as given below.

- A workload slicing scheme for handling data-intensive jobs in multi edge-cloud environment is presented.
- A SDN-based controller is designed to provide an energy-aware flow scheduling scheme with access of virtualized network resources.
- A multi-leader multi-follower Stackelberg game is formulated for providing optimal inter-DC migrations.
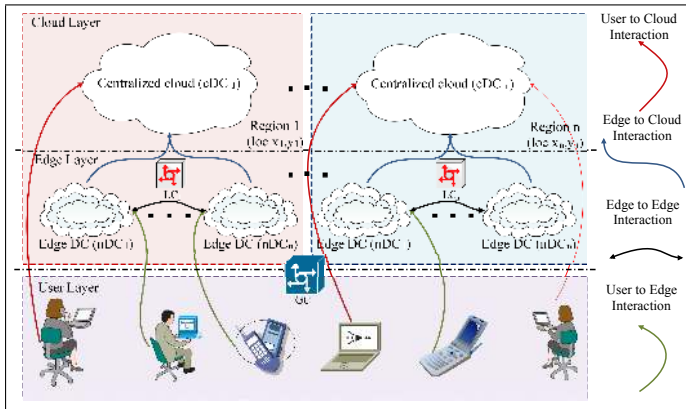

Fig. 1: System model

II. SYSTEM MODEL

Fig. 1 shows the system model comprising of a multi edge-cloud environment having n cloud and m edge geo-distributed 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 global controller is responsible for handling the workload classification and scheduling in multi edge-cloud environment. The local controller handle the inter-DC migrations.

NOMENCLATURE	
n, m, f	Number of cloud DCs, edge DCs, job type
W, W_{dt}, W_{ds}	Incoming, delay-tolerant, delay-sensitive workloads
F	Job with requirements: $a_{type}, b_{req}, c_{req}$
a_{type}	Type of application
b_{req}, c_{req}	Communication and Computational requirement
$\alpha(t)$	Arrival rate at time slot t
$Q_i^f(t+1)$	Size of queue for type f jobs at i^{th} DC at time $(t+1)$
$Q_i^f(t)$	Size of queue for type f jobs at i^{th} DC at time (t)
$Q_i^{pr}(t)$	Present size of queue at i^{th} DC at time (t)
$\lambda_i^f(t)$	Number of type f jobs routed to i^{th} DC
S_i	Total number of servers allocated incoming workload
μ_i	Processing speed of each server
SLA_p^v	SLA violations of pth server
t_p^{thr}	Time for which threshold utilization level is experienced
t_p^{act}	Total active time of server
D_p^{mig}	Performance degradation due to migration
t_i^{res}	Response time
t_{max}^{res}	Maximum achievable response time
D_i	Overall delay
D_i^{comm}	Communication delay
D_i^{mig}	Migration delay
D_i^{broc}	Processing delay
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 i^{th} edge device
E_i^P	Energy consumption of pth server of ith DC
E_i	Energy consumption of ith DC
E_i^c, E_i^o	Energy consumed for cooling & other activities of ith DC
E_i^{net}	Energy consumption of network resource of ith DC
E_{sw}^{net}	Energy consumption of switches
E_{port}^{net}	Energy consumption of ports
E_{idl}^P	Energy consumption of idle p^{th} server
E_{max}^P	Maximum energy consumption of pth server
U_i^P	Utilization of pth server of ith DC
$R^p(t)$	Amount of resources consumed at time t at k^{th} server
R_{max}^p	Amount of resources consumed at time t at k^{th} server
x_i^{edge}	Job requests handled by edge devices
E_i^{edge}	Energy consumed by i^{th} edge device
a_i, b_i, c_i	Pre-defined parameters for edge devices
C_i^f	Cost incurred for handling type f job at i^{th} DC
C_i^{tot}	Cost for handling type f job at i^{th} DC after migration
C_i^{comp}	Cost related to computing resources at i^{th} DC
C_i^{comm}	Cost related to communication at i^{th} DC
C_i^{eng}	Cost related to energy consumption at i^{th} DC
C_i^{pen}	Cost related to SLA violations at i^{th} DC
$C_{i \to k}^{mig}$	Cost related to migration from i^{th} DC to k^{th} DC
P_i, M_i, S_i	Processor, memory, storage required
ρ, ρ_e	Price coefficient for different resources and energy
C_i^L, C_i^{IDC}	Local and Inter-DC communication cost
C_i^{band}	Communication cost related to bandwidth requirements
b_{net}	Bandwidth cost coefficient
b_i^{comm}	Bandwidth requirement for communication
E_i^f	Energy required to handle type f job at i^{th} DC
$Y_i^f(t)$	Number of migrating jobs at time t
C_p^{slav}	Cost of SLA violation per unit time for p^{th} processor
T_p^{slav}	Duration of SLA violation for p^{th} processor
U_i, U_k	Utility function of ith and k^{th} DC
R_f	Revenue received for handling type f job
R_{mig}	Revenue received for hosting f job migrated
$\hat{U}_{ijk}^{map}$	Utility map of ijk pair
U_{ijk}	Utility function of ijk pair
ηav_i	Anticipated delay of the network after including new load
τ_i^{av}	Anticipated throughput after including new load
$d_{\frac{i \to k}{j}}$	Distance from i^{th} DC to k^{th} DC through flow path j
z_{ijk}	Decision variable

A. Workload model

Consider a workload (W) comprising of F type of jobs to be processed in multi edge-cloud environment. A job is described as $F : (a_{type}, b_{req}, c_{req})$ where a_{type}, b_{req}, and c_{req} denotes 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 i_{th} DC follow the queues dynamics [18] as below.

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

where, $\lambda_i^f(t)$ is the number of type f jobs routed to i^{th} DC.

B. QoS Model

SLA is the most important requirement during handling incoming workload in multi edge-cloud environment. If the resources required to process the workload exceeds 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 p^{th} 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 pth server of ith DC is given as below. [19].

$$SLA_p^v = \frac{1}{p} \sum_{p=1}^{P} \frac{t_p^{thr}}{t_p^{act}} D_p^{mig} \tag{2}$$

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

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

where, ϱ_w^{dg} and ϱ_w^{cp} denotes 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 below.

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

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 below.

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

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

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

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

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 ($D_{i \to k}^{mig}$) from i^{th} DC to k^{th} DC is given as below.

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

Now, in case an edge DC or devices is handling the job, then the delay incurred (D_i^{edge}) is defined using M/M/1 queuing model [?] and is given as below.

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

where, v_i and a_i denotes service rate and arrival rate of jobs.

C. Energy model

The energy consumption of a DCs comprise 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 ith DC is given as below.

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

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

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

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

The level of utilization of pth server of i_{th} DC depend on the amount of resources consumed ($R^p(t)$) at time t and maximum capacity of processor (R_{max}^p) and is given as below.

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

A major chunk of energy consumption of DCs depend on network infrastructure. The network devices consume energy on the basis of fixed energy consumption (E_{sw}^{net}) and dynamic energy consumption (E_{port}^{net}). So, the energy consumption of network devices in i^{th} DC is given as below.

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

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

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

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 q^{th} switch, working time of qth switch, dynamic power consumed by r^{th} port of qth switch, and working time of rth port of qth switch.

Now, expanding Eq. 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 average occupancy ratio of switch q, and Θ_r^q is 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 i^{th} edge device is given as below.

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

where, $a_m > 0$ and $b_m, c_m \geq 0$ are the pre-defined parameters.

D. Cost model

The operational cost (C_i^f) for handling type f job at i^{th} DC comprise of different sub-costs and is given as below.

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

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.

In some cases, migration of job from i^{th} DC to k^{th} DC occurs. Hence, a migration cost ($C_{i \to k}^{mig}$) 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 below.

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

The cost on computing resources allocated to handle a job depend 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 below.

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

where, ρ is 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 below.

$$C_i^{comm} = C_i^L + C_i^{IDC} \qquad (19)$$

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

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

where, b_{net} is the bandwidth cost coefficient.

The cost of energy (E_i^f) required to execute type f jobs at i^{th} DC is given as below.

$$C_i^{eng} = \rho_e E_i^f \qquad (21)$$

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

In order to meet QoS requirements, DCs have to migrate jobs to other DCs which involves a migration cost. The cost for migrating type f jobs from i_{th} DC to k_{th} DC over flow path j is given as below.

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

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

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

$$C_i^{pen} = \sum_p [C_p^{slav} T_p^{slav}] \qquad (23)$$

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

III. PROBLEM FORMULATION

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

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

For this purpose a combined utility is defined as below.

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

where, η_i^{av} and τ_i^{av} denotes the average anticipated throughput and delay of the network after including the new load. $d_{\frac{i \to k}{j}}$ is the distance from i^{th} to k^{th} DC through flow path j.

In order to select the optimal mapping from the above discussed matrix, a decision variable ($z_{ijk}, \forall t$) is defined as below.

$$z_{ijk} = \begin{cases} 1 & \text{for} \quad U_{ijk} > U_{ijk}^* \\ 0 & \text{for} \quad otherwise \end{cases} \qquad (26)$$

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

Hence, the objective function is formulated using integer linear programming and is given as below.

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

subject to following constraints

$$z_{ijk} \in [0,1] \qquad (28)$$
$$U_i(k) > U_i(k^*) \qquad (29)$$
$$U_k(t) > U_k(t-1) \qquad (30)$$
$$0 < \sum_f Q_i^{pr}(t)c_{req} \le S_i \qquad (31)$$
$$t_i^{res} \le t_{max}^{res} \qquad (32)$$
$$C_{i \to k}^{mig} < C_i^{pen} \qquad (33)$$
$$d_{(\frac{i \to k}{j})} < d_{(\frac{i \to k}{j})^*} \qquad (34)$$

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

IV. PROPOSED SCHEME

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

A. Workload slicing scheme for multi edge-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 pre-defined 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

---

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 $F : (a_{type}, b_{req}, c_{req})$
5:     **if** $F : (a_{type}, b_{req}, c_{req})$ are available with DC_i **then**
6:         Add workload $\to Q_N : (Q_1, Q_2, ..., Q_n)$
7:         Select flow path using Algorithm II
8:         Schedule job $F \to DC_i \to$ PPRR
9:     **else**
10:         Schedule job $\to DC_{i*} \to$ PPRR   ▷ $i \notin i^*$
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 $\to Q_M : (Q_1, Q_2, ..., Q_m)$
17:         Select flow path using Algorithm II
18:         Schedule job $\to ED_i \to$ PPRR
19:     **else**
20:         Add workload $\to Q_N : (Q_1, Q_2, ..., Q_n)$
21:         Schedule job $\to$ cloud DCs
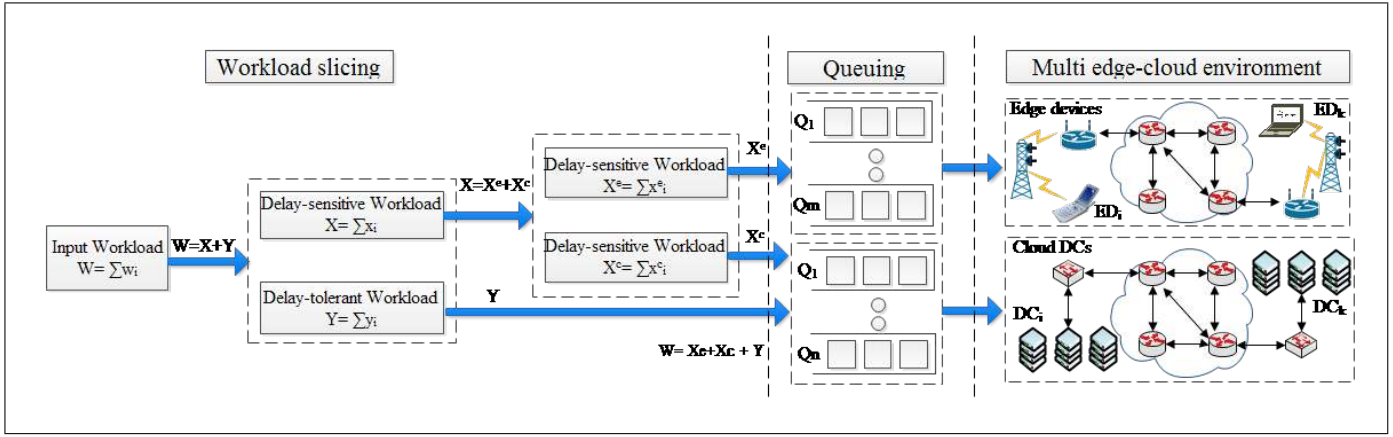22:   **end if**
23: **end if**

Fig. 2: Workload slicing scheme

geo-distributed cloud DCs. But, the real-time workload (X) is sub-divided into two parts as shown below. One part of workload (X^e) is scheduled to available edge DCs. However, there may be some workload (X^c) which require high computing resources that may not be available at edge DCs. Such workload slice is routed to cloud DCs. The architecture of workload slicing scheme is shown in Fig. 2. Algorithm 1 shows the working of proposed slicing scheme using prioritized preemptive round robin (PPRR) similar to [20] to schedule the jobs at DCs or edge devices.

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 SDN can flexibly adapt to new functionalities and network policies. The communication infrastructure in SDN follows open flow protocol [21], [22]. Fig. 3 shows the SDN architecture consisting of three decoupled planes: data, control, and application that are described as below.

1) Data plane: The data plane consist of FDs which act according to the forwarding decisions taken by SDN controller. Such decisions are configured into FDs using data-control plane interface. FDs contains 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 SDN controller. The instruction set consist of matching rule, priority, action, and statistics. The working of data plane in various steps as described below.

- Step 1: The source DCs that need to migrate the job to another DC sends a request. The request of DC is received by scheduler and queued for further processing.
- Step 2: The scheduler matches the input requirements with rules prescribed by SDN controller through instruction set. The matching rule consist of flow id, source IP address, source MAC address, virtual LAN address, port number, and transport protocols [21]. On the basis of matching rules, appropriate action is decided. The possible actions by FDs consists; forwarding, modifying, discarding, replicating, etc.
- Step 3: Once the appropriate action is decided, the request is forwarded to flow manager. This is followed by selection of appropriate flow table to complete the action.
- Step 4: Once the appropriate flow table is selected, the packets are migrated to 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.
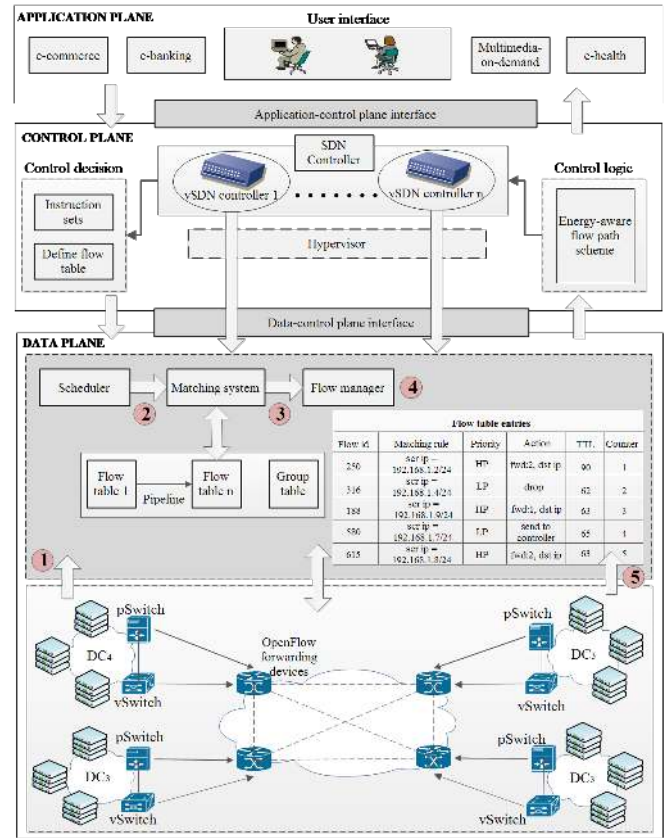

Fig. 3: Architecture of SDN-based control scheme

2) Control Plane: The control plane is the decision making plane which works on the basis of control logic. As the brain of the SDN architecture, all the forwarding decisions are taken by SDN controller. Using the control logic, the SDN controller forwards the programming and logic instructions into an instruction set. 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 large physical network into multiple virtual network. 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 act as an intermediate layer between the vSDN network and the underlying physical SDN network. Hence, allowing to exploit parallelism by running multiple NOS 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 given below [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 the load is evenly distributed. However, if the value of Υ is low, then it means the load is not evenly distributed and the controller needs to migrate the load using 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 which 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 appropriate queue for scheduling.

An energy-aware flow scheduling algorithm is presented to provide 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 sleep mode. Moreover, when all ports of a specific switch are in sleep mode, then the concerned switch is also put into 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 sleep mode, a decision variable ($\Psi_{syn}, \forall t$) is defined as below.

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

If (Ψ_{syn} = 0), then the switch shifts to sleep mode. For this purpose, a threshold time (t_{thr}) is considered. The value of Ψ_{syn} become 0 only if the switch is idle for threshold time (t_{thr}). The switch shifts back to 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 routed from DC_i to DC_k. The guaranteed flow rate (r_f) is given as below.

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

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 below.

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_{set} with no shared links
8:         **for** $f_{set} \in$ J **do**
9:           Calculate $t_{act} = activetime(f_{set})$
10:           Compute energy consumption using Eq. 14
11:           **if** (t_{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 to provision valid flow path
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 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 flows (f_{set}). After this, the active time (t^{act}) is computed for each element of f_{set}. In next step, the energy consumed by each element of 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 (line 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 (line 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 then rebuilds a valid flow path and then the incoming flow is scheduled again (line 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 (line 27-33).

3) Application plane: The purpose of application plane is to interact with various end user applications and provide feedback to controller through 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; (1) cloud to cloud, (2) edge to cloud, and (3) edge to edge. In order to participate in the migration the conditions shown in Table I may exist.

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

1) Stackelberg Game: The Stackelberg game is a strategic game in economics and is popular as a special case of non-cooperative games. It is two-period hierarchical game in which the players are classified as leader and follower [20]. Both the players in the game compete for the quantity and the leader is sometimes called as market leader. This is said because leader avails the benefit of initiating the game. By doing so leader can enforce is moves on followers. Bu, the leader must be aware *ex ante* that the follower observe 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 leaders best response is to play followers action [20]. Hence, in this way, both leaders and followers reach to an equilibrium state in order to maximize their payoffs. Stackelberg game has manifold advantages such as-(1) optimal choice in a distributed environment, (2) handles the economical aspects, (3) sequential movement of player, and (4) competitive behavior.

2) Why Stackelberg game?: In this work, the Stackelberg is is 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/edge devices) play their moves to reach an optimal solution. The source DC/edge devices act as leaders and announce their resource requirements to destination DCs/edge devices who act as followers. The game proceeds in a distributed edge-cloud environment where DCs or edge devices are geo-located. Moreover, the equilibrium between both the players is dependent on the economical factors (price and cost). Both leader and follower act in a sequential manner to compete with other DCs/edge devices. 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 multi-leader multi-follower Stackelberg game is presented. For this purpose, the game model comprises of following entities.

- **Players:** i source DCs/edge device (multiple leaders) and k destination DCs/edge device (multiple followers).
- **Strategy/action:** For leader, the strategy for selecting appropriate host for handling workload $F : (a_{type}, b_{req}, c_{req})$ is

$S_l = (f_1, f_2..., F_i)$, where $i \in I$. For followers, the strategy to finalize the price (P_k^{mig}) for hosting the migrating jobs is is $S_f = (p_1, p_2..., p_k)$, where $k \in K$.

- **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, 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 i^{th} DC that require to migrate data or job to k^{th} DC is given as below.

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

The utility function of kth DC that can be selected to handle the migrated data from i^{th} DC is given as below.

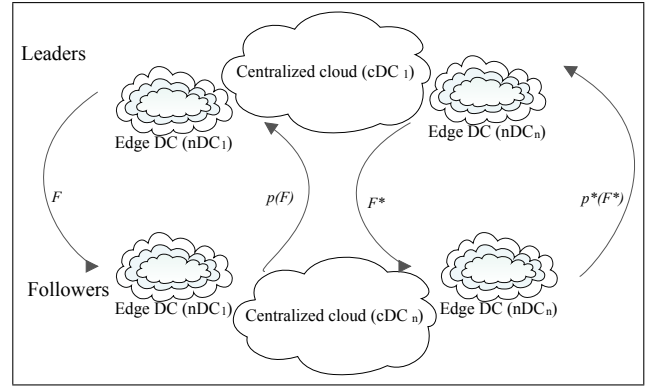$$U_k = [\sum_i R_{mig}] - \sum_{j=1}^{J}[C_i^f] \tag{39}$$

Fig. 4: Stackelberg game model

4) Proposed Stackelberg game-based algorithm: The working of Stackelberg game model is shown in Fig. 4. Using this model, a multi-leader multi-follower 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, multi leaders (*i* DCs) initiates 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 (line 2-15). In next move, the leader (DC_i) computes its utility (U_i) for each of the k DCs that have accepted the migration request (line 16-17). If the utility ($U_i(k)$) of DC_i with respect to k^{th} is more than ($U_i(k^*)$) of each of the DC other than k^{th} DC, then add DC_k in queue above DC_{k+1} (line 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 (line 22-23). 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 (line 24-25). Now, select the *ijk* pair and send consent to k^{th} DC. Once k^{th} DC confirms the deal, then migrate workload. Otherwise, select the next pair (line 26-34).

Algorithm 3 Stackelberg game for inter-DC migration

Input: DC_i, b_{req}, c_{req}
Output: flowpath j, ijk pair, DC_k

```
 1: for (i=1; i ≤ n; i++) do
 2:     F : (a_type, b_req, c_req) → DC_i                    ▷ Leader move
 3:     for (k=1; k ≤ n; k++) do
 4:         Check (c_req)                                    ▷ Follower move
 5:         if C_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 ≤ n; k++) do                            ▷ Leader move
17:         Compute U_i(k)
18:         if U_i(k) > U_i(k*) then
19:             Add DC_k in 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 k^th DC
27:             if 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
```

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, (1) only cloud DCs, (2) EDs, and (3) 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 (1) delay-sensitive and (2) 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 multi edge-cloud environment plays an important impact on the energy consumption of DCs. Fig. 5(e) shows the comparison of energy consumed by proposed multi edge-cloud environment with other two scenarios. The energy consumed by multi edge-cloud DCs is lower as compared to scenario when only cloud DC or EDs are used. Moreover, the proposed multi edge-cloud environment proves to be 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 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 is reduced. Moreover, with multiple options available for handling workloads, the SLA are easily met.

The proposed scheme uses SDN as underlying DC networks. The use of proposed energy-aware flow scheduling scheme for SDN reduces the energy consumption with respect to underlying networks. Fig. 5(g) shows that the energy consumed by proposed flow scheduling scheme for SDN 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 edge device so as to meet SLA. In that case, additional delay and cost are involved due to migration. However, appropriate selection of destination DC or ED that can serve the migrated job is an important task. The proposed multi-leader multi-follower 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 effective underlying networks and dynamic flow path can not only avoid additional delay but it can reduce migration cost also. In this context, Fig. 5(h) shows the migration delay incurred while migrating the jobs from source DC or ED to destination DC or edge device. The results depict a lower delay for SDN as compared to traditional networks. Moreover, the use of SDN has a strong impact in reduction of migration cost due to its dynamic and flexible nature. Fig. 5(i) shows that the migration cost for SDN 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 (4 cloud DCs and 5 EDs) is formulated. The value of decision variable (z_{ijk}) i 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 increases with respect to previous instance. Now, for first leader (DC_{l1}), the value of z_{ijk} is equal to 1 for 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 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

(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) Comparasion of energy consumption

(f) SLA violations

(g) Network energy consumption
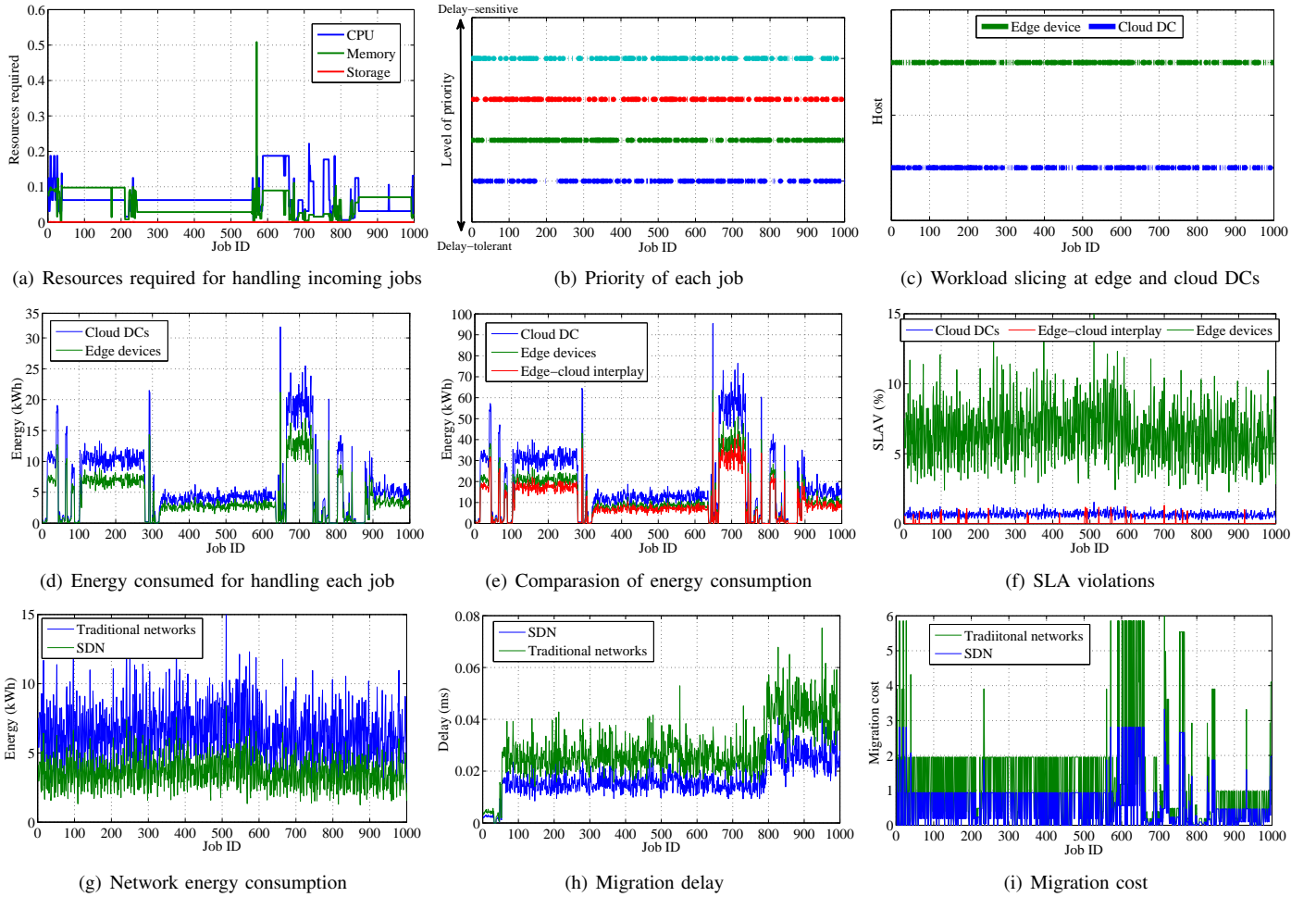
(h) Migration delay

(i) Migration cost

Fig. 5: Results obtained for single time-slot

leader. Similarly, DC_{l3} and DC_{l3} selects different destination DCs/EDs as per their resource and SLA requirements. So, the proposed game act as a optimal decision maker for destination host and flow path selection for inter-DC migrations.

B. Evaluation for 12-hour scenario

After analyzing the proposed scheme for single time-slot, it is evaluated for a longer time period (12-hrs). The results obtained clearly shows that the edge-cloud environment has 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 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 end up in 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 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 number of constraints. It clearly shows that the proposed problem is solvable till 10 number 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 edge-to-edge environment. It is evident that the edge devices 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 edge devices as compared to cloud DCs.

VI. CONCLUSION

In this paper, a workload slicing scheme has been designed for handling big data applications in a multi edge-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 multi-leader multi-follower Stackelberg game is formulated to

(a) Average SLA violations (b) Energy consumed (c) Migration rate

(d) Average overhead (e) Complexity analysis (f) Edge to edge analysis
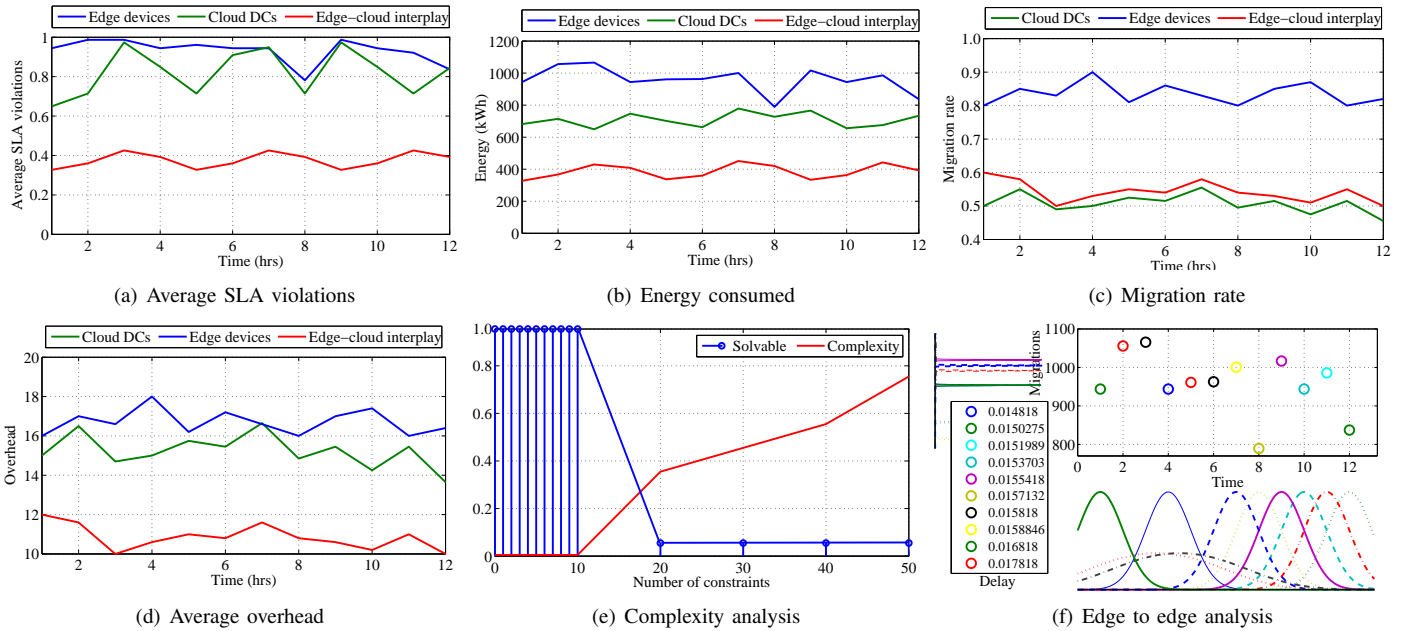
Fig. 6: Results obtained for 12-hr scenario

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 multi edge-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 Journal of Optical Communications and Networking*, 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 Network*, 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 and Anthesis*, pp. 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 Transactions on Computers*, 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 Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 314–323, Sep 2014.

[6] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Zomaya, and K. Wang, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE Transactions on Parallel and Distributed Systems*, 2016, 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 Transactions on Computers*, 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 Transactions on Computers*, 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 Transactions on Emerging Topics in Computing*, 2015, 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 Transactions on Cloud Computing*, 2016, 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 of Things Journal*, 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 Journal on Selected Areas in Communications*, 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 *IEEE International Conference on Communications (ICC)*, 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," *Computer Networks*, 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 Communications Surveys Tutorials*, vol. 18, no. 1, pp. 655–685, 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 Generation Computer Systems*, 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 *International Conference on Information Networking*, 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 Transactions on Parallel and Distributed Systems*, 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," *Concurr. Comput. : Pract. Exper.*, 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," 2017, doi: 10.1109/TCC.2017.2715817.

[21] G. S. Aujla, R. Chaudhary, N. Kumar, J. J. Rodrigues, and A. Vinel, "Data offloading in 5G-enabled software-defined vehicular networks: A stackelberg game-based approach," *IEEE Communication Magazine*, August 2017.

[22] G. S. Aujla, A. Jindal, N. Kumar, and M. Singh, "SDN-based data center energy management system using RES and electric vehicles," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

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