CONTAINER-AS-A-SERVICE AT THE EDGE: TRADE-OFF BETWEEN ENERGY EFFICIENCY AND SERVICE AVAILABILITY AT FOG NANO DATA CENTERS

KULJEET KAUR, TANYA DHAND, NEERAJ KUMAR, AND SHERALI ZEADALLY

ABSTRACT

In the last few years, we have witnessed the huge popularity of one of the most promising technologies of the modern era: the Internet of Things. In IoT, various smart objects (smart sensors, embedded devices, PDAs, and smartphones) share their data with one another irrespective of their geographical locations using the Internet. The amount of data generated by these connected smart objects will be on the order of zettabytes in the coming years. This huge amount of data creates challenges with respect to storage and analytics given the resource constraints of these smart devices. Additionally, to process the large volume of information generated, the traditional cloud-based infrastructure may lead to long response time and higher bandwidth consumption. To cope up with these challenges, a new powerful technology, edge computing, promises to support data processing and service availability to end users at the edge of the network. However, the integration of IoT and edge computing is still in its infancy. Task scheduling will play a pivotal role in this integrated architecture. To handle all the above mentioned issues, we present a novel architecture for task selection and scheduling at the edge of the network using container-as-a-service (CoaaS). We solve the problem of task selection and scheduling by using cooperative game theory. For this purpose, we developed a multi-objective function in order to reduce the energy consumption and makespan by considering different constraints such as memory, CPU, and the user's budget. We also present a real-time internal and external container migration technique for minimizing the energy consumption. For task selection and scheduling, we have used lightweight containers instead of the conventional virtual machines to reduce the overhead and response time as well as the overall energy consumption of fog devices, that is, nano data centers (nDCs). Our empirical results demonstrate that the proposed scheme reduces the energy consumption and the average number of SLA violations by 21.75 and 11.82 percent, respectively.

Kuljeet Kaur, Tanya Dhand, and Neeraj Kumar are with Thapar University.

Sherali Zeadally is with the University of Kentucky.

Digital Object Identifier:
10.1109/MWC.2017.1600427

INTRODUCTION

Rapid advancements in information and communications technology (ICT) along with an exponential increase in smart devices in the last few decades have led to the emergence of one of the most popular technologies, the Internet of Things (IoT). IoT consists of billions of interconnected devices for providing seamless services such as smart transportation, e-healthcare, smart education, and smart sensing to end users. Consequently, bulk data generation is inevitable from such a huge interconnection of heterogeneous and geographically distributed devices. This is evident from statistics given in [1], which predict that almost 50 billion devices will be interconnected to the Internet by 2020. The current number of connected devices is already greater than the world population and is exponentially increasing, as shown in Fig. 1a. The huge amounts of data that will be produced by all these interconnected objects will need to be handled efficiently. Moreover, these large numbers of interconnections would also be associated with real-time network traffic in order to access different services. This would also require seamless computation and processing of the data collected from various devices with heterogeneous capabilities.

INTEGRATION CHALLENGES OF IOT AND CLOUD COMPUTING

From the above discussion, we note that the transmission of large amounts of data to the core cloud computing platform requires context-aware services to be available to end users. For instance, owners of high-speed vehicles may be interested in knowing the "best" route (e.g., the one with the least traffic) to their destination. In contrast, other users might be interested in managing their home appliances remotely based on the current demand-response mechanism.

In short, the integration of IoT with the cloud computing platform is necessary for efficient processing of stored data at the cloud repository. The cloud data centers are robust with respect to handling a diverse range of computations for various applications. However, the transfer of large

1536-1284/17/$25.00 © 2017 IEEE

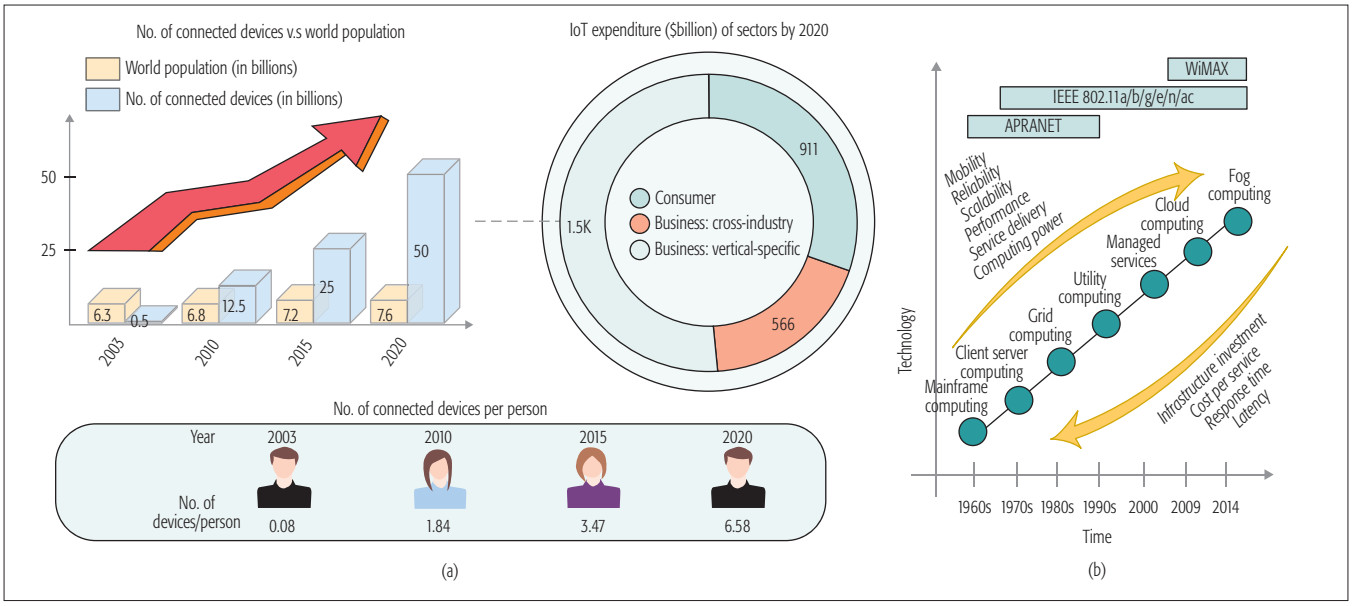

Figure 1. Edge computing revolution: a) number of connected devices vs. the world population; b) evolution of edge computing.

chunks of data to the remote cloud platform from the mobile nodes needs higher bandwidth, which in turn may cause higher latency and network overhead. This in turn poses additional challenges such as achieving lower response time and high service availability [2, 3]. Moreover, the mobility of users exacerbates these problems further, particularly for delay-sensitive applications. For instance, according to a recent survey conducted by Gartner, nearly a quarter billion vehicles will be connected to the Internet by 2020. These vehicles are expected to become an important component of the future IoT-based infrastructure for providing in-vehicle services and automated driving capabilities. These statistics indicate that the number of consumers and the data they generate are expected to increase exponentially in the future. Therefore, there is a need for a new technology that can achieve fast response times for executing various jobs, even when the connected nodes/smart gadgets (vehicles, mobile sensors, moving objects, etc.) are highly mobile. In such an environment, fog computing can provide support to real-time delay-sensitive applications to end users. Figure 1b depicts the evolution of various technologies in the last few decades. As the figure shows, fog computing seems to be a promising technology to address the challenges we have identified earlier. Its significant advantages over existing technologies such as cloud computing, managed services, utility, grid, client-server, and mainframe computing are also shown in the figure. For instance, various performance metrics such as computing power, service delivery, performance, reliability, scalability, and mobility have shown significant improvements in this context with recent technological advances. In contrast, response time, cost per service, infrastructure investment, and latency have gradually reduced with increasing technological advancements.

The Evolution of Edge/Fog Computing

To enable seamless, real-time data processing, the concept of edge/fog computing [2] was proposed by Cisco in 2014. This concept extends the notion of cloud computing to the customer's premises, wherein the computing, networking, and storage requirements would be met at the edge of the network. It also deals with the processing of the workload on the fog devices (placed at the edge of the network) instead of relying on the core of the cloud platform to perform the required processing. Due to this reason, edge computing has emerged as a promising solution for enabling the seamless processing of data in the proximity of the user in real time. Edge computing is also often referred to as "cloud close to the ground" [4].

Edge Computing vs. Cloud Computing

The distinctive characteristics of edge computing over conventional cloud computing (based on a centralized approach) have increased its popularity among researchers and academicians. In comparison to cloud computing, edge computing is based on the concept of localization to enable fast processing and reduce latency. In addition, the fog devices used in edge computing are widely deployed at locations (e.g., public libraries, shopping malls, parking lots, and highways) that are close to users. In contrast, the core computational units used in cloud computing (i.e., data centers, DCs) are not as distributed and are often found at locations far from the user, resulting in relatively slower response times. Edge computing serves various services at the edge of the network using fog hardware devices such as nano-DCs (nDCs). Unlike conventional DCs, nDCs are characterized by relatively low computational and storage capabilities. These nDCs consist of computing devices including-PCs, tablets, laptops, wearable computers, smart displays, and so on. To support real-time data efficiently, there is a need to schedule the various tasks in an optimal manner in order to meet various SLA parameters such as response time, service availability, latency, and costs. In order to address these challenges, in this article, we propose container-as-a-service (CoaaS) to achieve task selection and scheduling at the edge of the network. In comparison to virtual machines (VMs), containers are relatively

The gateway provides a channel to handle data communication between an external network and the core network. Hence, the edge gateways relay the heterogeneous data and user requests to the second layer of the system for further processing. In short, this layer is only responsible for aggregating and redirecting the user task requests to the CoESMS.

lightweight virtualization instances [5, 6]. Within VM, many containers can exist for managing task scheduling and load balancing in a virtualized environment. Hence, their scheduling and migration are considered more appropriate than conventional VMs in terms of reduced network bandwidth utilization and energy consumption.

Research Contributions of This Work

We summarize the major contributions of this work as follows.

- We present a novel architecture called "container-based edge service management system (CoESMS)" that leverages the advantages of IoT and an edge computing platform for real-time data processing. It takes the benefits of the popular CoaaS for enhancing the performance and scalability of the fog computing devices.
- We propose CoaaS for energy-efficient task selection and scheduling at the edge of the network in order to meet users' SLA requirements. Task selection and scheduling are achieved using cooperative games among brokers and among containers. Using these two cooperative games, the objectives with respect to nDC's energy consumption and tasks' makespan (the maximum time it takes for a task to complete) are achieved.
- We present a performance evaluation of our proposed scheme and demonstrate its superior performance (using various performance metrics) over other existing approaches.

Organization

The rest of the article is structured as follows. The following section presents our proposed architecture and its layers. Next, CoaaS for energy-efficient task scheduling is explored. We then present the experimental performance evaluation results. Finally, we conclude the article.

Container-Based Edge Service Management System

This section presents the detailed architectural diagram used in the proposed CoESMS.

Figure 2 shows the architecture of CoESMS consisting of three main entities: resource consumers (*C*), utility provider (*P*), and suppliers (*S*). It also consists of five different layers, which we describe as follows.

Layer 1: resource consumer laer (RCL) } *C*

Layer 2: requirement collector layer (RCol)
Layer 3: controller layer (CL) } *P*
Layer 4: broker layer (BL)

Layer 5: computation layer (Col) } *S*

We describe each layer below.

Layer 1: Resource Consumer Layer

This layer comprises the fog resource consumers such as IT clients and end users. These consumers initially access the smart devices through a local area network (LAN) and submit their service requests via the edge gateway. The gateway provides a channel to handle data communication between an xternal network and the core network. Hence, the edge gateways relay the heterogeneous data and user requests to the second layer of the system for further processing. In short, this layer is only responsible for aggregating and redirecting the user task requests to the CoESMS.

Layer 2: Requirement Collector Layer

This is the second layer of the proposed system and comprises two modules: the task requirement module (TRM) and the user task request module (UTRM). Using these two modules, the tasks that would be executed at the edge are selected, while the rest of the tasks are forwarded to the core of the network. The main functionality of the TRM is to evaluate the task's request requirements with respect to the various parameters for further processing. These parameters include task type, task size, budget, and time. The parameter "task type" can either be computation-intensive, storage-intensive or network-intensive task. Hence, the proposed fog platform provides services to the user only if the task type does not fall within this classification. Otherwise, tasks are redirected to the core of the network.

The next parameter is the "task size" and is classified into four categories. The first category represents tasks that have a low requirement for CPU time and memory; the second category denotes tasks with intermediate CPU and low memory requirement; the third category comprises tasks with intermediate CPU and high memory requirement; and the fourth category represents tasks that require high CPU usage and high memory requirement. The other two parameters, budget and time, are also considered as dynamic parameters. Their values vary in predefined ranges according to the service level agreement (SLA) between the user and the utility provider. Based on the above classification, each task request is associated with the above specified parameters. The corresponding configuration information is then stored using a heap data structure in the form of an application form. The proposed work is done by the user requirement module (URM). Additionally, the URM associates the user defined task requests with the required number of containers based on each one's configuration information. The module also assigns each user request with a unique request id. This identifier helps to schedule the task on the required containers and helps to track the process. After this, the generated application form is forwarded to the next layer.

Layer 3: Controller Layer

This layer is an important part of the proposed scheme and primarily controls the task scheduling process. We describe this layer in detail in the next section. This layer comprises two modules: the resource monitoring module (RMM) and the container scheduling module (CSM). The RMM continuously keeps track of the idle containers that are available for task scheduling. Based on the information received from this module, the CSM schedules the tasks on the containers in an energy optimal manner using the proposed game theoretical framework, which we discuss in the next section.

Layer 4: Broker Layer

This layer acts as an intermediate entity between layers 3 and 5. The interaction between these layers is extensively used in the task scheduling

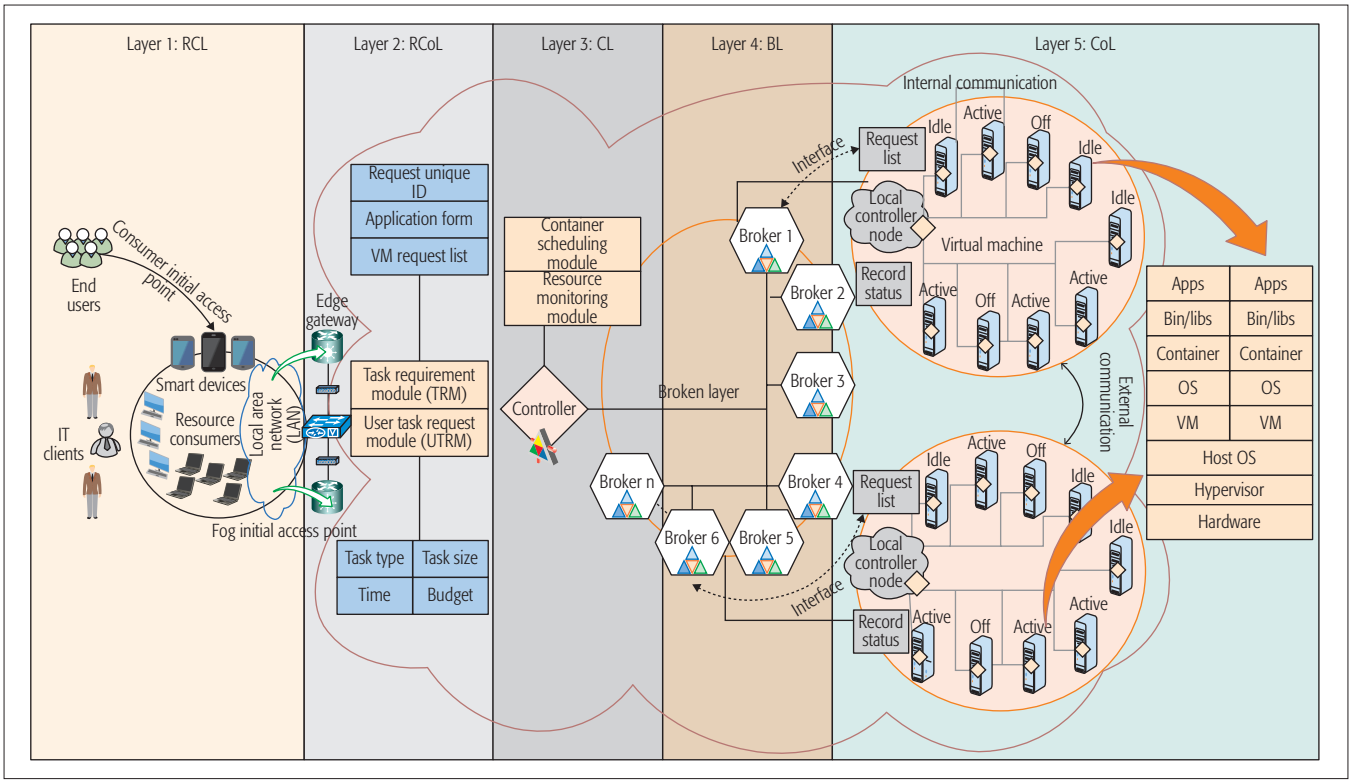

FIGURE 2. Proposed architectural diagram of the conatiner-based edge service management system (CoESMS).

process. This layer is designed to select, rank, and reserve resource combinations in the form of containers. This is achieved with the help of the broker that belongs to the fog computing platform. The primary responsibility of this fog broker is to handle the user services, keep track of available VMs and containers, and delegate the requests to the containers of VMs.

Layer 5: Computation Layer

The last layer comprises the nDCs that actually execute the tasks using the lightweight containers. It also incorporates a node-level local controller which keeps track of all containers at the nDC level and their current status. Depending on their current status, the controller selects the containers that can be allocated to different tasks. Additionally, it also performs container migrations within and between the VMs. These migrations are carried out for reducing the overall energy consumption of the nDCs during unexpected failures

CoaaS for Energy-Efficient Task Scheduling at the Edge

CoaaS is an emerging technique that allows task scheduling on lightweight containers and supports their migration whenever required. It can also be viewed as an alternative solution to hypervisor-based virtualization, which is inadequate to execute time-critical tasks. The latter also lacks the capability to reallocate runtime resources. At present, four major virtualization technologies are available in the literature; these include para, full, operating system (OS) level, and native virtualization. Para-virtualization was first introduced by the Xen Project team and is considered as a lightweight virtualization technique. It enables hardware virtualization and does not rely on the virtual extensions of the host machine. Full virtualization provides a VM environment that fully simulates the host's underlying hardware. Native virtualization partially simulates the hardware to run the full-fledged OS. The OS-level virtualization is also known as container-based virtualization and is the most popular one among all the virtualization techniques [7].

In this work, we focus on container-based virtualization because it helps to deploy the tasks over the containers. Moreover, it also supports lightweight migration within and among VMs. This, in turn, helps in live system updates and eliminates the need for hardware emulation by using the underlying host OS. It should be noted that all the containers inside VMs share a single OS kernel. The main advantages of using containers over VMs include light weight, increased performance, higher efficiency, and no need for privilege instruction trapping. In short, containers can be executed at a higher speed than VMs and can be used for real-time data collection from end users. Containers also share kernel instances, OS, and network connections. The VMs, on the other hand, have their own virtualized network, BIOS, CPU, and OS. Additionally, containers enable a higher degree of application execution using fewer OSs in comparison with VMs. Table 1 highlights the relative differences between containers and VMs.

Container scheduling and migration as proposed in this work can be best understood by using Fig. 3, which shows that the container scheduling and migration are managed automatically by the docker (a software framework built around the container engine). The docker contains all the components required for runtime task execution such as configuration files and databas-

The simplest approach to trigger the container migration(s) is on the basis of energy usage of the servers. Hence, the migration would be initiated whenever the utilization of the server exceeds or falls behind the predefined upper and lower threshold limits, respectively. These migrations can be either within (internal) or between the VMs (external).

Parameter	Containers	Virtual machines
Type of virtualization	Based on OS-level virtualization where the virtual layer acts as an application above the OS.	Part of completely virtualized environment which abstracts only the underlying physical hardware.
Degree of security and quality assurance	High risks involved due to direct deployment of containers over OS.	Relatively low risks are involved if the containers are deployed inside VMs.
Type of resource sharing	Containers share kernel instances, OS, bare file system and network connection.	VMs have their own virtualized network, BIOS, CPU, OS and disk storage.
Performance	Enable maximum degree of application execution using the least number of servers.	VMs can execute a higher number of applications with a variety of OS.
Type of user space	Separate user space to run each application instance.	VMs contain common user space for each application instance.
Kernel state	Require booting for running OS and application loading.	Require application loading only since the kernel is already operational.
Start and stop time	Start and stop time of docker container is less than 50 ms.	Start and stop time of VM is 30-40 and 5-10 secs respectively.
Degree of overhead	Low startup overhead for launching containers.	Relatively high overhead for launching VM.
Memory and CPU usage	Lesser CPU and memory usage per container instance.	Large memory and CPU usage per VM instance.
Migration and management	High migration speed and management as they-support lightweight package.	Low migration speed and slower management because they aredone on a large scale.
Cost	Cost overheads are lower because of lower operational and hardware investment.	VMs are typically heavyweight which incur high cost overheads.
Need of guest OS	No need to have a separate guest OS because each instance of application shares a common kernel and host OS.	VMs allow to share single hardware with multiple guest OS.
Tools	Various container virtualization tools are OpenVZ, docker, Sandboxie, Warden/Garden.	Various hyper-virtualization tools are Virtual PC, Virtual Box, WM-Ware, Xen, KVM.
Standardization	Not well standardized and involves complexity in OS and kernel specification.	Well standardized system having the same capabilities same as the bare-metal system.

Table 1. Comparison between containers and virtual machines (VMs).

es. It enables an additional layer of abstraction and virtualization. It is particularly used for Linux OS and leverages the resource isolation feature of the underlying kernel. This, in turn, permits containers to be independently executed within the kernel instance, thereby eliminating the need to initiate and maintain VMs [7].

Figure 3 illustrates task scheduling using containers. In the proposed scheme, only L number of containers have been considered on two VMs. The docker engine helps schedule the tasks over the portable containers. The scheduled containers then execute the tasks. After task execution, the containers are freed. The docker also handles container-to-container migration for various scenarios such as task failure, downtime, and energy management. For instance, the simplest approach to trigger the container migration(s) is on the basis of energy usage of the servers. Hence, the migration would be initiated whenever the utilization of the server exceeds or falls behind the predefined upper and lower threshold limits, respectively. These migrations can be either within (internal) or between the VMs (external), as shown in Fig. 3.

Operation of CoESMS: Task Selection and Task Scheduling

This section describes the operation of the proposed CoESMS. The major emphasis of CoESMS is on energy consumption reduction while meeting users' requirements for accessing various services such as traffic information, parking information, and infotainment services. It aims to provide an efficient task scheduling management system while satisfying the increasing user demand for data access and addressing the constraints of the traditional cloud computing platform.

The overall operation of CoESMS is shown in Fig. 4a. The TRM module of layer 2 validates the different tasks according to their respective task types as specified above. Based on this evaluation, the tasks are either processed at the edge or at the core of the network. Moreover, layer 2 generates the configuration list of the requested tasks to be executed at the edge using the heap data structure.

This user's requests application form is sent to the controller of layer 3. The CSM module then validates the number of containers (η_i) required

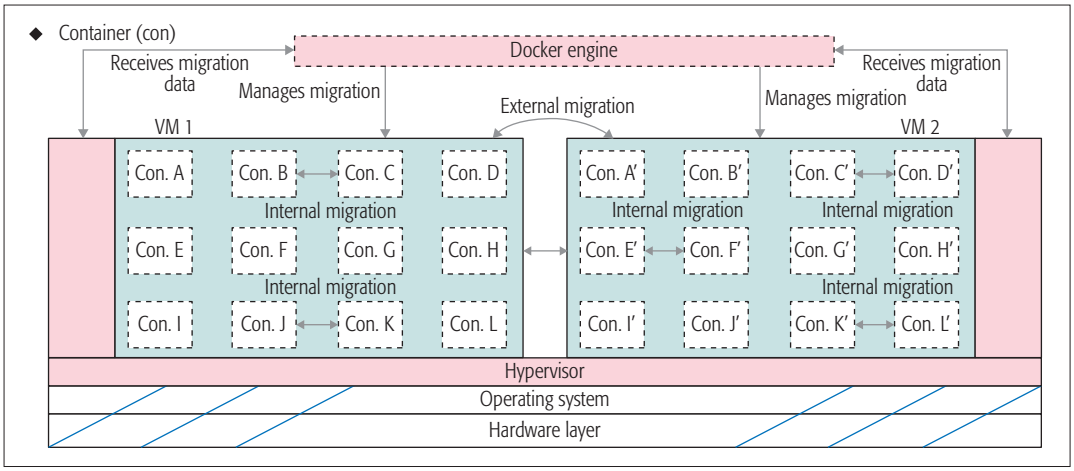

FIGURE 3. Task scheduling at the container level.

by each *i*th task. This η_i is then compared to the available number of containers (α) as reported by the RMM. If the number of containers required by the task is less than or equal to the number of available containers, the task is scheduled for container allocation and is passed on to the next layer. Otherwise, the task is not configured for allocation at the edge, and in this case the task is sent to the core for further processing.

In the next phase, the CSM selects the broker *j* among *N* available brokers as depicted in Fig. 2. This selection of the broker for further task processing is formulated as a cooperative game theory [8–10]. In the proposed scheme, we have considered an *n*-player game, where each player (i.e., broker) attempts to reduce the overall communication cost based on the current bandwidth (B_j) and the load status (W_j). The utility function of brokers (μ_j) is formulated using weighted contributions of these two parameters, that is, B_j and W_j, as follows:

$$\mu_j = \frac{1}{N} \times \frac{\alpha . B_j}{\beta . W_j \,/\, W_j^{\max}}$$
$$s.t. \quad \alpha, \beta \in [0,1]$$

where $W_j^{\max}$ denotes the maximum load capacity of the *j*th broker, and the parameters α and β refer to weights for B_j and W_j, respectively. The higher the value of μ_j, the higher are the chances of selecting the *j*th broker and vice versa. Moreover, the selected broker sends the configuration details of the task *i* under consideration to the local controller of layer 5 for final task scheduling. The local controller then maps the tasks to the available containers across different VMs based on various configuration parameters (e.g., task type, CPU time, budget, and memory). The mapping of the tasks to the appropriate containers can be viewed as a multi-objective cooperative game and is defined as follows.

Definition 1: Suppose we have a total of *M* task requests that need be to mapped to *P* containers across *Q* VMs available at the nDC level. The makespan of the *i*th task request is represented as A_i, $i \in [1, M]$ and refers to its maximum completion time. The primary objective of the multi-objective cooperative game theoretical model is to schedule the set of task requests to the *k*th container(s) such that the total energy consumption of the nDC and makespan of the tasks ($F(x)$) are minimized while satisfying the following set of constraints:

Minimize $F(x) = f(E(x), A(x))$;
s.t. $c_i(x) \le \beta_{x,k}$, $\forall i, k$,
$m_i(x) \le \gamma_{x,k}$, $\forall i, k$,
$b_i(x) \le \Lambda_{x,k}$, $\forall i, k$,
$x \in S$

where $F(x)$ is the objective function as per the solution *x* among the available set of feasible solutions (S). Here, the parameter S denotes the available search space, which constitutes the feasible solution according to the set of constraints taken. $E(x)$ and $A(x)$ represent the minimization functions corresponding to the nDC's energy consumption and task execution time, respectively. The constraint parameters $c_i(x)$, $m_i(x)$, and $b_i(x)$ correspond to the *i*th task's CPU, memory, and budget requirements, respectively. In addition, $\beta_{x,k}$, $\gamma_{x,k}$, and $\Lambda_{x,k}$ represent the limits on CPU, memory, and budget with respect to resource usage function on the *k*th container for a given task assignment, respectively.

The multi-objective scheduling scheme discussed above is formulated using cooperative game theory. However, due to the complexity of this problem, it is further formulated as a sequential cooperation game. Hence, the major parameters of this game are elaborated as follows:

- The set *P* containers across the nDC act as players in the cooperation game.
- The utility function of the competing players is computed as follows:

$$\mu_k = \arg\left(\min\left(E(x), A(x)\right)\right)$$

- The set of strategies (Δ and β) adopted by the players is determined by the following set of constraints:

$c_i(x) \le \beta_{x,k}$, $\forall i, k$
$m_i(x) \le \gamma_{x,k}$, $\forall i, k$
$b_i(x) \le \Lambda_{x,k}$, $\forall i, k$

Here, strategy Δ refers to the task distribution matrix with respect to different VMs and containers. Strategy β represents the CPU allocation matrix. The objectives and the above mentioned constraints are defined accordingly. We describe them in Fig. 4b. Here, the parameters E_{act} and E_{idle} refer to the total energy consumption of nDC when it is active and idle, respectively. Similarly, P_{act}, P_{idle}, T_{act}, and T_{idle} refer to the nDC's power

If the number of containers required by the task is less than or equal to the number of available containers, then the task is scheduled for container allocation and is passed on to the next layer. Otherwise, the task is not configured for allocation at the edge and in this case the task is sent to the core for further processing.

consumption and related time slots. Other parameters are container-specific and refer to the total number of tasks mapped, number of processors assigned, data size of tasks mapped, processing rate, task's processing time, and task's cost. These are represented using parameters δ_k, θ_k, d_k, β_k, p_k, and c_k, respectively.

- Each player aims to minimize the multi-objective function $F(x)$ in the multi-constraint environment.

In cooperative games, the players in a group coordinate their moves and actions while leveraging the advantages of some transferable utility. This means that the players with a greater utility function can coordinate with the players with a lower utility function to achieve optimal results. The most popular approach to obtain the optimal value of a defined function in this case is by using the Lagrangian equation. However, due to the complexity involved, it is difficult to reach its optimal solution. Hence, in order to deduce the results of the formulated game, we express it as a sequential cooperation game in which the participating entities, called the players, initially start by selecting a predefined strategy and observe the actions of the preceding players. Hence, the intermediate solutions across different stages adhere to the equations as specified in Fig. 4b. The symbol, S, as given in the equations, denotes the set of stages in the game, and each player uses a set of strategies based on the task and CPU allocation in the previous stages. It is important to mention here that the Nash equilibrium for the proposed scheme can be achieved as described in [11].

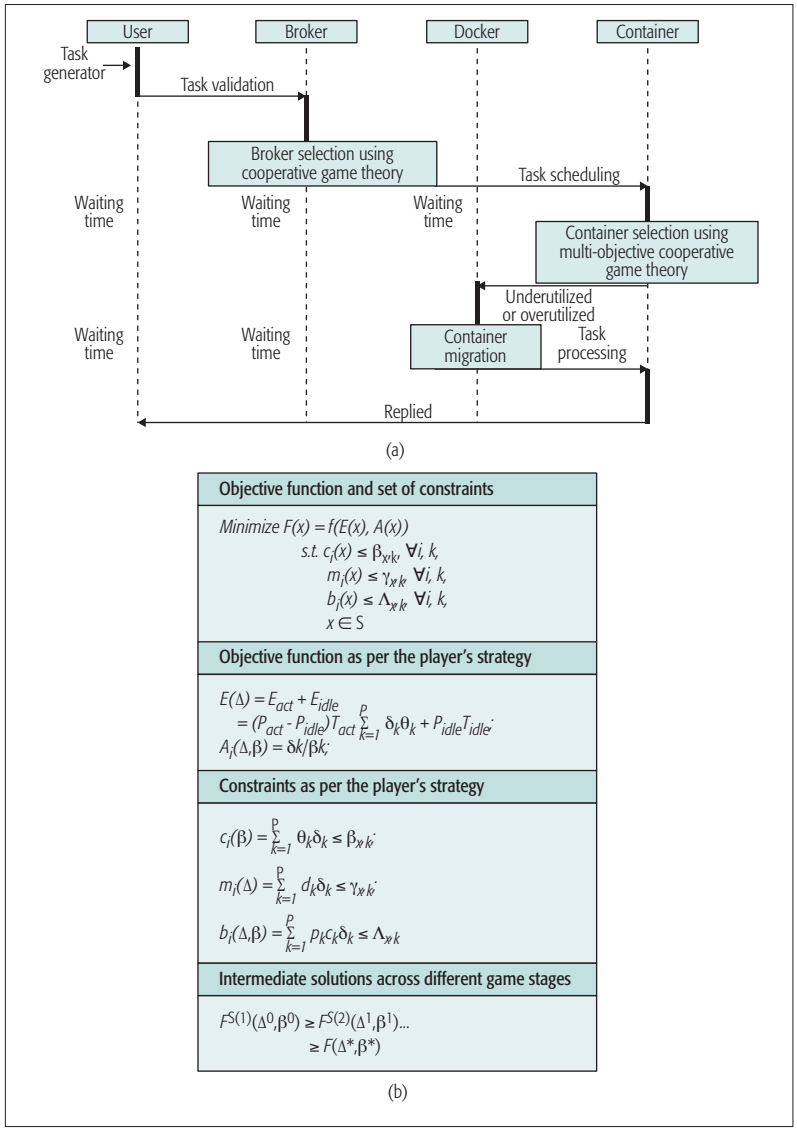

FIGURE 4. Detailed operation of CoESMS: a) sequence diagram for task selection and scheduling using CoESMS; b) the multi-objective game theoretical approach for task scheduling.

PERFORMANCE EVALUATION

SIMULATION SETUP

This section investigates the impact of the proposed CoESMS on the overall energy utilization at the edge of the network, while ensuring adherence to the SLA. It is implemented using an extended version of the popular cloud simulator CloudSim, ContainerCloudSim [12]. It provides a containerized environment with respect to cloud computing for modeling and simulation. This is done by comparing the proposed scheme with an existing scheme that does not implement CoESMS. This operation of the existing scheme can be summarized as follows. It selects and schedules the tasks on available containers on a first-come first-serve basis. Additionally, the broker selection is relatively simple: a particular broker is selected based on its availability and its current workload, in contrast to our proposed scheme, which is based on the game theoretical concept.

The experimental simulation tests were repeated 25 times across a 24-hour simulation period. The related simulation parameters with respect to nDCs, VMs, and containers are summarized in Table 2. A total of 200 nDCs, 450 VMs, and 750 containers were used to perform extensive simulations. Additionally, four different types of VMs and three types of containers were also considered, wherein each VM has different CPU and memory configurations, as shown in Table 2.

To evaluate the two schemes based on the above simulation setup, workload traces from PlanetLab were used [13]. The performance of the two schemes were compared using the following four performance metrics.

Total Number of Container Migrations: This metric evaluates the total number of internal and external container migrations so as to minimize the total energy utilization of the servers available at the edge of the network. A simple threshold-based mechanism is considered for this purpose where the utilization levels of servers is above 80 percent or below 50 percent (if utilization is below 50 percent, migration is also done in the proposed scheme).

Container Overhead Analysis: Although containers are lightweight entities, they do have associated overheads, especially during their migrations. This overhead is due to the container startup time and number of migrations done.

Average Number of SLA Violations: The SLA metric is defined according to [12, 14]. It is considered to be violated when a VM hosting container(s) does not get its requested CPU share. Alternatively, it can also be defined as the difference between the required and allocated CPU share of each VM.

Total Energy Consumption (kWh): This met-

Nano data center configuration and power characteristics					
nDC type	CPU (3 GHz) cores	Memory (GB)	P_{idle} (W)	P_{max} (W)	Number of nDCs
1	8	128	93	135	200

Container and virtual machine configurations							
Container type	CPU MIPS (1 core)	Memory (GB)	Number of containers	VM type	CPU cores	Memory (GB)	Number of VMs
1	4658	128	250	1	2 cores	1	113
2	9320	256	250	2	4 cores	2	113
3	18636	512	250	3	1 cores	4	112
				4	8 cores	8	112

TABLE 2. Simulation setup with respect to the configurations of nDCs, VMs, and containers.

ric represents the amount of energy actually utilized by the individual servers to provide various services to the end users. The total energy consumption of the nDC is calculated by aggregating the individual energy consumption of its servers. Here, CPU utilization is chosen as an important parameter to estimate energy consumption using the linear model as defined in [12, 15].

Results and Discussions

Containers are virtual entities that are more lightweight than VMs. Containers also support seamless internal and external migrations without imposing any significant overheads on the underlying system's CPU and memory utilization and network bandwidth. Thus, the proposed CoESMS efficiently schedules the tasks in an energy-efficient manner using containers. Figure 5a shows that the average number of container migrations is higher with CoESMS when compared to the other scheme by 8.42 percent. This directly implies that the proposed scheme minimizes the total energy consumption of the servers by triggering container migrations whenever required.

In the proposed scheme, container migrations are initiated based on the server's utilization level as mentioned above. As containers are relatively lightweight entities in comparison to VMs, they generate negligible computational and network overhead. However, the main overhead with containers is caused by the container's startup time. Hence, the overhead analysis of both schemes has been done with respect to the average startup time per container migration. Our results are depicted in Fig. 5b, which shows that the average startup time of the containers in both schemes is comparable. This is because the containers share the kernel instances and support quick startup times compared to VMs.

In addition, the experimental results shown in Fig. 5c further show that the proposed CoESMS also reduces the number of SLA violations compared to the existing scheme. This can be attributed to the fact that the proposed scheme maps the tasks to containers based on the multi-objective game theoretical approach, which formulates the game among the containers and selects the most optimal task for execution. As a result, the proposed scheme achieves an 11.82 percent decrease in the number of SLA violations compared to the existing scheme.

Finally, the overall energy utilization within 24 hours of simulation is shown in Fig. 5d. The results are derived using the energy consumption function shown in Fig. 4b.

This consumption is in accordance with different tasks with respect to their CPU requirements. Results demonstrate that the proposed scheme achieves 21.75 percent better results compared to the existing scheme. This can be attributed to the energy-efficient task scheduling on containers and their migrations, which is the core of the proposed CoESMS.

Conclusion

The rapid growth of the Internet of Things in the last decade has been enabled by cloud computing technologies to some extent. However, the exponential increase in the number of connected devices is creating new challenges for the existing cloud computing infrastructure in terms of network delays and higher bandwidth consumption. To address these emerging challenges, we leverage the benefits of edge computing for providing ubiquitous services to end users. However, task selection and scheduling at the edge of the network play a very significant role. Thus, in this work, we propose CoESMS for energy-efficient task selection and scheduling using cooperative game theory. We propose game theoretical models from broker to broker and container to container in order to minimize the overall energy utilization of servers and makespan of tasks while scheduling the tasks on lightweight containers. Extensive simulations have been performed on workload traces obtained from PlanetLab, and the results indicate that our proposed scheme performs better than the existing scheme with an energy utilization reduction of almost 21.75 percent and on average, a decrease of 11.42 percent for the number of SLA violations.

Acknowledgments

We thank the Guest Editors and the anonymous reviewers for their invaluable feedback and comments that helped us to improve the quality and presentation of this article. This work is supported by a fellowship from Tata Consultancy Services, India, and the Council of Scientific and Industrial Research with reference number: 22/717/16/EMR-II.

Containers are virtual entities that are more lightweight than VMs. Containers also support seamless internal and external migrations without imposing any significant overheads on the underlying system's CPU and memory utilization and network bandwidth. Thus, the proposed CoESMS efficiently schedules the tasks in an energy-efficient manner using containers.

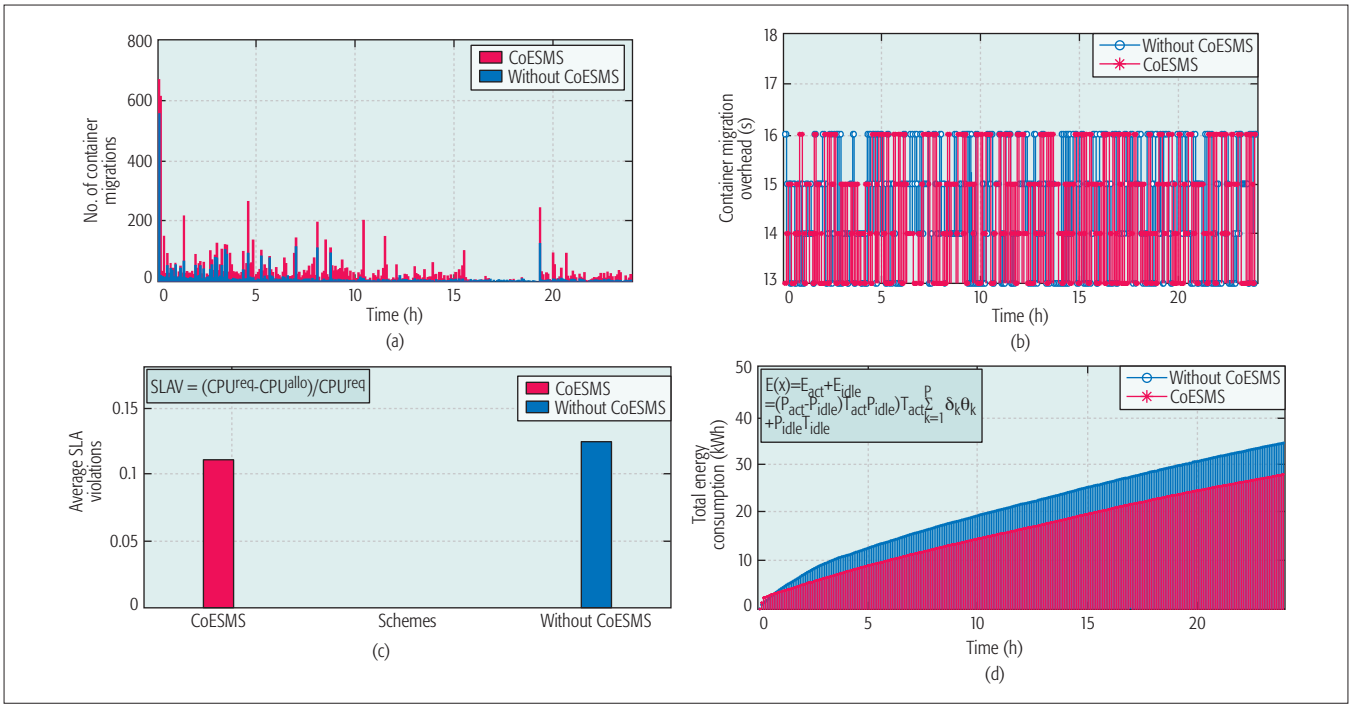

FIGURE 5. Performance comparison between the proposed and existing schemes with respect to different performance evaluation metrics: a) number of container migrations with respect to time; b) overhead analysis of the schemes with respect to number of container migrations per unit time; c) average number of SLA violations observed during task scheduling: d) total energy consumption of nDCs with respect to time.

REFERENCES

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," Cisco Internet Business Solutions Group, Apr. 2011.

[2] "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," Cisco Systems, 2015.

[3] M. Aazam and E.-N. Huh, "Fog Computing: The Cloud-IoT/IoE Middleware Paradigm," *IEEE Potentials*, vol. 35, no. 3, 2016, pp. 40–44.

[4] J. Gubbi *et al.*, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Generation Computer Systems*, vol. 29, no. 7, 2013, pp. 1645–60.

[5] C. Pahl, "Containerisation and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, 2015, pp. 24–31.

[6] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, Sept. 2014, pp. 81–84.

[7] R. Peinl, F. Holzschuher, and F. Pfitzer, "Docker Cluster Management for the Cloud-Survey Results and Own Solution," *J. Grid Computing*, vol. 14, no. 2, 2016, pp. 265–82.

[8] R. Duan, R. Prodan, and X. Li, "Multi-Objective Game Theoretic Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds," *IEEE Trans. Cloud Computing*, vol. 2, no. 1, 2014, pp. 29–42.

[9] R. Kaewpuang *et al.*, "Cooperative Virtual Machine Management In Smart Grid Environment," *IEEE Trans. Services Computing*, vol. 7, no. 4, 2014, pp. 545–60.

[10] W. Saad *et al.*, "Coalitional Game Theory for Communication Networks," *IEEE Signal Processing Mag.*, vol. 26, no. 5, 2009, pp. 77–97.

[11] N. Kumar *et al.*, "Bayesian Coalition Negotiation Game as a Utility for Secure Energy Management In a Vehicles-to-Grid Environment," *IEEE Trans. Dependable and Secure Computing*, vol. 13, no. 1, 2016, pp. 133–45.

[12] S. F. Piraghaj *et al.*, "Containercloudsim: An Environment for Modeling and Simulation of Containers in Cloud Data Centers," *Software: Practice and Experience*, Wiley, 2010.

[13] K. Park and V. S. Pai, "Comon: A Mostly-Scalable Monitoring System for Planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, 2006, pp. 65–74.

[14] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, 2012, pp. 1397–1420.

[15] M. Blackburn and G. Grid, "Five Ways to Reduce Data Center Server Power Consumption," *The Green Grid*, vol. 42, 2008, p. 12.

BIOGRAPHIES

KULJEET KAUR received her B.Tech. degree in computer science and M.E. degree in information security. She is currently working toward a Ph.D. degree with the Department of Computer Science and Engineering, Thapar University. Her main research interests include radio frequency identification, cloud computing, and vehicle-to-grid.

TANYA DHAND received her B.Tech degree in information technology from Guru Nanak Dev Engineering College, Ludhiana. She is currently pursuing an M.Tech degree in software engineering at Thapar University. Her research interests are mainly in cloud computing, the Internet of Things, and fog computing.

NEERAJ KUMAR is working as an associate professor in the Department of Computer Science and Engineering, Thapar University. He received his M.Tech. from Kurukshetra University followed by his Ph.D. from SMVD University in CSE. He was a postdoctoral research fellow at Coventry University, United Kingdom. He has more than 150 research papers in leading journals and conferences of repute. His research is supported by UGC, DST, CSIR, and TCS. He is an Associate Editor of *IJCS*, Wiley, and *JNCA*, Elsevier.

SHERALI ZEADALLY is an associate professor in the College of Communication and Information at the University of Kentucky. He received his Bachelor and doctorate degrees in computer science from the University of Cambridge, England, and the University of Buckingham, England, respectively. He is a Fellow of the British Computer Society and the Institution of Engineering Technology, England.