Container-based Load Balancing for Energy Efficiency in Software-defined Edge Computing Environment

Amritpal Singh[a], Gagangeet Singh Aujla[b,*], Rasmeet Singh Bali[a]

[a] *Computer Science & Engineering Department, Chandigarh University, Mohali, India*
[b] *Department of Computer Science, Durham University, Durham, United Kingdom*

Abstract

The workload generated by the Internet of Things (IoT)-based infrastructure is often handled by the cloud data centers (DCs). However, in recent time, an exponential increase in the deployment of the IoT-based infrastructure has escalated the workload on the DCs. So, these DCs are not fully capable to meet the strict demand of IoT devices in regard to the lower latency as well as high data rate while provisioning IoT workloads. Therefore, to reinforce the latency-sensitive workloads, an intersection layer known as edge computing has successfully balanced the entire service provisioning landscape. In this IoT-edge-cloud ecosystem, large number of interactions and data transmissions among different layer can increase the load on underlying network infrastructure. So, software-defined edge computing has emerged as a viable solution to resolve these latency-sensitive workload issues. Additionally, energy consumption has been witnessed as a major challenge in resource-constrained edge systems. The existing solutions are not fully compatible in Software-defined Edge ecosystem for handling IoT workloads with an optimal trade-off between energy-efficiency and latency. Hence, this article proposes a lightweight and energy-efficient Container-as-a-Service (CaaS) approach based on the software-define edge computing to provision the workloads generated from the latency-sensitive IoT applications. A Stackelberg game is formulated for a two-period resource allocation between end-user/IoT devices and Edge devices considering the service level agreement. Furthermore, an

*Corresponding Author
Email addresses: amritpal_bcet@yahoo.co.in (Amritpal Singh), gagi_aujla82@yahoo.com (Gagangeet Singh Aujla), rasmeetsbali@gmail.com (Rasmeet Singh Bali)

energy-efficient ensemble foe container allocation, consolidation and migration is also designed for load balancing in software-defined edge computing environment. The proposed approach is validated through a simulated environment with respect to CPU serve time, network serve time, overall delay, and lastly energy consumption. The results obtained show the superiority of the proposed approach in comparison to the existing variants.

Keywords: Container-as-a-Service, Edge computing, Stackelberg game, Software defined networking, Resource optimization.

---

1. Introduction

During the novel coronavirus or COVID-19 global pandemic, the delivery of online services to every household has been a major concern for every service provider. Most of the top service providers (Amazon, Google, eBay, etc) located across the globe rely on the Cloud computing paradigm for service provisioning [1]. However, in the current situation, the Cloud computing sector has been one of the most affected technologies as they are responsible for handling this situation of the enhanced workload with the huge dependence of the end-user domain. Cloud computing provides flexible and on-demand delivery of services and computation infrastructure (servers, storage, networking and software) to the end-users. Their services are hosted over geo-located cloud data centres (DCs) where ICT resources (servers, storage devices such as disks, communication networks), redundant or backup power supplies, environmental controls (e.g. air conditioning, fire suppression) and security devices are deployed to provide round the clock service through the world. Cloud ecosystem utilises virtualization technologies to schedule different types of workloads (e.g. scientific workflows, multi-tier web applications, IoT workloads) on the minimal number of servers to ensure the better utilized of resources. But, different workloads may have different resource utilization footprints and may further differ in their temporal variations. Although, cloud computing tries to provide round the clock resource footprints to the end-users to handle their workload from past decades, however, it has to witness the problem of sustainability and scalability in such COVID-like scenarios. Moreover, the Internet of Things (IoT) revolution has already added to the cloud workload from the past decade [2, 3].

The global escalation for cloud resources results in two major challenges, 1) a drastic increase in energy consumption, and 2) degradation in the re-

sponse time/latency for the desired online services for latency-sensitive applications and IoT systems. The first challenge concerns the growing demand for electricity and related carbon emissions that are caused by the expansion of massive DCs. Looking into the facts, in the year 2000, DCs consumed 70 billion kWh of energy, that further increased to 330 billion kWh by 2007 [4] and it was projected to touch 1000 billion kWh till 2020 [4]. But, the current pandemic scenario can take these projections to a further higher level. This drastic increase in energy consumption ultimately leads to the overall expenditure of the DCs and end up in harmful carbon emissions. Therefore, it is the biggest responsibility of global service providers to design and utilize energy-efficient approaches and solutions [5, 6, 7]. The second challenge relates to the advancements in IoT workloads and mission-critical applications that require lower latency and higher data rate. These stringent requirements if not fulfilled may end up in mission failure or unsatisfactory performances in IoT-based applications and systems (smart homes, smart grid, etc).

Edge Computing [8] has come up as a promising paradigm that can complement the cloud to provide resources or process IoT workloads closer to the location of the data source. This provides an add-on layer to process the data on local servers rather than forwarding it to the remote cloud, thereby improving the quality of service (QoS). Like the majority of cloud providers often create a geo-distributed multi-cloud environment across different countries [9], similarly, there can be a local multi-edge environment connected via the software-defined network (SDN) for service provisioning in a limited landscape [10, 11]. This provides an alternate solution for the cloud providers to schedule their delay-intensive workloads locally and computationally heavy workload at the remote data centres. However, there may be one another challenges in the local software-defined edge computing ecosystem that relates to the mobility of the end-users (like vehicles or drones in a smart city). This challenge may need more percentage of service or data migrations happening across different edge nodes (or servers) located across a geographic layout (like smart cities). These increased migrations can lead to service breakage, increased energy consumption, and degraded response time. Moreover, it may take additional resources, energy and delay to reconfigure and re-establish the lost like to restart the services. To resolve these challenges, Container-based virtualization, a lightweight approach can minimise the additional energy consumption and delay due to dynamic migrations happening across software-defined edge computing ecosystem [12] [13] [14]. The major research questions that arise from the above discussion

are listed below.

- **How to execute an IoT or mission-critical workloads across multiple edge servers via SDN?**
- **How to minimize the service breakage due to mobility and control the service link re-establishment consequences using container-based virtualisation?**
- **How to optimize the consumption and balance the load among multiple edge servers while avoiding SLA violations?**

1.1. Contributions

To answer the above-mentioned research questions, in this paper, we present a container-based load balancing approach for energy-efficiency in the software-defined edge computing environment. The key building blocks of the proposed approach are as follow:

- A multi-layered system model for software-defined edge computing is proposed for handling diverse IoT workloads in an energy-efficient manner. To achieve this, a multi-objective driven task scheduling scheme based on energy, delay, and service level of agreement (SLA) is designed.
- A multi-leader multi-follower Stakelberg game is formulated for energy-aware resource allocation is designed for scheduling IoT workload at the edge layer.
- An energy-efficient ensemble for container allocation, consolidation, and migration is proposed for horizontal load balancing scheme using container-based virtualization is designed.

2. Related work

Various existing works have addressed the above challenges related to the related to network latency and energy consumption for workload using the cloud computing landscape. For example, Berl *et al.* [15] discussed different energy efficient approaches used for the cloud infrastructure. The techniques and methods for resource allocation highlighted by Dabbagh *et al.* in [16]

relates to the reduction of the energy consumption by considering the virtualization platform. A thermal-aware task allocation approach for multi-core systems was proposed by Sheikh *et al.* [17] to manage the computing resources for multi-core systems. Another approach for task allocation using cooperative game model using Nash bargaining concept was introduced by the Khan *et al.* [18] for computational grids to optimize the energy consumption. However, the increased energy consumption and higher latency are still biggest challenges in the cloud computing environment. A discussion on scheduling and allocation approaches on single, multi-core and distributed systems for reducing the energy and power consumption were highlighted by Sheikh *et al.* [19]. To resolve the energy consumption challenges, some of the authors utilized edge computing to compliment the workloads of the Cloud. For example, Zhao *et al.* [20] proposed an efficient scheduling scheme for cloud resources to satisfy the QoS for Analytics as a Service (AaaS) alongside minimizing the energy consumption of DCs. A context-aware approach was proposed by Dar *et al.* [21] to categorize the data and device according to their confidentiality level to minimize the energy consumption at different levels. Alqahtani *et al.* [22] proposed a greedy heuristic approach to assign the computing resources at the edge level in order to process the tasks with minimal energy consumption. An efficient resource allocation and resource adjustment for services by integrating Markov decision process approach at edge level was proposed by Deng *et al.* [23] to sustain the service-level-agreement index for end users. Another approach considering edge computing platform proposed by the Liao *et al.* [24] utilized machine learning combined with Lyapunov optimization, and matching theory selection framework. This work considered a trade off between reliability of service, energy consumption, and backlog awareness. A complete mapping scheme was proposed between jobs to virtual machines and virtual machines to physical machines according to the behaviour of the incoming jobs by Mishra *et al.* [25] with a key focus to optimize the energy consumption. Another approach using two-tier virtual machine platform was proposed by the Chen *et al.* [26] to resolve the issue of mapping of controller and virtual machine and minimize the energy consumption during processing the workload. However, none of the above approaches utilized the container-based virtualization. In this direction, a multi-index task classification and scheduling approach was introduced by Kumar *et al.* [27] based on container-as-a-service for cloud data centers. The proposed scheme forwarded the incoming task to the data centers with desired resources based on the underlying priority of the workloads.

The mobility of the end devices can impact the SLA and QoS agreed to the end users by the service provider. Therefore, an intelligent and adaptive platform is required to handle the workload of the movable end devices. In this regard, an SDN-enabled energy management approach was introduced by Aujla *et al.* [28] to control the consumption of energy at various data centers to process the allocated jobs. Authors associated the renewable energy resources with the data centers to control the energy consumption during processing the jobs and reduce carbon emissions. They integrated the energy-efficient flow scheduling algorithm with SDN-enabled network to control the traffic flow in the network. The results proved the effectiveness of the scheme when compared with standard approaches. A MapReduce-based task scheduling approach was proposed by Kaur *et al.* [29] to reduce the energy consumption during processing of the tasks. During peak hours, UPSs contribute to provide the energy to the DCs to process the tasks. The evaluated results validated the proposed scheme in terms of DCs sustainability. However, the above mentioned approaches focus on edge computing platform and have entire focus on cloud data centers. Hence, it becomes very relevant to analyse the impact of container-based virtualization on the distributed edge nodes using a SDN-based network architecture. A recent work [30] considers an energy-efficient approach in the egde-cloud environment to resolve some of the highlighted challenges. A comparison of the existing proposals related to cloud resources is shown in Table 2.

3. System Model

The proposed system model comprises the different layers in a software-defined edge computing environment to handle IoT workloads and end-user requests. The proposed system model is depicted in the Fig. 1. The multi-layered system model is discussed as below:

- **User layer:** This layer comprises of the different end-user device (static devices: laptops, and dynamic devices: car, cell phones) and IoT sensors (or devices) located at geo-distributed positions. At this layer, the i users (V_i) demand for m type of n resources (memory, bandwidth, storage space, processing power) for data processing and analysis. The bottom layer is also known as the workload generating layer and it uses the underlying network infrastructure to transfer the data to the next layer for further action.

Table 1: Comparisons of the existing works

Author	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Berl *et al.* [15]	✓	×	×	server	×	×	✓
Dabbang *et al.* [16]	✓	×	✓	virtual machine	×	×	✓
Sheikh *et al.* [17]	✓	×	✓	server	×	×	×
Khan *et al.* [18]	✓	✓	×	server	×	×	✓
Sheikh *et al.* [19]	✓	×	✓	server	×	×	×
Zhao *et al.* [20]	✓	×	✓	server	×	×	✓
Alqahtani *et al.* [22]	✓	×	×	server	×	✓	✓
Deng *et al.* [23]	✓	×	×	servers	×	✓	✓
Liao *et al.* [24]	✓	×	×	server	×	✓	✓
Mishra *et al.* [25]	✓	×	×	virtual machine	×	×	×
Chen *et al.* [26]	✓	×	×	virtual machine	×	×	×
Kumar *et al.* [27]	✓	×	✓	container	×	×	✓
Aujla *et al.* [28]	✓	×	✓	server	✓	✓	✓
Aujla *et al.* [31]	✓	✓	×	server	×	×	✓
Proposed	✓	✓	✓	container	✓	✓	✓

- 1: Energy efficiency, 2: Game theory, 3: Load balancing, 4: Type of resource, 5: SDN, 6: Edge computing 7: SLA violation.

- **Edge layer:** This layer comprises of j edge servers (or devices) (S_j) that are deployed at geo-distributed locations to provide the computing and storage resources for handling (processing or analysing) the end-user (V_i) or IoT workloads. The User layer forwarded the workload requirements ($\mathbb{R}_i^{req}$) from the connected devices/users that are converted into a job ($\mathbb{J}_i$) that is executed using sufficient edge resources.

- **Data plane layer:** This layer includes the forwarding infrastructure (like, switches, hub, etc) that are used to forward the incoming workload to the destination for further processing. These forwarding devices follow the flow rules installed by the controller to forward the incoming data traffic to the next hop to reach its ultimate destination.

- **Controller layer:** At this layer, a logically centralized controller is deployed to build the flow rules and policies for the flow tables installed at the forwarding devices. The controller works as per the OpenFlow

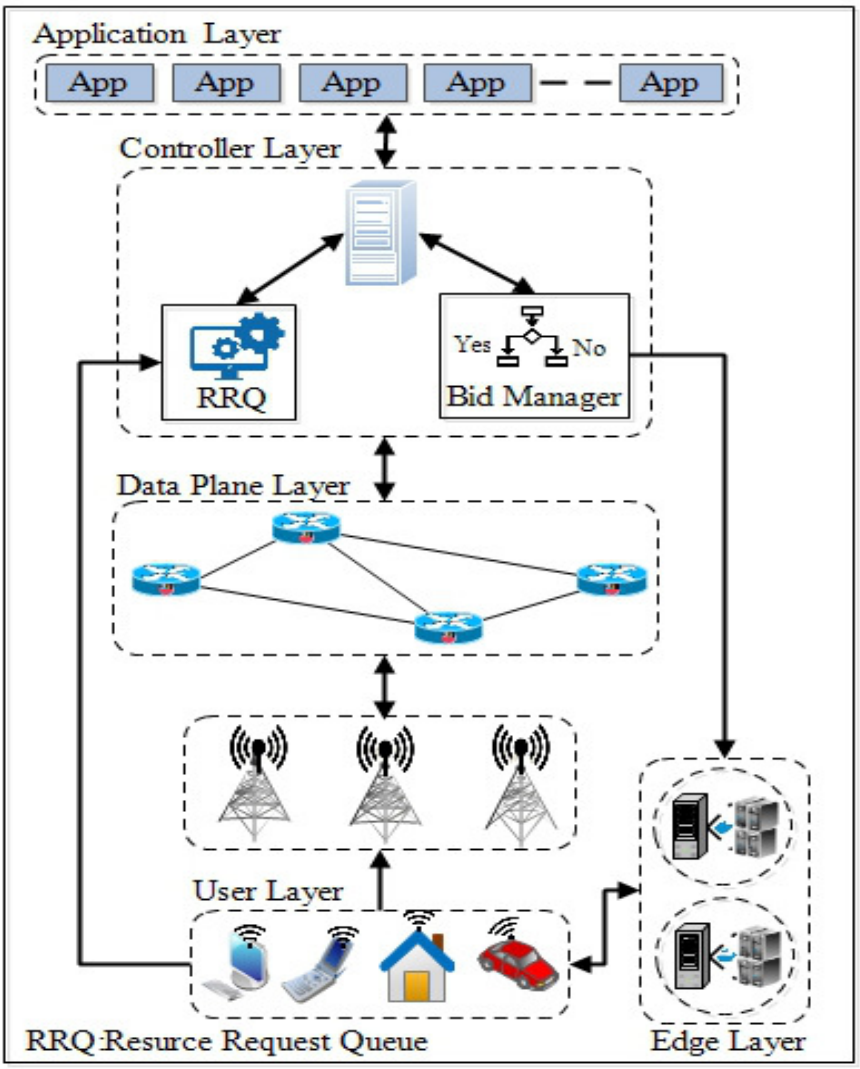

Figure 1: Software-defined edge computing model for service provisioning

protocol to build (rebuild) the flow policies to provide the desired latency in the network. The IoT workload is forwarded to the selected port as per the flow table entries directed by the controller. Moreover, a bid manager is also located at this layer to manage the resources available at the edge servers centrally. The resource demand of each

user or IoT workload is forwarded to the resource request queue (RRQ) for processing. The bid manager selects the bidding request from RRQ and maps it to the available resources at the edge layer. If the available resource are more than the resources requested, then they are allocated and the associated container is initialized to process the workload. In this model, V_i users request for various type of resources ($\mathbb{R}_i^{req}$) from the service provider for executing job ($\mathbb{J}_i$). The availability of the resources ($\mathbb{R}_{k\to j}^{ava}$) is checked by the bid manager as follows.

$$\mathbb{R}_{k\to j}^{ava} = \mathbb{R}_{k\to j}^{tot} - \mathbb{R}_{k\to j}^{asg} \tag{1}$$

where, $\mathbb{R}_{k\to j}^{tot}$ represents the total resources at the edge servers and $\mathbb{R}_{k\to j}^{asg}$ are resources currently assigned to the end-users. If the required amount of resources are available, then the bid manager uses the proposed Stackelberg game presented in the subsequent sections to select the optimal resource list and forward the matched list to the service provider for resource allocation.

- **Application layer:** This layer comprises of various end-user or IoT applications that provide the feedback to the controller to modify the rules or policies dynamically.

4. Preliminaries and Problem Formulation

The problem formulation is based on three metrics, i.e., delay, energy, and SLA. The detailed explanation about these metrics are given as below.

4.1. Delay

The total delay in the software-defined edge ecosystem comprises of computation delay ($\mathbb{T}_{\mathbb{J}_i}^{res}$), migration delay ($\mathbb{T}_{\mathbb{J}_i}^{mig}$), network delay ($\mathbb{T}_{\mathbb{J}_i}^{net}$) and caching delay ($\mathbb{T}_{\mathbb{J}_i}^{cac}$) [32]. The overall delay for executing i^{th} job is given as below.

$$\mathbb{T}_{\mathbb{J}_i} = \mathbb{T}_{\mathbb{J}_i}^{res} + \mathbb{T}_{\mathbb{J}_i}^{mig} + \mathbb{T}_{\mathbb{J}_i}^{net} + \mathbb{T}_{\mathbb{J}_i}^{cac} \tag{2}$$

Now, $\mathbb{T}_{\mathbb{J}_i}^{res}$ refers to the totakl response time witnessed while executing the job and it is defined as below [32].

$$\mathbb{T}_{\mathbb{J}_i}^{res} = \frac{1}{\varrho_j \times \eta_j - \Theta_i(t)} + \frac{1}{\varrho_j} + (\mathbb{T}_{\mathbb{J}_i}^{net})(\eta_j(t)) \tag{3}$$

Table 2: Notation Table

Notation	**Description**
$\mathbb{A}$	Available resources at edge and cloud layer
$\mathbb{B}$	Price list offered by the follower to the leader
$\mathbb{E}$	Energy consumption
$\mathbb{F}$	Demand and fulfillment list of followers and leaders
$\mathbb{L}$	List of demand resources by follower
$\mathbb{N}$	Combinatorial structure of all resources
$\mathbb{P}$	Followers resources demand
$\mathbb{S}$	List of all Followers
cpu	CPU computation power
mem	Memory of the server
bw	Bandwidth of the network
i	End users index
j	Server's index
k	Container's index
th	Threshold value of resources
P_r	Price of resources
P^{lp}	Loss price of the bid
C	Cost of resources
$\mathfrak{C}$	Container
$\mathbb{H}_{hl}$	Historical lowest bid price
$\mathbb{H}_{hh}$	Historical highest bd price
$\mathbb{D}_{hd}$	Historical demand fulfillment ratio
I_n	Normal Information for standard Follower
U	Utility function
S	Server
$\mathbb{J}$	Job/Tasks
V	Connected devices at User end

where, ϱ_j is the computation speed of the j^{th} server, η_j are the total number of servers used to process the task, Θ_i is the queue of the ready tasks, and $\mathbb{T}^{net}_{\mathbb{J}_i}$ is the delay in the network.

The network delay ($\mathbb{T}^{net}_{\mathbb{J}_i}$) is calculated by adding propagation, serializa-

tion, queuing, processing and jitter delay as given below [32].

$$\mathbb{T}^{net}_{\mathbb{J}_i} = \Big(\frac{d_{sw1,sw2} \times \psi_{f,sw1}(t) \times \psi_{f,sw2}(t) \times a_{sw1,sw2}}{\mathbb{T}_{med}(t)}\Big)^{prop} + \Big(\sum_{z\in sw}\sum_{j\in p}\frac{P_{z,j}(t)}{B_{z,j} \times \Delta_{z,j}(t)}\Big)^{ser} \tag{4}$$

$$+ \Big(\sum_{z\in sw}\sum_{j\in p}\frac{|\Theta(t)|}{B_{z,j} \times \Delta_{z,j}(t)}\Big)^{que} + \Big(\sum_{z\in sw}\mathbb{T}^{z}_{pr}(t) \times \sum_{f}(t^{end}_{f} - t^{str}_{f}) \times \psi_{f,z}(t)\Big)^{pro} \tag{5}$$

where, $d_{sw1,sw2}$ is the measured distance between switches, $\psi_{f,sw1}(t)$ and $\psi_{f,sw2}(t)$ are the decision variable for flow considering the switches, $a_{sw1,sw2}$ used to check the adjancy between switches, $\mathbb{T}_{med}$ is the used medium propagation delay, $P_{z,j}(t)$ is the size of the packet at time (t), $B_{z,j}$ is the considered bandwidth at z^{th} switch, $\Theta_{z,j}$ is the switch occupancy ratio, $\Theta(t)$ is the ready queue at time t, $\mathbb{T}^{z}_{pr}$ denotes the processing delay, t^{end}_{f} is the finish time of f flow and t^{str}_{f} is the starting time of f flow.

The delay to migrate the job from k_1 container to k_2 container as well as $\mathbb{T}^{ext}_{\mathbb{J}_i}$ delay, i.e., delay to migrate the i^{th} job from j_1 node to j_2 node are defined as below [33].

$$\mathbb{T}^{mig}_{\mathbb{J}_i} = \mathbb{T}^{int}_{\mathbb{J}_i} + \mathbb{T}^{ext}_{\mathbb{J}_i} \tag{6}$$

The internal migration relates to the time taken to migrate the job from one containers to other on same edge node. It is given as below.

$$\mathbb{T}^{int}_{(\mathbb{J}_i,j,k_1\rightarrow k_2)} = \mathbb{T}^{sen}_{\mathbb{J}_i,j,k_1} + \mathbb{T}^{rec}_{\mathbb{J}_i,j,k_2} \tag{7}$$

The external migration refers to the time taken to migrate the job from one edge node to another edge node. It is mentioned as below.

$$\mathbb{T}^{ext}_{(\mathbb{J}_i,j_1\rightarrow j_2,k)} = \mathbb{T}^{sen}_{\mathbb{J}_i,j_1,k} + \mathbb{T}^{rec}_{\mathbb{J}_i,j_2,k} \tag{8}$$

where, $\mathbb{T}^{net}_{\mathbb{J}_i}$ is the delay incurred to transmit the job from user (V_i) to the node, $\mathbb{T}^{net}_{\mathbb{J}_i}$ is the delay incurred to transmit the i^{th} job from j_1 to j_2.

The caching delay $\mathbb{T}^{cac}_{\mathbb{J}_i}$ in the network is calculated by leaving time (t^{lea}) from the edge and the new entry time (t^{ent}) on the new edge and is given as below [32]:

$$\mathbb{T}^{cac}_{\mathbb{J}_i} = t^{ent} - t^{lea} \tag{9}$$

4.2. Energy Consumption

The entire life cycle of service provisioning involves a huge amount of energy consumption associated with different utilized resources and processes during the execution of $\mathbb{J}_i$ at $S_{k \to j}$ edge node. There are different levels at which energy is consumed, i.e., computing ($\mathbb{E}_{ijk}^{com}$), network ($\mathbb{E}_{ijk}^{net}$), migration of services ($\mathbb{E}_{ijk}^{mig}$) and data caching ($\mathbb{E}_{ijk}^{cac}$) for i^{th} user with k^{th} container of j^{th} server at edge node. The total energy consumed ($\mathbb{E}_{ijk}^{tot}$) during service provisioning is defined below.

$$\mathbb{E}_{ijk}^{tot} = \mathbb{E}_{ijk}^{com} + \mathbb{E}_{ijk}^{net} + \mathbb{E}_{ijk}^{mig} + \mathbb{E}_{ijk}^{cac} \tag{10}$$

Now, $\mathbb{E}_{ijk}^{com}$ is based on the summation of energy consumption of idle server ($\mathbb{E}_{j}^{idle}$) and maximum energy consumption ($\mathbb{E}_{j}^{max}$). It is defined as below [32].

$$\mathbb{E}_{ijk}^{com} = \mathbb{E}_{j}^{idle} + (\mathbb{E}_{j}^{max} - \mathbb{E}_{j}^{idle})\mathbb{U}_{j}^{k} \tag{11}$$

In the above equation, the utilization of j^{th} server with k^{th} running container ($\mathbb{U}_k$) is calculated on the basis of the resources consumed ($\mathbb{R}_{k \in j}$) at time t and the maximum available resources ($\mathbb{R}_{k \in j}^{max}$). Here, $\mathbb{U}_k$ is computed as given below [32].

$$\mathbb{U}_{j}^{k} = \left(\frac{\mathbb{R}_{k \in j}(t)}{\mathbb{R}_{k \in j}^{max}} \right) \times 100 \tag{12}$$

The network infrastructure is the another major energy consumption and it is based on two components, i.e., fixed ($\mathbb{E}_{j}^{fix}$) and dynamic ($\mathbb{E}_{j}^{dyn}$). The fixed component relates to the part like fan, chassis, etc and the dynamic components comprises of the active ports and links. The total network-based energy consumption ($\mathbb{E}_{ijk}^{net}$) in the edge ecosystem is given as below [33].

$$\mathbb{E}_{ijk}^{net} = \mathbb{E}_{j}^{fix} + \mathbb{E}_{j}^{dyn} \tag{13}$$

Now, to meet the changing SLA requirements, often service migration happens in the edge environment. There can be two types of migrations, i.e., internal (between containers of same server) from container $k_1 \to k_2$ or external (between different servers) from $j_1 \to j_2$. The total energy consumed for the migration of services is defined as below [33].

$$\mathbb{E}_{ijk}^{mig} = \mathbb{E}_{k_1 \to k_2}^{int} + \mathbb{E}_{j_1 \to j_2}^{ext} \tag{14}$$

The energy consumed for internal migration is at both ends, i.e., sender end ($\mathbb{E}^{sen}_{k_1}$) and receiver ($\mathbb{E}^{rec}_{k_2}$) and is given as follows

$$\mathbb{E}^{int}_{k_1 \to k_2} = \mathbb{E}^{sen}_{k_1} + \mathbb{E}^{rec}_{k_2} \tag{15}$$

The energy consumed for external migration also occurs at sender ($\mathbb{E}^{sen}_{j_1}$) and receiver ends ($\mathbb{E}^{rec}_{j_2}$), respectively. The energy consumed due to external migration is given as below

$$\mathbb{E}^{ext}_{j_1 \to j_2} = \mathbb{E}^{sen}_{j_1} + \mathbb{E}^{rec}_{j_2} \tag{16}$$

If there is a miss in the flow table, $Packet\_In$ request is forwarded to the controller for generation of new entry and same is updated in all the available flow tables and in the cache of the network. So, there is energy consumption at sender side and receiver side while updating the entries in all the caches in the network. The cache energy consumption ($\mathbb{E}^{cac}_{ijk}$) is given in Eq. 17 [32].

$$\mathbb{E}^{cac}_{i(j_1 \to j_2)k} = \mathbb{E}^{sen}_{j_1} + \mathbb{E}^{rec}_{j_2} \tag{17}$$

4.3. SLA Violation

The service providers are bounded for the availability of the services to the end-users (V_i) with respect to the desired SLA commitments. The SLA ($\aleph^{k \to j}_i$) considering i^{th} job assigned on k^{th} container of j^{th} server at time t is calculated as below [33].

$$\aleph^{k \to j}_i = \sum_i \sum_j \sum_k \left(\frac{\mathbb{R}^{req}_i(t) - \mathbb{R}^{all}_{ijk}(t)}{\mathbb{R}^{req}_i(t)} \right) \tag{18}$$

If in a case, the SLA commitments are not met by the service provider, then they have to bear a penalty to compensate the end-users. The penalty associated with the SLA violations ($\mathbb{P}^{\aleph}_{jk}$) is calculated considering the cost $\Upsilon^{\aleph}_j$ of each violation and the duration of time ($t^{\aleph}_j$) the violation occurred on j^{th} server as defined below [33] .

$$\mathbb{P}^{\aleph}_{jk} = \sum_j (\Upsilon^{\aleph}_j t^{\aleph}_j) \tag{19}$$

4.4. Problem formulation

The above mentioned models are considered for the proposed scheme and two type of mapping variables are defined as below.

4.4.1. Job resource mapping variable

A decision flag ($\alpha_{i,j,k}$) is used to map the job with the allocated resources at edge node is represented as follows.

$$\alpha_{ijk} = \begin{cases} 1: & \text{If } \mathbb{J}_i \text{ is mapped to } k^{th} \text{ container of } j^{th} \text{ server} \\ 0: & \text{Otherwise} \end{cases}$$

4.4.2. Server to container mapping variable

A decision flag (β_{jk}) is used to address the mapping of the k^{th} container with j^{th} edge server is represented as follows.

$$\beta_{jk} = \begin{cases} 1: & \text{If } k^{th} \text{ container is mapped to } j^{th} \text{ edge server} \\ 0: & \text{Otherwise} \end{cases}$$

4.5. List of Objective functions

The proposed scheme consider the following multiple objectives that should be satisfactorily achieved.

1. ***Minimal Energy Consumption***: The first objective function $F_1(\alpha_{ijk}, \beta_{jk})$ is considered for minimal energy consumption to maintain the QoS and SLA. The function is given as below.

$$F_1(\alpha_{ijk}, \beta_{jk}) = min \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} \left(\mathbb{E}_{ijk}^{tot} \right) \times \alpha_{ijk} \times \beta_{jk} \tag{20}$$

2. ***Minimal Delay*:** The second objective function $F_2(\alpha_{ijk}, \beta_{jk})$ is considered for minimal delay during the transmission of data from one layer to other. The function is given as below.

$$F_2(\alpha_{ijk}, \beta_{jk}) = min \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} \left(\mathbb{T}_{\mathbb{J}_i} \right) \times \alpha_{ijk} \times \beta_{jk} \tag{21}$$

3. ***Maximal commitment towards SLA***: The third objective is to provide continuous services to the users to maintain the SLA. The objective function is $F_3(\alpha_{ijk}, \beta_{jk})$ is given as below.

$$F_3(\alpha_{ijk}, \beta_{jk}) = max \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} \left(Pen_{jk}^{SLA} \right) \times \alpha_{ijk} \times \beta_{jk} \tag{22}$$

4.6. List of constraints

In concern to the above mentioned objectives, the list of constraints are mentioned below.

$$\mathbb{H}1 : \sum_{i=I} \sum_{j=J} \sum_{k=K} \mathbb{R}^{pro} \times \mathbb{R}^{all}_{ijk} \leq \mathbb{P}^{ul}$$

$$\mathbb{H}2 : \sum_{i=I} \sum_{j=J} \sum_{k=K} \mathbb{R}^{mem} \times \mathbb{R}^{all}_{ijk} \leq \mathbb{M}^{ul}$$

$$\mathbb{H}3 : \alpha_{ijk} \in 0, 1; \forall i, j, k$$

$$\mathbb{H}4 : \beta_{jk} \in 0, 1; \forall j, k$$

$$\mathbb{H}5 : \mathbb{B}_n > 0; \forall n$$

$$\mathbb{H}6 : \mathbb{O}_n > 0; \forall n$$

5. Stakelberg Game-based Proposed Workload Allocation Scheme

The aim of the proposed scheme is to provide an energy-efficient solution for handling IoT workflows in a software-defined edge computing environment. To maintain scalability and minimal overhead, container-based virtualization is configured at edge servers. Also, a mobility scenario is considered where the services are migrated from one server to another. The proposed scheme works on the basis of multi-leader multi-follower Stakelberg game [31] for the allocation of resources to the incoming IoT workloads.The preliminaries related to the Stakelberg game are provided as below.

5.1. Game formulation

A *multi-leader* and *multiple-follower* stackelberg game is used to allocate the resources to the end users. Here, the end users (multi-leaders) demand for the resources from the edge servers (multi-followers). The bid manager communicate with the controller to fetch the cost of each edge server and maps the same with the list of the demanded resources for each incoming workload. The stackelberg game operates in non-cooperative manner between different multiple followers [31]. Initially, the leaders start the game and forward the list of required resources to the bid manager and track the actions of the followers. The followers analyse the required resources and revert to the leaders with the best offers. The leader-follower play their moves and finally achieve an equilibrium state with both having best possible utility

from where they can not back out.

The utility function used by the leaders and followers to calculate the benefits (price, cost or revenue) are defined as below.

- **Utility function of leaders:** The end users pay the price (P_i) from the generated revenue (R_i) to avail the resources from the edge servers. Their utility function is defined as below.

$$U_i^l = R_i - P_i \tag{23}$$

  If the calculated value of U_i^l is greater in comparison to the value calculated in previous requests, the value is finalized for bidding.

- **Utility function of followers:** The revenue (R_j^k) of j^{th} edge node with k^{th} container depends upon the bid price of the follower (P_i) and is given as below.

$$R_j = \sum_i P_i \tag{24}$$

  There cost (C_j^f) part comprises of may additional expenditure like maintenance cost (C_j^{mnt}), SLA violation cost (C_j^{sla}), migration cost (C_j^{mig}) and energy utilization price (P_j^{utl}) [31]. The total cost at the follower side is defined as below.

$$C_j^f = C_j^{mnt} + C_j^{sla} + C_j^{mig} + P_j^{utl} \tag{25}$$

  Now, acceptance or rejection at the follower end relies in the hands of the Bid Manager based on the utility function of the followers. Based on the costs and revenue, the utility function (U_j^f) of an edge node is given as below.

$$U_j^f = R_j^f - C_j^f \tag{26}$$

  After calculating the value of U_j^f, the bid manager accept the request, that provides profit to the followers.

5.1.1. Resource allocation Scheme using Stackelberg Game

The limited resources at edge servers are allocated using the two-period Stackelberg game considering leaders and followers. Both the parties try to maximise utilities, 1) leaders in the form of more resources in the limited cost,

and 2) the followers in the form of cost for the usage of the resources by the leaders. The basis preliminaries related to the proposed resource allocation approach are defined as follows.

$$\mathbb{N} = (\mathbb{P}, \mathbb{B}, \mathbb{L}, \mathbb{A}, \mathbb{F}) \tag{27}$$

where, $\mathbb{P}$ is the total cost ($\mathbb{S}$) that leaders (edge servers) demand for resources, i.e., $\mathbb{P} = \{p_1, p_2, .., p_s\}$, $\mathbb{B} = \{b_1, b_2, ..., b_s\}$ denotes the price at which the resources are offered by the $\mathbb{S}$ followers. The price of the resources demanded by $\mathbb{S}$ leaders includes, $\forall b_i,\ b_i = (s_i^{cpu}, s_i^{bw}, s_i^{mem})$, a list of cost bid by the leader for CPU, required bandwidth and memory. $\mathbb{L} = \{l_1, l_2, ..., l_s\}$ is the resource demand list by $\mathbb{S}$ leaders, $\forall l_i,\ l_i = \{u_i^{cpu}, u_i^{bw}, u_i^{mem}\}$ is the demand list of CPU, required bandwidth and memory by $\mathbb{S}$ leaders. Now, $\mathbb{A} = \{v_i^{cpu_j^k}, v_i^{bw_j^k} v_i^{mem_j^k}\}$ is the list of available resources at j^{th} server's k^{th} container (j^k). $\mathbb{F} = \{f^{cpu}, f^{bw}, f^{mem}\}$ is the list that stores the requirement and fulfillment relationship between the leaders and followers.

If $f < l$, the demand list of the leaders is greater than the available resources with the followers, then the request is rejected and if $f \geq l$, then the demand can be fulfilled. In this case, the bid manager computed the utility function (U^f) to calculate the cost of all the available edge nodes and thereafter the node with maximum utility that do not deviate from its bid is selected for resource allocation. Bid manager computes the utility for followers and tries to achieve an equilibrium state based on the mathematical proof presented in our previous work [31].

The edge server allocates the resources to the leader if and only if the revenue of the leader is more than the price of the resources at edge nodes. As per the leader, the procedure for resource allocation is defined as follows.

$$P_j = max \sum_{j=1}^{j=s} s_j^{cpu} + s_j^{bw} + s_j^{mem} \tag{28}$$

To allocate the resources and to maximize the P_j/R_j value, considered rate of $u_i^{cpu} \leq v^{cpu_j^k},\ u_i^{bw} \leq v^{bw_j^k},\ u_i^{mem} \leq v^{mem_j^k}$ must be profitable.

Now, with perspective to the followers, to provide the resources to the leaders, the loss (P_j^{lp}) value must be minimal. +-

$$P_j^{lp} = min \sum_{i=1}^{i=s} (s_i^{cpu} - s_{\prime}^{cpu}) + (s_i^{bw} - s_{\prime}^{bw}) + (s_i^{mem} - s_{\prime}^{mem}) \tag{29}$$

where, s_l is the lowest bid price offered by the leader for the resources.

The demand of the resources by the leaders varies with the behavior of the application. To manage the resources efficiently, Stackelberg game model is used to gain the maximum profit for follower and leader.

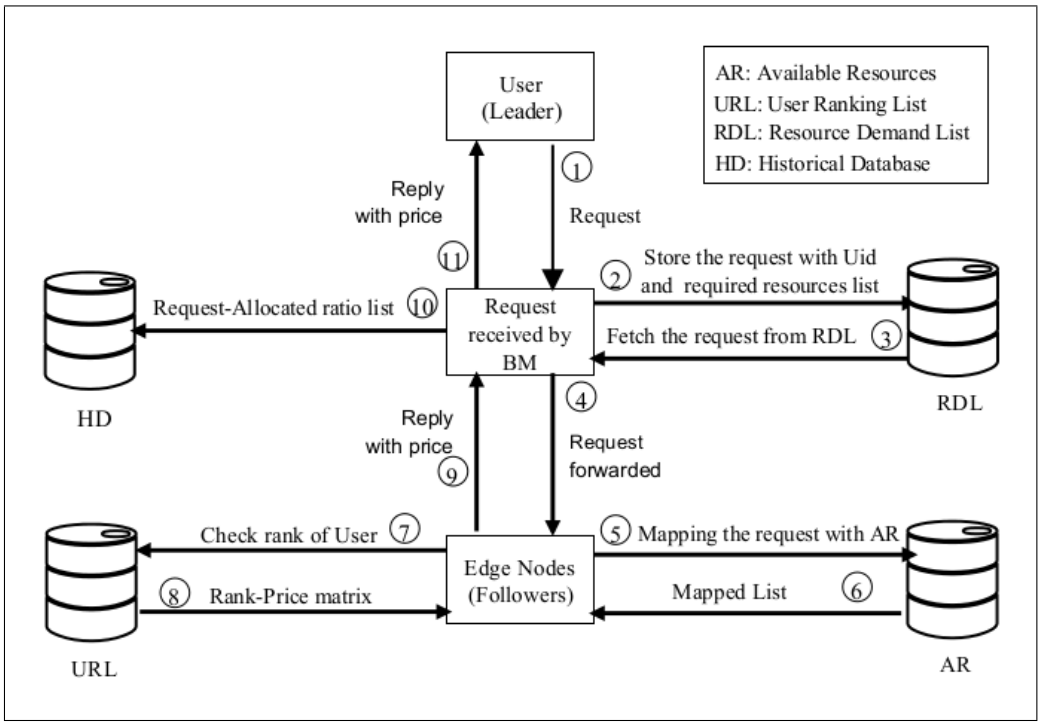

Figure 2: Stackelberg game: User as Leader (UaL)

- **User as Leader (UaL):** At UaL layer, the end users (Leaders) forward the list of required resources to the Resource Demand List (RDL) database with the user-id of the leader. According to the demand of the resources, top leaders are considered as Premium Leaders as shown in Fig. 2. The bid manager (BM) has the list of all the available resources at edge nodes stored with index as Available Resources list (AR). The BM check the rank of leader from User Rank List (URL) to set the priority of the requested resources of the leader to maintain the QoS.

- **Edge as Follower (EaF):** At this layer, the followers receive the demanded price of the resources from the BM from AR to initialize the bargaining process. The premium list forwarded by the Leader's Layer

is stored in Ranked List of Leaders (URL). Therefore, it is the responsibility of the Bid manager to provide the resources to the premium leaders on priority. To ensure resource availability to the premium leaders, an information control strategy is used to allow the premium leaders to bid with the minimum price for the resources. The bid manager maintain the demand fulfillment ratio of the leaders in Historical Database for further references. The followers prioritize the Premium leaders for bidding to allocate the resources with the minimum cost through the Bidding Manager as shown in Fig. 3.

In Historical Database following information is provided to the pre-

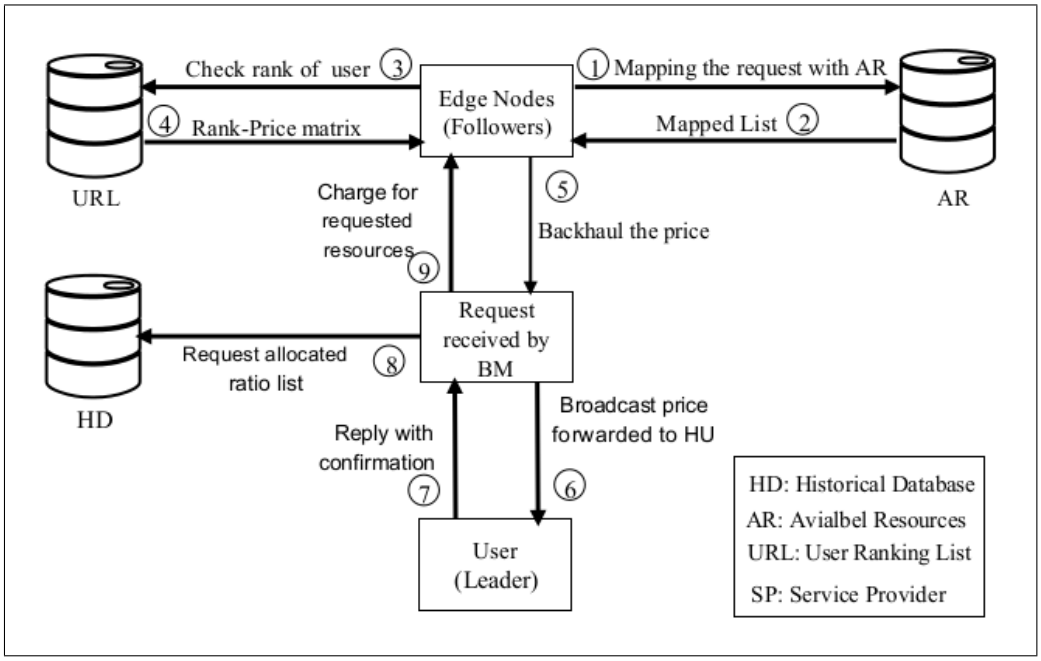

Figure 3: Stackelberg game: Edge as Follower (EaF)

mium leaders,

$$I_f = (\mathbb{H}_{hl}, \mathbb{H}_{hh}, \mathbb{D}_{hd}) \tag{30}$$

where, $\mathbb{H}_{hl}$, is the historical lowest bid price, $\mathbb{H}_{hh}$ is the historical highest bid price and $\mathbb{D}_{hd}$ is the requirement fulfillment ratio of the follower. For normal leaders, a standard information is provided as follows.

$$I_n = (\overline{\mathbb{H}}_{hl}, \overline{\mathbb{H}}_{hh}) \tag{31}$$

where, $\overline{\mathbb{H}}_{hl}$ and $\overline{\mathbb{H}}_{hh}$ is the normal information for other leaders for bidding for the resources.

- **Bidding Layer:** At this layer, the BM collects the bidding information from follower Layer for available resources and bidding rate for availing resources from leaders Layer by using Eq. [30,31]. The winner of the bid is selected and $(S_j^k, \mathbb{T}_i, b_i)$ is forwarded to the follower layer for further reference.

5.2. Container Consolidation and Migration Scheme

Now, the most changeable aspect of the proposed scheme is to minimise the energy consumption based on the load balancing approach using an ensemble of container allocation, consolidation and migration of the active tasks among different containers ($\mathfrak{C}_k$) running at different edge nodes (S_j). The proposed approach tries to distribute the workload on minimum number of containers as to meet the energy constraint. The container consolidation scheme is used to optimize the energy consumption among different $\mathfrak{C}_k j$ service nodes. The work flow of the proposed scheme is shown in Fig. 4.
The various entities related to the container consolidation scheme are described in the following steps.

- ***Load Calculator (LC)***: In this phase, the load of the different containers is calculated by the LC and as per considered threshold values (tH) for the $\mathfrak{C}_k$ of S_j^{th} server, the containers are added to various defined lists (where a is the maximum and b is the minimum ideal threshold value).

- **Overloaded list (OL)**: If $tH > a$, then the particular $\mathfrak{C}_k$ is forwarded to OL.

- **Underloaded List (UL)**: If $tH < a$, then the $\mathfrak{C}_k$ is forwarded to UL.

- **Optimal List (OpL)**: If $b < tH < a$, then forwarded to OpL.

- **Idle List (IL)**: The $\mathfrak{C}_k$ who are deactivated from UL, are shifted to the IL.

- **Container Migrated List (CML)**: The host with $tH > a$ are added into CML, to migrate the load to the other containers to make it ideal one.

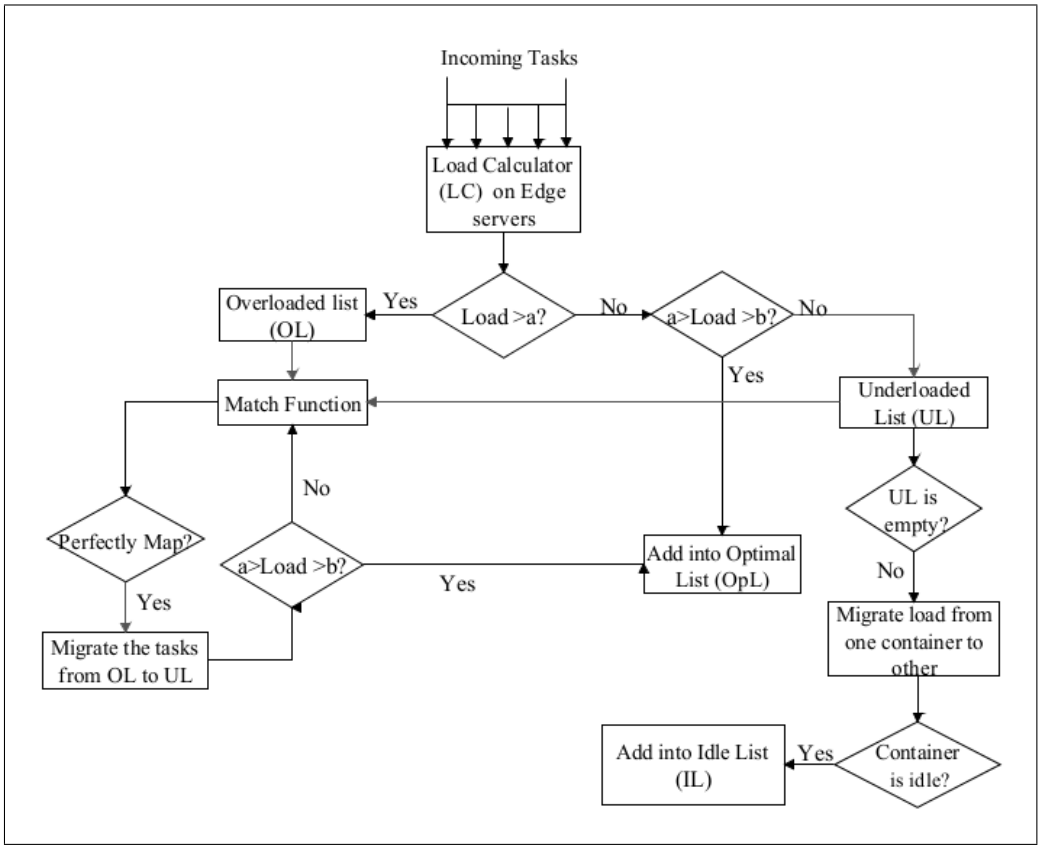

Figure 4: Flowchart of container consolidation scheme

- **Destination Container List (DCL)**: If there is need to migrate the load from OL to UL containers, then this list is mapped with the CML to make it ideal.

In container consolidation scheme, the main focus is to save energy consumed for processing the workload. The overloaded containers consume more energy as compare to ideal containers. So, the workload hanled by the overloaded containers is migrated to the $\mathfrak{C}_k^{UL}$ container, s.t., the required resources must be available at $\mathfrak{C}_k^{UL}$ container. So, all container like $\mathfrak{C}_k^{OL}$ are migrated to $\mathfrak{C}_k^{PL}$ till its threshold values reaches ideal point. In the next cycle, the load of $\mathfrak{C}_k^{UL}$ containers is again verified. If the services of k^{th} container is shifted to $k+n^{th}$ container, then, make sure destination $k+n^{th}$ container load is not greater than ideal threshold value. If source k^{th} container services is nill, then add into Idle List (IL). In the end, update the $\mathbb{R}_{k\in j}^{ava}$ of each list (OL, UL, IL).

The Algorithm I describes the working of the proposed ensemble container management scheme.

Algorithm 1 Container Consolidation Scheme

Input: S_j,$\mathfrak{C}_k$,$\mathbb{R}^{req}$
Output: Destination (S_j^k)

for (j=1;j≤ J;j++) **do**
  for (k=1;k≤ K;k++) **do**
    Compute $\mathbb{U}_k$ using Eq. 21
    if ($\mathbb{U}_j^k > \mathbb{U}_a^{tH}$) **then**
      ADD $\mathfrak{C}_{(k\in j)} \rightarrow OL$ & CML
    else if $\mathbb{U}_j^k < \mathbb{U}_b^{tH}$) **then**
      ADD $\mathfrak{C}_{(k\in j)} \rightarrow UL$ & DCL
      Shift control $\rightarrow CML$
      Mapping of $\mathbb{R}^{req}$: ($CML \iff DCL$)
      if true **then**
        Migrate $CML_k^{res} \rightarrow DCL_k$
      else
        Activate $\mathfrak{C}_{(k\in j)}^{new} \rightarrow IL$
      end if
    else
      ADD $\mathfrak{C}_{(k\in j)} \rightarrow PL$
    end if
    Update $\mathbb{R}_{k\in j}^{ava} \rightarrow (UL, OL, IL)$
  end for
end for

6. Results and Evaluation

To validate the proposed scheme, the experimentation is performed using the CloudSimSDN simulator [34]. The proposed scenario produce more prudent results on SDN enabled network as compare to existing approach. The detailed explanations on simulation settings and obtained results are provided in the subsequent sections.

6.1. Testbed configuration

In the proposed scheme, a tree topology is developed with four hosts and the required switches. One switch act as a core switch, that is further

connected with the two child edge switches. The link configuration of the proposed scheme is shown in Table 3. Using stackelberg game model for

Table 3: Link Configuration

Link	**Bandwidth**
Core $\Longleftrightarrow$ Edge Servers	0.1 Mbps
Edge Servers $\Longleftrightarrow$ Hosts	0.1 Mbps

optimal resource allocation, the price is assigned to each host (Follower) as well as to the VM_s (Leaders). The price of each VM_s is fixed at 1 and this price is negotiable between the leaders and the followers. The price (P) varies according to the requested resources and topology, i.e., ($P \in (1, 2)$). The size of workload and cloudlets in the proposed scheme varies with the defined topology. In total, three scenario's are considered for the validation of the proposed scheme in contrast to the existing approaches. The results are simulated using three type of workload size considering 50 tasks, 57 tasks and 70 tasks with static arrival time and different packet sizes. In the defined scenario's, each host forward different size of the packet data to the other host for processing. The arrival time of all the task is also assumed to be same in the proposed configuration file. The allocation of the VM_s is based on the final auction between the leaders and followers.

6.2. Results validation

In Fig. 5(a), the CPU serve time to process the allocated workload using the proposed scheme is computed on different size of workload to verify the efficiency of the proposed scheme. The lower value of the energy consumption using the proposed scheme is due to the optimal selection of resources using stackelberg game.

In Fig. 5(b), the network serve time obtained using the proposed scheme is analyzed for the defined scenarios. The network serve time of the proposed scheme dominate the serve time of the standard approach due to the support of the underlying SDN architecture used for connecting the geo-distributed edge servers to the proposed approach. The comparison of average total serve time of different workloads is shown in Fig. 5(c). The time consumed

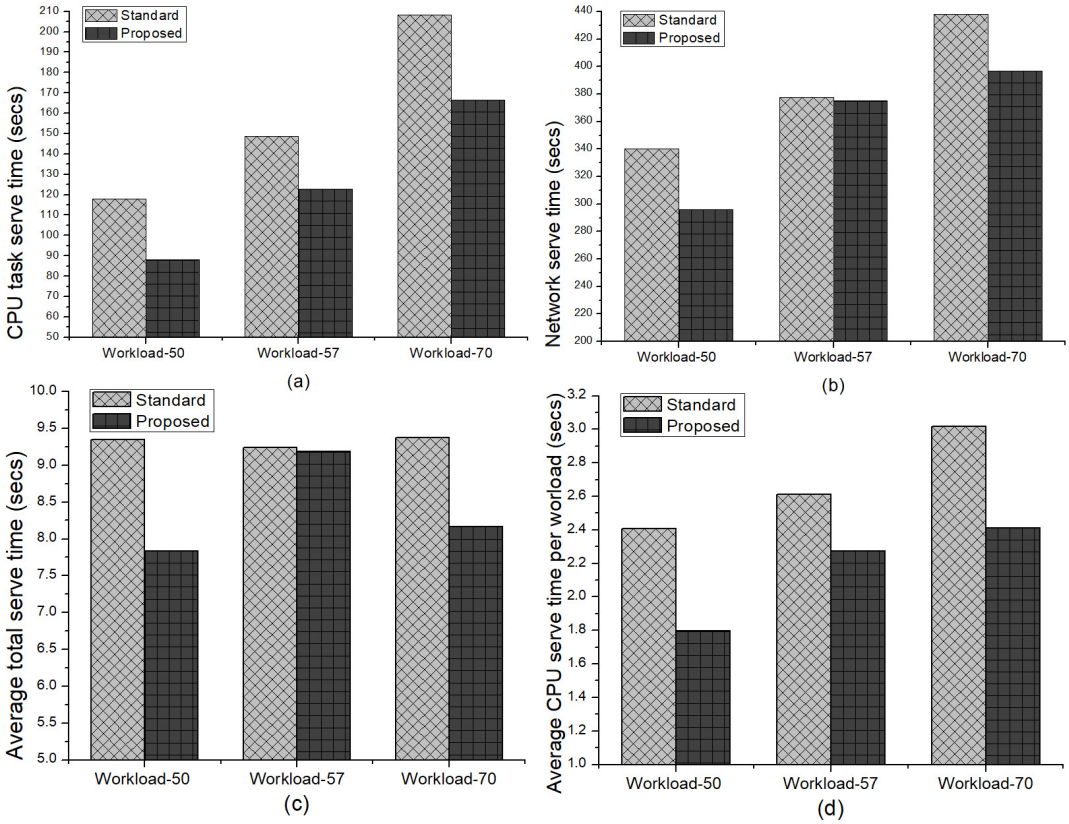

Figure 5: a) CPU task serve time b) Network serve time c) Average total serve time d) Average CPU serve time per workload

to serve the task depends on the size of the packet when transmit from one host to another. The CPU serve time depends upon the size of workload assigned to the $\mathfrak{C}_{k \to j}$ for processing. The average CPU serve time per workload assigned is shown in Fig. 5(d). The processing time of CPU in proposed scheme is better in contrast to the existing variant. The size of the workload directly affects the network time, however the resource allocation policy can reduce the transmission delay in the network as shown in Fig. 6(a). The edge devices define the specifications of the allocated $\mathfrak{C}$ to process the allocated task. The requested cost of the leader depends upon the type of available resources and network traffic. The final resource allocation depends upon the agreement bidding cost between the follower and the leader and accordingly the resources are assigned to the follower. The average CPU serve time

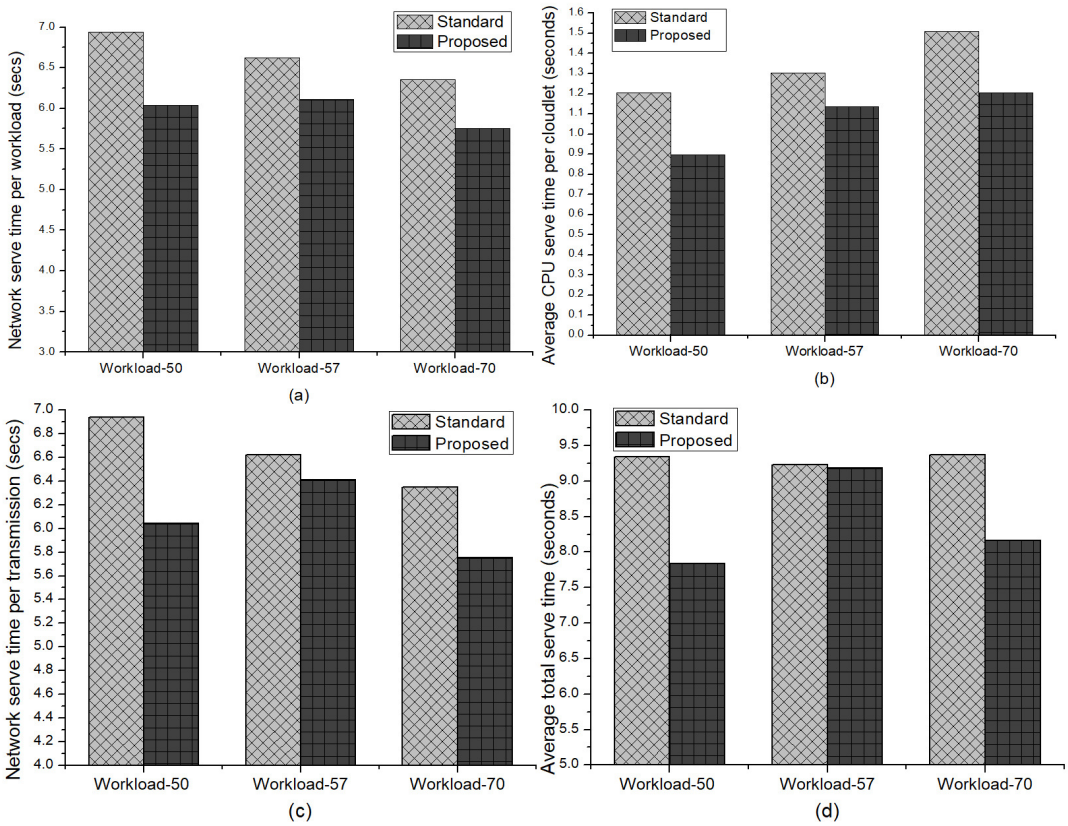

Figure 6: a) Average network serve time per workload b) Average CPU serve time per cloudlet c) Average network serve time per transmission d) Overall total serve time

considering different workloads per edge is shown in Fig. 6(b). During the workload transmission, initially, an input from the V is forwarded to the data plane, then to edge layer, and finally the workload is shifted to the allocated resources for processing. Therefore, a number of transmissions take place to process the allocated task and the average network serve time per transmission is reflected in Fig. 6(c). The total serve time including transmission time, network serve time, CPU server time incurred for handling the workloads is shown in Fig.6(d). Different scenario's are considered for comparison with the existing schemes to justify the proposed scheme.

Energy is one of the major operational cost for processing the allocated tasks from various users. Energy is consumed by the V for collecting the workload and forward the same to the upper layer for further processing.

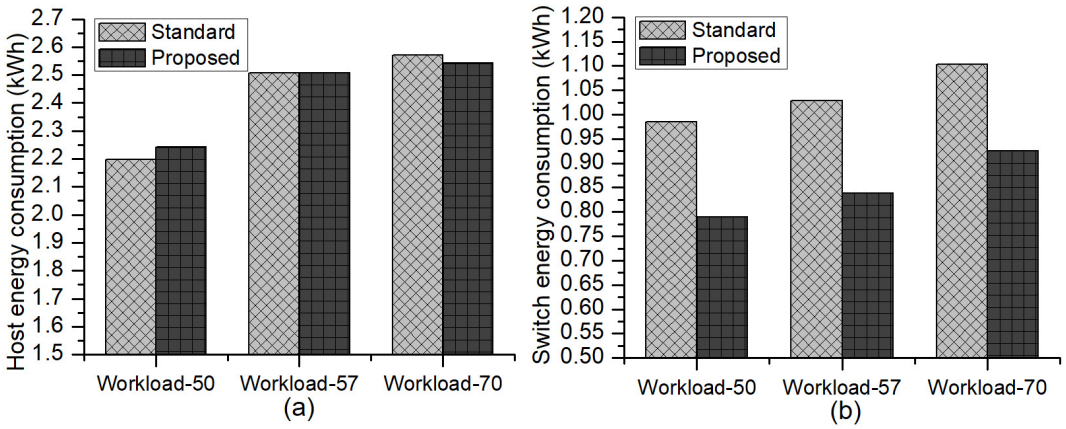

Figure 7: a) Host energy consumption b) Switch energy consumption

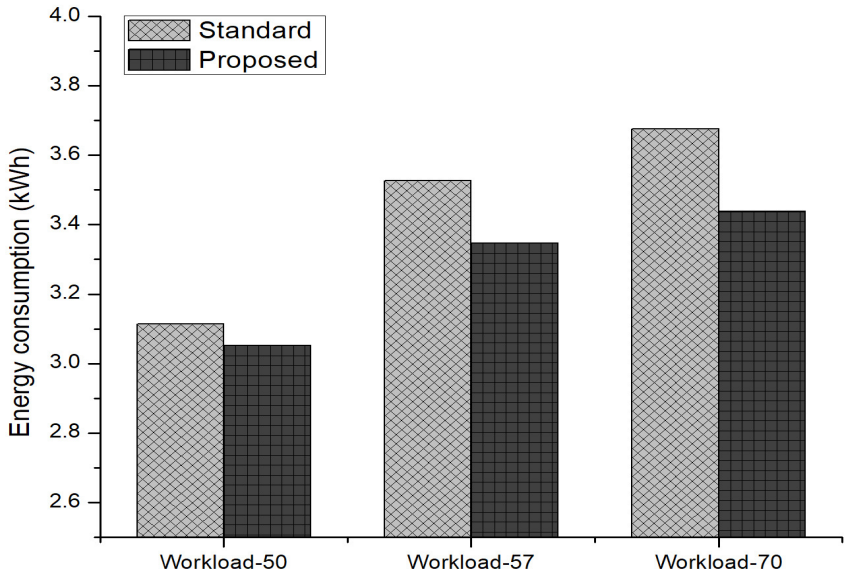

Figure 8: Total energy consumption

The consumption of energy depends upon the behavior of the collected workload and category of V. The energy consumption by the various hosts for

transmission of workload is shown in Fig.7(a). The energy consumption used to process the workloads is compared with the existing approach and highlighted in Fig.7(b). The energy consumption in proposed scheme is lesser as compared to the existing variant. The allocated resources by the Stackelberg game are feasible to process the assigned workload and the rate of migration between nodes is minimal as compare to the existing approach and the same is highlighted in Fig.7(b). The total energy consumption including host and switch is shown in Fig.8). It can be noticed that the energy consumption using the proposed scheme reduces with any change in the size of the workload.

7. Conclusion

In this paper, an energy-efficient resource allocation scheme complimented by an ensemble container selection, consolidation and migration approach is proposed to improvise the QoS in distributed edge computing environment. The IoT workloads are allocated the edge resources for execution of different tasks and analysis. For this purpose, multiple containers are configured on the edge servers to accomplish the request of various users, which in turn reduce overall energy consumption and the latency in the network. The management of the resource distribution mechanism is performed by a multi-leader multi-follower Stackelberg game. The game-based approach is used to resolve the resource pricing problem between the user and edge resource provider. SDN-based architecture is also used to improvise the edge computing through flexible flow policies to manages the network intelligently by configuring a centralized controller. The experimental results supports the proposed energy-efficient resource allocation and optimization technique. The results are obtained in the form of CPU serve time, network serve time, overall delay,and lastly energy consumption. The obtained results clearly depict that they align superior with respect to considered standard approach.

References

[1] J. Wang, Y. Yang, T. Wang, R. S. Sherratt, J. Zhang, Big data service architecture: A survey, Journal of Internet Technology 21 (2020) 393–405.

[2] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, P. Siano, Iot-based smart cities: a survey, in: 2016

IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC), IEEE, 2016, pp. 1–6.

[3] B. Yin, X. Wei, Communication-efficient data aggregation tree construction for complex queries in iot applications, IEEE Internet of Things Journal 6 (2018) 3352–3363.

[4] T. Mastelic, I. Brandic, Recent trends in energy-efficient cloud computing, IEEE Cloud Computing 2 (2015) 40–47.

[5] C. Ju, Y. Gao, A. K. Sangaiah, G.-j. Kim, et al., A pso based energy efficient coverage control algorithm for wireless sensor networks, Computers, Materials & Continua 56 (2018) 433–446.

[6] Q. Tang, K. Yang, P. Li, J. Zhang, Y. Luo, B. Xiong, An energy efficient mcds construction algorithm for wireless sensor networks, EURASIP Journal on Wireless Communications and Networking 2012 (2012) 83.

[7] M. Long, X. Xiao, Outage performance of double-relay cooperative transmission network with energy harvesting, Physical Communication 29 (2018) 261–267.

[8] W. Li, Z. Chen, X. Gao, W. Liu, J. Wang, Multimodel framework for indoor localization under mobile edge computing environment, IEEE Internet of Things Journal 6 (2018) 4844–4853.

[9] Y. Luo, K. Yang, Q. Tang, J. Zhang, P. Li, S. Qiu, An optimal data service providing framework in cloud radio access network, EURASIP Journal on Wireless Communications and Networking 2016 (2016) 1–11.

[10] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, J. González, Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn, Future Generation Computer Systems 111 (2020) 763–779.

[11] B. Xiong, K. Yang, J. Zhao, W. Li, K. Li, Performance evaluation of openflow-based software-defined networks based on queueing model, Computer Networks 102 (2016) 172–185.

[12] C. Iwendi, P. K. R. Maddikunta, T. R. Gadekallu, K. Lakshmanna, A. K. Bashir, M. J. Piran, A metaheuristic optimization approach for

energy efficiency in the iot networks, Software: Practice and Experience (2020).

[13] Q. Ijaz, E.-B. Bourennane, A. K. Bashir, H. Asghar, Revisiting the high-performance reconfigurable computing for future datacenters, Future Internet 12 (2020) 64.

[14] N. M. F. Qureshi, I. F. Siddiqui, A. Abbas, A. K. Bashir, K. Choi, J. Kim, D. R. Shin, Dynamic container-based resource management framework of spark ecosystem, in: 2019 21st International Conference on Advanced Communication Technology (ICACT), IEEE, 2019, pp. 522–526.

[15] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, K. Pentikousis, Energy-efficient cloud computing, The computer journal 53 (2010) 1045–1051.

[16] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment, IEEE network 29 (2015) 56–61.

[17] H. F. Sheikh, I. Ahmad, Z. Wang, S. Ranka, An overview and classification of thermal-aware scheduling techniques for multi-core processing systems, Sustainable Computing: Informatics and Systems 2 (2012) 151–169.

[18] S. U. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Transactions on Parallel and Distributed Systems 20 (2008) 346–360.

[19] H. F. Sheikh, H. Tan, I. Ahmad, S. Ranka, P. Bv, Energy-and performance-aware scheduling of tasks on parallel and distributed systems, ACM Journal on Emerging Technologies in Computing Systems (JETC) 8 (2012) 1–37.

[20] Y. Zhao, R. N. Calheiros, G. Gange, K. Ramamohanarao, R. Buyya, Sla-based resource scheduling for big data analytics as a service in cloud computing environments, in: 2015 44th International Conference on Parallel Processing, IEEE, 2015, pp. 510–519.

[21] Z. Dar, A. Ahmad, F. A. Khan, F. Zeshan, R. Iqbal, H. H. R. Sherazi, A. K. Bashir, A context-aware encryption protocol suite for edge computing-based iot devices, The Journal of Supercomputing (2019) 1–20.

[22] A. Alqahtani, D. N. Jha, P. Patel, E. Solaiman, R. Ranjan, Sla-aware approach for iot workflow activities placement based on collaboration between cloud and edge, in: 1st Workshop on Cyber-Physical Social Systems (CPSS) 2019, Newcastle University, 2019.

[23] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, A. Y. Zomaya, Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method, IEEE Transactions on Industrial Informatics 16 (2020) 6103–6113.

[24] H. Liao, Z. Zhou, X. Zhao, L. Zhang, S. Mumtaz, A. Jolfaei, S. H. Ahmed, A. K. Bashir, Learning-based context-aware resource allocation for edge-computing-empowered industrial iot, IEEE Internet of Things Journal 7 (2019) 4260–4277.

[25] S. K. Mishra, D. Puthal, B. Sahoo, P. P. Jayaraman, S. Jun, A. Y. Zomaya, R. Ranjan, Energy-efficient vm-placement in cloud data center, Sustainable computing: informatics and systems 20 (2018) 48–55.

[26] Y. Chen, X. Chen, W. Liu, Y. Zhou, A. Y. Zomaya, R. Ranjan, S. Hu, Stochastic scheduling for variation-aware virtual machine placement in a cloud computing cps, Future Generation Computer Systems 105 (2020) 779–788.

[27] N. Kumar, G. S. Aujla, S. Garg, K. Kaur, R. Ranjan, S. K. Garg, Renewable energy-based multi-indexed job classification and container management scheme for sustainability of cloud data centers, IEEE Transactions on Industrial Informatics 15 (2018) 2947–2957.

[28] G. S. Aujla, N. Kumar, Sdn-based energy management scheme for sustainability of data centers: An analysis on renewable energy sources and electric vehicles participation, Journal of Parallel and Distributed Computing 117 (2018) 228–245.

[29] K. Kaur, S. Garg, N. Kumar, G. S. Aujla, K.-K. R. Choo, M. S. Obaidat, An adaptive grid frequency support mechanism for energy management in cloud data centers, IEEE Systems Journal (2019).

[30] G. S. Aujla, N. Kumar, A. Y. Zomaya, R. Ranjan, Optimal decision making for big data processing at edge-cloud environment: An sdn perspective, IEEE Transactions on Industrial Informatics 14 (2017) 778–789.

[31] G. S. Aujla, M. Singh, N. Kumar, A. Zomaya, Stackelberg game for energy-aware resource allocation to sustain data centers using res, IEEE Transactions on Cloud Computing (2017).

[32] G. S. Aujla, N. Kumar, Mensus: An efficient scheme for energy management with sustainability of cloud data centers in edge–cloud environment, Future Generation Computer Systems 86 (2018) 1279–1300.

[33] G. S. Aujla, A. Singh, M. Singh, S. Sharma, N. Kumar, K.-K. R. Choo, Blocked: Blockchain-based secure data processing framework in edge envisioned v2x environment, IEEE Transactions on Vehicular Technology (2020).

[34] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, R. Buyya, Cloudsimsdn: Modeling and simulation of software-defined cloud data centers, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2015, pp. 475–484.