Please refer the paper as:

M. Adhikari, S. N. Srirama: Multi-Objective Accelerated Particle Swarm Optimization with a Container-based Scheduling for Internet-of-Things in Cloud Environment, Journal of Network and Computer Applications, ISSN: 1084-8045, 137:35-61, 2019. Elsevier. DOI: https://doi.org/10.1016/j.jnca.2019.04.003

Multi-Objective Accelerated Particle Swarm Optimization with a Container-based Scheduling for Internet-of-Things in Cloud Environment

Mainak Adhikari[a], and Satish Narayana Srirama[b]

[a]Department of Computer Science & Engineering,
Thapar Institute of Engineering and Technology, Patiala, India
[b]Mobile and Cloud Lab, Institute of Computer Science
University of Tartu, Estonia

E-mail: [a]mainak.ism@gmail.com, [c]srirama@ut.ee

Abstract: Over the last decades, cloud computing leverages the capability of IoT-based applications by providing computational power as a form of a container or virtual machines (VMs). Most of the existing scheduling strategies deploy the VM instances for each task which require maximum start-up time and consumes maximum energy for processing the tasks. However, containers are a lightweight process and start in less than a second. In this paper, we develop a new energy-efficient container-based scheduling (EECS) strategy for processing various types of IoT and non-IoT based tasks with quick succession. The proposed method use accelerated particle swarm optimization (APSO) technique for finding a suitable container for each task with minimum delay. Resource scheduling is another important objective in a cloud environment for better utilization of the resources in the cloud servers. The EECS strategy can deploy the containers on an optimal cloud server with an optimal scheduling strategy. The main objectives of EECS are to minimize the overall energy consumptions and computational time of the tasks with efficient resource utilization. The effect of the control parameters of the APSO technique is investigated thoroughly. Through comparisons, we show that the proposed method performs better than the existing ones in terms of various performance metrics including computational time, energy consumption, CO_2 emission, Temperature emission, and resource utilization.

Keywords— Internet-of-Things, Cloud computing; Containers; APSO technique; Computational time; Energy consumption.

1. Introduction

A diverse set of resources connected through a high-speed network offer a new computing paradigm in a distributed environment, called cloud computing [1]. The cloud providers provide three types of services including Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS) [2]. PaaS model offers a computing platform to the users including operating system, database, web server,

programming language, etc. The users can easily create and deploy a new application in that environment without managing the underlying software and hardware layers [3]. In the SaaS model, the users request for the application software without managing the infrastructure and platform to run the applications [4]. The IaaS cloud model consists of thousands of computing servers with sufficient resources in a cloud data center (CDC) and each of them can run multiple virtual machine (VM) instances simultaneously [5]. Cloud provides various types of computing resources such as CPU, memory, storage as an on-demand basis to the users as a form of VM. VM instances are becoming a popular technology in a cloud environment for deploying and executing large-scale distributed applications [6]. Each VM instance requires a separate operating system image, which adds overhead in the memory and storage footprint. This approach also has limited portability of applications between various cloud service providers.

Nowadays, in contrary to the traditional monolithic applications, the computing devices generate a composition of various types of small and specialized processes as a form of micro-service applications [7]. Instead of monolithic architecture, in micro-service architecture, each service is self-contained and implements a single application at a time instance. Container technologies such as Docker, Kubernets or Linux Containers, have such ability to deploy and execute micro-service applications efficiently and effectively in CDC [8-9]. Containers are stand-alone and self-contained units that enable users to handle a customized execution environment instead of bulky VM instances. Similar to the VM instances, the containers enable the resources of a single computing server and enable the resources to execute the micro-services or Internet-of-Things (IoT) applications [10-12]. Containers use the similar properties of a modern Linux operating system kernel such as lib containers, cgroups, and Linux containers. The main advantages of the containers are- (i) They can initiate quickly and launch within a second in a computing server based on the requirements of the tasks, and (ii) Containers require very small memory to store the information and consume a minimum amount of resources to run an application. Unlike VM instances, containers improve the utilization and performance of the resources and increase the parallelism among the tasks [13-14]. Moreover, the researchers and developers prefer to deploy the containers in the private cloud environment instead of the public and hybrid cloud environment due to security and privacy.

Containerized applications deploy the micro-services on a cluster of heterogeneous servers in a CDC, rather than a single server with a set of VM instances. This property may create a lightweight environment for the applications and utilizes the computing resources efficiently. Over the years, most of the Industries and educational organizations are relying on this technology for deploying multiple modern-day applications such as IoT and event-driven applications, web services, Big-data, etc. Such types of applications need faster completion and response time with minimum system overhead [15]. However, most of the recent scheduling strategies are based on VM instances for executing such modern-day applications in CDC which consumes the maximum amount of storage and processing power due to its own operating system. Using VM instances also consume maximum energy of the CDC and impose a large performance penalty as a form of various Quality-of-Service (QoS) parameters. Another important issue in the cloud environment is resource scheduling as a form of VM scheduling or container scheduling. The main purpose of an optimal scheduling strategy is to find an optimal loaded active server for the selected VM instances or containers which may maximize the resource utilization efficiency in the CDC. Moreover, most of the existing scheduling strategies have more than one conflict objectives while executing the applications in the CDC. These problems are referred to as multi-objective optimization problems (MOPs).

In past two decades, two types of multi-objective evolutionary algorithms have been developed to solve MOPs- (i) finding a set of Pareto optimal solutions in a single run and (ii) non-Pareto-based algorithm based on decomposition method [16]. Here, the algorithm decomposes a MOP into a number of single-objective optimization problems (called sub-problems). There are still many recent optimization algorithms whose effectiveness is yet to be explored in the context of a multi-objective scheduling problem [17]. The optimization algorithms are the high-level problem-independent algorithms which have a set of rules to find an optimal solution to a given problem. The convergence speed of the optimization algorithms is the global

(or nearly global) optimal which is better than the traditional techniques [18]. Therefore, the meta-heuristic algorithms have also been increasingly used to solve the multi-objective optimization based scheduling strategy in a cloud environment. One such algorithm is the accelerated particle swarm optimization (APSO) technique [19-20]. In the twentieth century, the PSO technique was introduced by Eberhart and Kennedy. The standard PSO uses both the current global best and the individual best of the particles to find an optimal solution. Due to the limitations of convergence speed and accuracy, the PSO algorithm is modified based on its velocity and displacement and called as APSO technique. APSO technique uses the global best of the individuals for convergence of the algorithm and reduces the randomness as iteration proceeds.

Here, we propose a new multi-objective aware energy-efficient container-based scheduling (EECS) strategy using APSO technique. The contribution of the proposed method is two folded. The first contribution is to find an optimal container for each requested task based on multi-objective aware APSO technique. In this phase, we design a fitness function based on bi-objective optimization parameters including energy consumption and computation time of the executing devices using the weighted-sum method. This method linearly aggregates all the individual objective function of a MOP into one objective by using a weight vector. During finding an optimal executing device each objective of the particle is normalized based on the maximum and minimum values of the corresponding objective function. Such normalized objective function helps to eliminate the impact of various amplitudes on multi-objective. The existing APSO technique helps to find an optimal executing device for each task as a form of containers, local IoT devices or VM instances in a 2-D plane based on the bi-objective optimization parameters. The main objective of the first contribution is to minimize the total energy consumption and computation time of the CDC. The second contribution is to schedule the tasks on an optimal loaded server with a rule-based strategy. This may maximize the utilization of the resources and minimizes the resource wastage of the computing servers and the CDC. For simulation, we deploy the Docker image in the private cloud servers for each task based on the bi-objective optimization parameters and the resource requirements by the tasks. The proposed EECS strategy is compared with the existing ones in terms of various performance metrics including computational time, energy consumption, CO_2 emission, Temperature emission, and resource utilization. The results of the performance metrics are further evaluated based on various statistical parameters including maximum, minimum, mean and standard deviation. The major contributions of this work are summarized as follow.

A. Discuss the merits of containers for executing micro-service applications such as IoT and Event-driven applications, web services, Big-data over VM instances with a comparative analysis.
B. Design a multi-objective optimization problem considering two major scheduling objectives, namely, energy consumption and computational time. The multi-objective aspects are dealt with a weighted sum approach based fitness function to evaluate the quality of the solution.
C. APSO technique has been incorporated to address the container-based scheduling strategy, which selects a suitable container for each task.
D. Devise an effective rule-based strategy for selecting an optimal loaded server in the CDC for the selected containers for further execution.
E. Finally, the effect of the control parameters of the APSO technique is investigated thoroughly. We also evaluate the performance of the proposed algorithm over synthetic datasets using various performance metrics.

The rest of the paper is organized as follows. The merits and demerits of various existing VM-based and container-based scheduling strategies in a cloud environment are discussed in Section 2. The overview of the multi-objective optimization strategy and APSO technique are presented in Section 3. The overview of the containers and their effectiveness over VM instances for micro-service applications are presented in Section 4. The system model and problem formulation of the proposed method is discussed in Section 5. The proposed EECS strategy is discussed in Section 6. The performance analyses of the proposed algorithm are discussed in Section 7. Finally, the conclusion and future work are given in Section 8.

2. Related Work

Over the times, there have been numerous research initiative tailored in task scheduling for solving various types of multi-objective optimization problems (MOPs) in a cloud environment. Nowadays, most of the researchers are used different types of meta-heuristic strategies such as particle swarm optimization (PSO), Genetic algorithm (GA), Honey-bee algorithm (HBA), ant-colony optimization (ACO), Fuzzy strategy, etc. to solve MOPs due to their accuracy and convergence speed. Here, we discuss few currently published MOP-based task scheduling strategies in the cloud [20-22]. Kaur et al. have proposed a task scheduling strategy for solving MOP problem using bacteria foraging optimization technique [20]. The algorithm solves three objectives of the scheduling strategy such as flow time, makespan and resource usage cost. This algorithm found the suitable VM instances for assigning the independent jobs which meet the scheduling objectives and deployed on a server for further processing. Kumar et al. have proposed an energy-efficient task scheduling strategy in the cloud [21]. This algorithm met three scheduling objectives of the server such as Execution cost and time and the total energy of the server. The authors have applied the PSO technique for finding an energy efficient VM instance for each task that met the total execution cost and time. Ramezani et al. designed a PSO based load balancing strategy in the cloud [22]. The authors have applied the PSO technique for finding an optimal server of a CDC to migrate the VM instance from a heavily loaded server to the minimum loaded server. The main objectives of this algorithm are to balance the loads among the servers and minimize the total execution time of the tasks.

The above-mentioned task scheduling algorithms meet multiple scheduling objectives using various meta-heuristic strategies; however, the algorithms fail to design an efficient task scheduling strategies to minimize the total computation time and energy consumptions of the tasks, especially IoT-based tasks. Most of the real-time tasks such as IoT, event-driven applications, etc want the result with quick succession. However, the VM instances take too much time (nearly a minute) to deploy in a CDC for processing a task and consume more energy due to the consumption of an excessive amount of resources. This may maximize the total computational time and the energy consumption of the tasks. To overcome the above-mentioned challenges, a new technique is initiated in a cloud environment, called container-based scheduling. Containers require minimum time to deploy and consume a minimum amount of resources for executing a task. Due to the advantages of the containers, in recent years, the researchers have developed different types of container-based scheduling in a cloud environment [23-37]. Bernstein et al. have proposed a container-based scheduling strategy in a PaaS cloud environment [23]. The author discussed the overview of the containers over VM instances and the importance of this technology for executing various types of applications. This paper deployed a Docker container over Linux container and Kubernetes. This paper only presented the architectural view of the containers in a cloud environment. Kaewkasi et al. have proposed a container-based scheduling strategy based on ACO technique [24]. The authors have developed an optimal scheduler over the Docker container which can efficiently schedule the tasks with minimum execution time. The main objective of this algorithm is to utilize computing resources efficiently and improves the performance of the system.

Li et al. have proposed a container-based scheduling schema in the cloud to achieve high availability of the computing resources [25]. This scheduling strategy found a suitable host to deploy the containers and monitor the hosts for further deployment strategy and container migration purpose. The containers are migrated to another host due to the failure of the host or balancing the loads among the available hosts in the CDC. Yi et al. have proposed a container-based strategy in a Fog-cloud environment [26]. The main objective of this strategy is to minimize the delay of the tasks while minimizing the execution time. This method tried to deploy most of the tasks to the Fog nodes, however, due to lack of resources, few tasks need to deploy to the CDC for further computation purpose. Perez et al. have designed a container-based task deployment strategy in a serverless environment [27]. This algorithm deployed the Docker container over AWS Lambda platform and presented the performance improvement of this strategy over existing ones. The authors stated that this environment is most suitable for bursty workloads of short stateless jobs. Tang et al. have developed a container-based auto-scaling strategy in an elastic cloud environment [28]. The authors

have developed a container-based auto-scalar model in a cloud environment. This paper developed a mathematical model for auto-scaling strategy and discussed the necessity of auto-scaling strategy in a container-based cloud environment.

Zhou et al. have designed two online and offline container-based scheduling strategies in cloud environment [29]. The proposed strategies select an optimal container for each task which meets the deadline while maintaining an efficient inter-container dependency. The objectives of the paper are two folded. First, the authors use a compact-exponential technique for handling non-conventional scheduling objectives. Secondly, the authors design an optimal framework for generating a primal solution based on the existing constraints and produced a near-optimal solution. Zhang et al have developed a cost-effective container-based scheduling strategy in cloud environment [30]. The authors have developed an adaptive and effective scheduler based on integer linear programming technique which can easily integrate to the container orchestration framework. The main objective of the work is to minimize the overall cost of the tasks including computational and transmission cost. Tao et al. have developed a global container-based resource allocation strategy based on fuzzy inference system flexible deployment and high availability of the resources [31]. The authors applied their algorithm over various use-cases to prove the effectiveness and efficiency of the proposed method. Guerrero et al. have designed an optimal container allocation strategy using a genetic algorithm with Non-dominated Sorting Genetic Algorithm-II [32]. The proposed strategy minimizes the network overhead and system failure while maximizing system performance and resource utilization of the server. The authors are mainly considered the micro-services for processing in containers as a form of Kubernetes with an efficient container allocation strategy including resource elasticity. Li et al. have developed a container-based scheduling strategy in a cloud environment using artificial fish swarm algorithm [33]. The main objectives of the algorithm are to improve the load balancing among the cloud servers and utilize the computing resources efficiently.

Mao et al designed a container-based scheduling strategy in a cloud environment for minimizing the drawback of the VM instances [34]. The proposed strategy deployed the containers as a form of Docker image for processing the tasks with minimum delay and computation time. The authors classified the tasks into two categories- CPU-intensive and memory-intensive and deployed the containers for the tasks on the suitable node in the cluster. Havet et al. have introduced container-based scheduling in the CDC that leverages the principals from the generation of garbage collection [35]. The proposed framework investigated the current status and resource usage of the active containers and deployed the containers on the physical machines based on the requirements of the containers. The main objective of the strategy is to minimize the overall energy consumption of the cloud servers. Fazio et al. have designed a container-based strategy for processing micro-services in cloud environment [36]. This strategy maximizes the reliability and scalability of the CDC with efficient resource utilization. Wan et al. have developed a container-based strategy for processing micro-services [37]. The authors have designed a communication-efficient container-based framework which deployed the containers on the cloud servers as a form of Docker images. The main objectives of the work are to minimize the overall cost while utilizing the resources efficiently. In our previous paper [38], we have developed a scheduling and resource provisioning strategy in an IaaS cloud environment using K-means and Bat algorithm. This algorithm helped to find a set of the optimal loaded server for assigning the VM instances for further execution. This strategy also derived a rule-based strategy for finding a suitable VM instance for each task. The superiority of this method over existing ones is also shown using various simulation runs.

Most of the existing container-based scheduling strategies try to optimize a single objective of the QoS parameters. However, single objective optimization may reduce the performance and efficiency of the CDC. Nowadays, most of the applications or tasks are generated from real-time devices IoT devices. So, to run such types of applications or tasks with minimum computational time and energy consumption is one of the challenging issues in a cloud environment, which have not discussed yet in the existing literature. To overcome the short comes of the existing strategies, we have developed an energy efficient container-based

scheduling strategy for solving multiple objectives of the task scheduling. Here, the IoT-based task may execute locally or offload the resource-intensive or event-driven tasks to the CDC. The task scheduler of the CDC finds a suitable container for each task based on the APSO technique using two QoS parameters including energy consumption and computational time of the computing devices. This may minimize the total computational time and the energy consumption of the servers. The proposed strategy also finds an optimal server based on a rule-based strategy for assigning the selected containers. This may maximize the resource utilization of the computing devices efficiently.

3. Preliminaries

Here, we first discuss the multi-objective optimization strategy followed by the overview of the APSO technique.

3.1 Multi-objective Optimization

Definition 1. *Global Minimum:* Given a function $F: \varphi \subseteq R^n \to R, \varphi \neq \emptyset$, for $y \in \varphi$, the global minimum of function *f* is $F^* \triangleq F(y^*) > -\propto$, if and only if

$$\forall y \in \varphi: F(y) \leq F(y^*) \tag{1}$$

Here, y^* represents the global minimum solution, F is the objective function and the set of φ is the feasible region $\varphi \in S$, where S represents the whole search space [16].

Definition 2. *Multi-objective Optimization (MOO):* MOO problems (MOPs) at time *t* is defined as follows [16].

Minimize $\boldsymbol{F(y, t)} = \{f_1(y, t), f_2(y, t), f_3(y, t), f_4(y, t), ..., f_m(y, t)\}^T$

Subject to: $\boldsymbol{x} \in \theta$

where, $\boldsymbol{y} = (y_1, y_2, y_3, y_4, ..., y_m)^T$, is the *m* dimensional decision vector, θ is the decision space and $\boldsymbol{F}: \theta \to R^i : i = (1, 2, 3, \dots, n)$, consists of *n* real-valued objective functions and R^n is called the objective space. In other word, $\boldsymbol{y} = (y_1, y_2, y_3, y_4, ..., y_m)^T$ which will satisfy the *n* inequality constraints which is defined as

$$h_i(y) \geq 0: i = 1, 2, 3, \dots, n \tag{2}$$

The *p* equality constraints are represented are

$$g_i(y) = 0: i = 1, 2, 3, \dots, p \tag{3}$$

This will optimize the vector function as

$$\boldsymbol{F(y, t)} = \{f_1(y, t), f_2(y, t), f_3(y, t), f_4(y, t), ..., f_n(y, t)\}^T \tag{4}$$

where, $\boldsymbol{y} = (y_1, y_2, y_3, y_4, ..., y_m)^T$ represents the *m* dimensional decision vector.

Definition 3. *Decision Vector Domination:* A decision vector y_1 Pareto dominates another vector y_2 at time *t*, denoted by $y_1(t) \succ y_2(t)$, if and only is,

$$\left.\begin{matrix} \forall i = 1, \dots, M, & f_i(y_1, t) \preccurlyeq f_i(y_2, t) \\ \exists i = 1, \dots, M, & f_i(y_1, t) \prec f_i(y_2, t) \end{matrix}\right\} \tag{5}$$

Definition 4. Pareto Optimal Set: Let *y* and y^* are decision vectors, and if the decision vector y^* is said to be non-dominated at time *t* if and only if there is no other decision vector *y* such that $y(t) \succ y^*(t)$. The Pareto-Optimal Set (POS) is the set of all Pareto optimal solutions [16], *i.e.:*

$$POS = \{y^*(t) | \nexists y(t), y(t) \succ y^*(t)\} \tag{6}$$

Definition 5. Pareto-optimal Front: Pareto-optimal Front (POF) is the corresponding objective vectors of the POS at time *t*.

$$POF = F\{y^*(t) | y^* \epsilon\, POS \tag{7}$$

An ideal multi-objective optimization strategy must have the ability to find a set of optimal solutions and at the same time, the solutions must be as diverse as possible. Here, we solve a bi-objective optimization strategy in a container-based cloud environment. The main objective of this work is to find an optimal executing device as a form of local IoT device, container or VM instances based on energy consumption and computational time of the tasks.

3.2 Accelerated Particle Swarm Optimization (APSO) Technique

Particle swarm optimization (PSO) is a population-based stochastic optimization technique developed by Eberhart and Kennedy in 1995. The method was inspired by social behavior of bird flocking and fish schooling that does not have any leader in the group. A flock of animals achieves their best condition simultaneously by communicating among the members who already have a better position. Animals who are in a better position will inform the others to their flocks and others will follow them simultaneously. This would happen repeatedly until the best condition of the food source is discovered. The PSO algorithm finds the optimal value of a population using the working principle of the animal society. PSO consists of a swarm of particles which represents a potential value. A particle represents an individual (either fish or bird) from a population. It has the ability to move to the defined problem space and represents a candidate solution based on an optimization technique. Each particle is represented by its position and velocity. Each Particle *i* keeps track of its best position by Pbest (Y^*_i) = {Y^*_1, Y^*_1, Y^*_1,..., Y^*_n}. The best of all Pbest is denoted as the global best position Gbest (G^*). Values of each particle are determined based on the fitness function. The movement of the particles is defined by their velocity. The velocity is represented by a vector and it has a magnitude and direction which is defined as follow.

$$V_i(l+1) = V_i(l) + \alpha..r_1.(G^* - Y_i(l)) + \beta.r_2.(Y_i^*(l) - Y_i(l)) \tag{8}$$

where Y_i and V_i are the position vector and velocity of the i^{th} particle respectively and r_1 and r_2 are two random numbers uniformly distributed in the interval [0,1]. The velocity is determined by the best position of the particle so far and the best position in which any of the particles has been so far. Based on this, it is imperative to be able to measure how good or bad a particle position is. At each step, the algorithm changes the velocity of the particle towards the Pbest and Gbest location. The updated position of each particle is represented as

$$\vec{Y}_i(l+1) = \vec{Y}_i(l) + \vec{V}_i(l+1) \tag{9}$$

To increase the diversity of quality solution for each individual, the normal PSO technique uses both the current *Pbest* and *Gbest* position of the individuals. Though, the diversity can be simulated using some randomness. A modified version of PSO algorithm is called as APSO algorithm [20] which is used to accelerate the convergence of the algorithm based on global best location. The velocity vector of the APSO algorithm is formulated in Eq. (10) [39-40]. The illustration of the velocity calculation of APSO technique is shown in Fig. 1.

$$V_i(l+1) = V_i(l) + \alpha\varepsilon + \beta(Y^* - Y_i(l)) \tag{10}$$

where **ε** is a random vector uniformly distributed in the range [0, 1]. The updated location of a single particle to increase the convergence even further using one step is formulated in Eq. (11).

$$Y_i(l+1) = (1-\beta)Y_i(l) + \beta G^* + \alpha\varepsilon^l \tag{11}$$

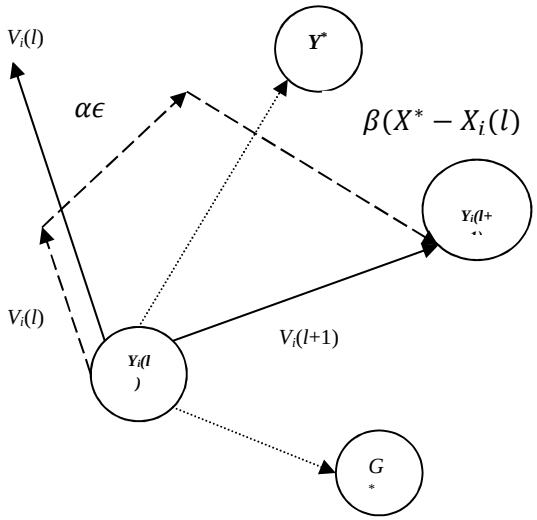


Fig. 1. Illustration of velocity calculation of APSO technique

The APSO technique is used to find optimal executing devices based on two QoS parameters including energy consumption and the computation time of the tasks. Due to the faster convergence speed and accuracy from the existing optimization technique, i.e. genetic algorithm, ant-colony optimization, honey-bee algorithm, PSO, simulated annealing, we select the APSO technique for solving the bi-objective optimization strategy for container-based scheduling strategy.

4. Overview of containers and their effectiveness

Cloud computing is a computing paradigm, where all computing resources are available to process various types of tasks and applications as a pay-per-use basis. The cloud providers receive various types of applications or tasks to execute based on their requirements. We broadly classify the tasks and applications into two categories- IoT based task/applications and Non-IoT-based tasks/applications. The IoT-based devices generate various real-time applications which may execute locally in the micro-services or microprocessor or may offload the tasks to the CDC for effective and faster processing. The Non-IoT devices generate various types of batch mode tasks or applications which require maximum resources for processing and storage. Moreover, the Non-IoT devices may generate few event-driven applications for further processing. The Non-IoT devices directly offload the tasks or applications to the CDC, which may be CPU-intensive or memory-intensive applications. The IoT-based applications are either event-driven or request-based applications. The event-driven applications are generated by the users or the systems based on some specific events such as keystrokes, fraud detection, real-time warehouse management, receiving HTML based messages, etc. The users also request for some specific resources for executing or storing their tasks, which types of applications are called request based applications. Request-based applications are either CPU-intensive (require more CPU) or memory-intensive (require more storage) applications. The classification of various types of user-driven applications/tasks is shown in Fig. 2.

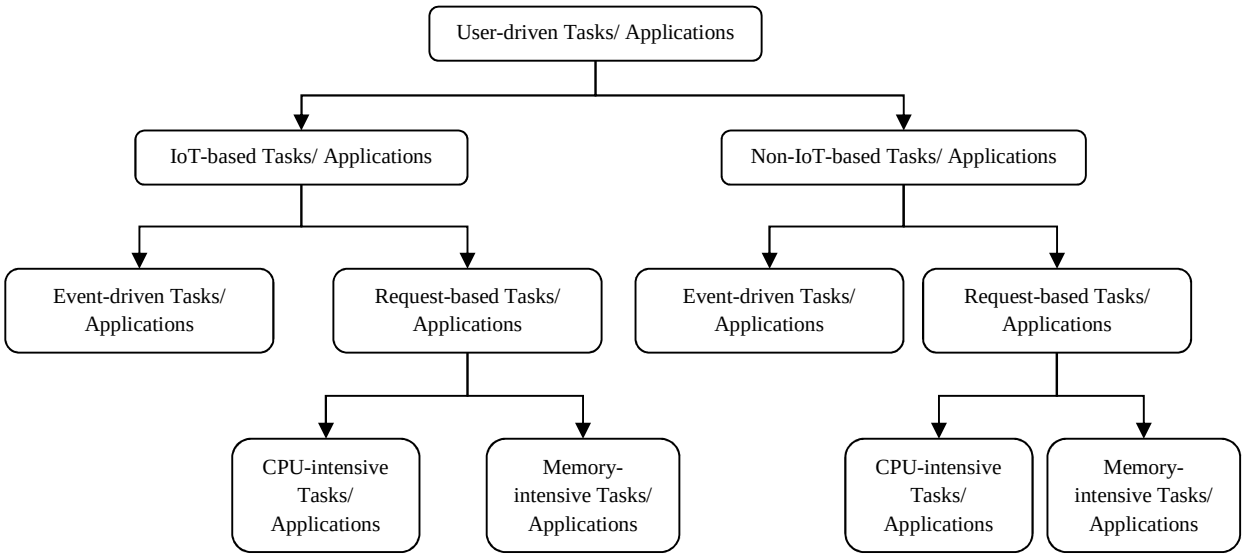


Fig. 2. Types of user-based Tasks/ Applications

The underlying technology that makes the cloud environment important in a distributed environment is virtualization. Virtualization technology allows the providers to assign multiple numbers of VM instances for the tasks or applications without the knowledge of the users. The maturity of the cloud environment mostly depends on the virtualization technology which improves the utilization of the resources and minimizes the energy consumption of the tasks. In past decades, cloud providers provide the resources to the users as a form of VM instances. The VM instances virtualize the computing resource in hardware level where each VM instance has its own underlying Operating System (OS) and shares the computing resources of a host server based on "Bear-Metal" or "Hypervisor". The hypervisor or virtual machine monitor of each server is a software or firmware that helps to deploy and run the VM instances. The main advantage of the VM instances is that they help to virtualize the resources of the servers and increases the parallelism of the tasks while improving the utilization of the computing resources. However, the VM instances not only run the full version of the OSs but also a virtualized copy of the hardware needs to run by the OS of each VM instance. This may quickly add a lot of storage of the memory and increase the CPU cycle. The VM instances also take more than a minute time to deploy and maximize the overall energy consumption and cost of the tasks.

However, to overcome the drawbacks of the VM instances, a new type of virtualization technology is used, known as containers. Containers provide a lightweight environment for the tasks and applications which use the process-level virtualization technique for sharing and utilizing the computing resources efficiently. Containers are stand-alone and self-contained units that enable users to handle a customized execution environment in the form of Docker, Linux Containers or Kubernetes images instead of bulky VM instances. Containers sit on the top of the server with their own OS. Each container shares the OS kernel (e.g. libraries, binaries) of the hosted server. This may reduce the needs to reproduce the OS code and server may run a number of tasks or application with a single OS. The structure of the VM instances and containers are shown in Fig. 3. Thus the containers require only a few megabytes to store and start-up within a few milliseconds. This technology may increase the parallelism among the tasks and utilize the resources more efficiently than VM instances. Container technology is better for the cloud providers that want to run a maximum number of applications or tasks on a minimum number of servers. This may minimize the overall energy consumption and the exceptional cost of the applications or tasks. The differences between the two different virtualization technologies in a cloud environment are shown in Table 1.

Table 1. Difference between Containers and VM instances

Properties	Containers	Virtual Machines (VMs)
Process Type	Lightweight.	Heavyweight.

Type of OS	All containers share the OS of the host server.	Each VM has its own OS.
Performance	Native Performance	Limited Performance
Portability	Easily portable from one server to another server.	Complex process to migrate VMs from one server to another server.
Auto-scaling Process	Easily scale-in or scale-out the resources as per requirements.	Complex process for scale-in and scale-out.
Type of virtualization	OS-level virtualization.	Hardware level virtualization.
Start-up Time	Start-up time in seconds.	Start-up times in minutes.
Required Memory space	Require less memory space to run an application or task.	Require more memory space to run an application or task due to the overhead of OS.
Cost Minimization	Effective use of resources and less consumption of resources may reduce the overall cost.	Require more cost due to the consumption of more amounts of resources.
Energy Efficient	Less resource consumption may reduce overall energy consumption.	Maximize energy consumption due to the maximum amount of resource usage.
Parallelism	Maximize the parallelism among the tasks and applications.	Reduce the parallelism among the tasks and applications.
Security	Less secure due to process-level virtualization.	More secure due to full isolated and maintain hardware-level virtualization.

For experimental purpose, we have deployed Docker containers in four heterogeneous private cloud servers and assigned the 1000 tasks to the Docker images as per their requirements. During processing the tasks on the Docker images we have investigated that the Docker images contain very minimum time to compute the tasks with minimum energy consumption and also increases the parallelism among the tasks. This may also increase the resource utilization of the cloud servers. The same strategy we have also applied over VM instances and assigned the tasks to the suitable VM instances as per the resource requirements by the tasks. However, due to maximum resource consumption by the VM instances and maximum deployment time, the performances of the cloud servers are going down which increases the computation time and energy consumption of the tasks. This virtualization technique also minimized the parallelism among the task and the resources utilization of the cloud servers. The performance analysis between containers and VM instances in terms of start-up time, memory usage and energy consumption is shown in Fig. 4.

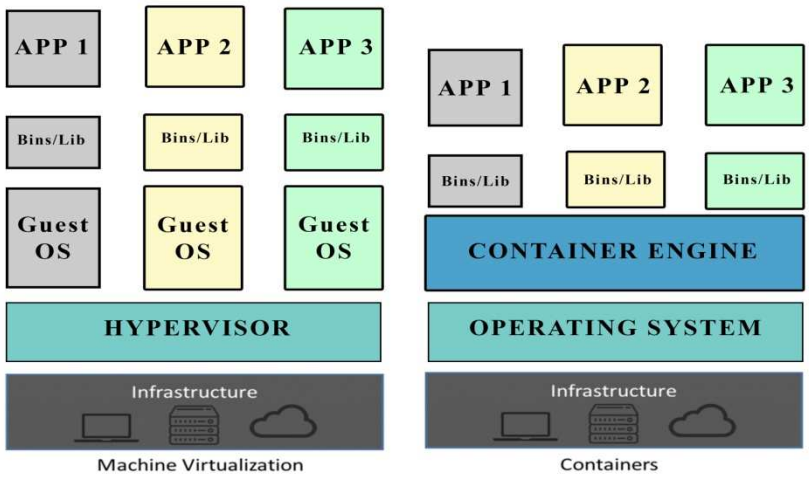


Fig. 3. Difference between VM instances and containers based on their structure

The container technologies have widely used by the academicians and researcher due to the advantages of Docker and Kubernetes. Docker is an open platform tool which makes it easier to create, deploy and to execute the tasks or applications as a form of containers. Docker Containers allows the system to spread the tasks among the resources which can execute them in a faster way. The main components of a Docker are

Docker Swarm, Docker Compose, Docker Images, Docker Daemon, and Docker Engine. Docker can manage its own infrastructure in the same ways as an application is being managed. Docker uses Linux kernel control groups and namespace to run independent containers in a server. The control groups provide the various types of resources to run a task or an application and namespace provide the details about the running application of the operating environment such as process tree, User Id, network usage rate, etc. The main advantage of the Docker platform is to ship, test, and deploy application quicker which may reduce the computational time and cost of the tasks or applications.

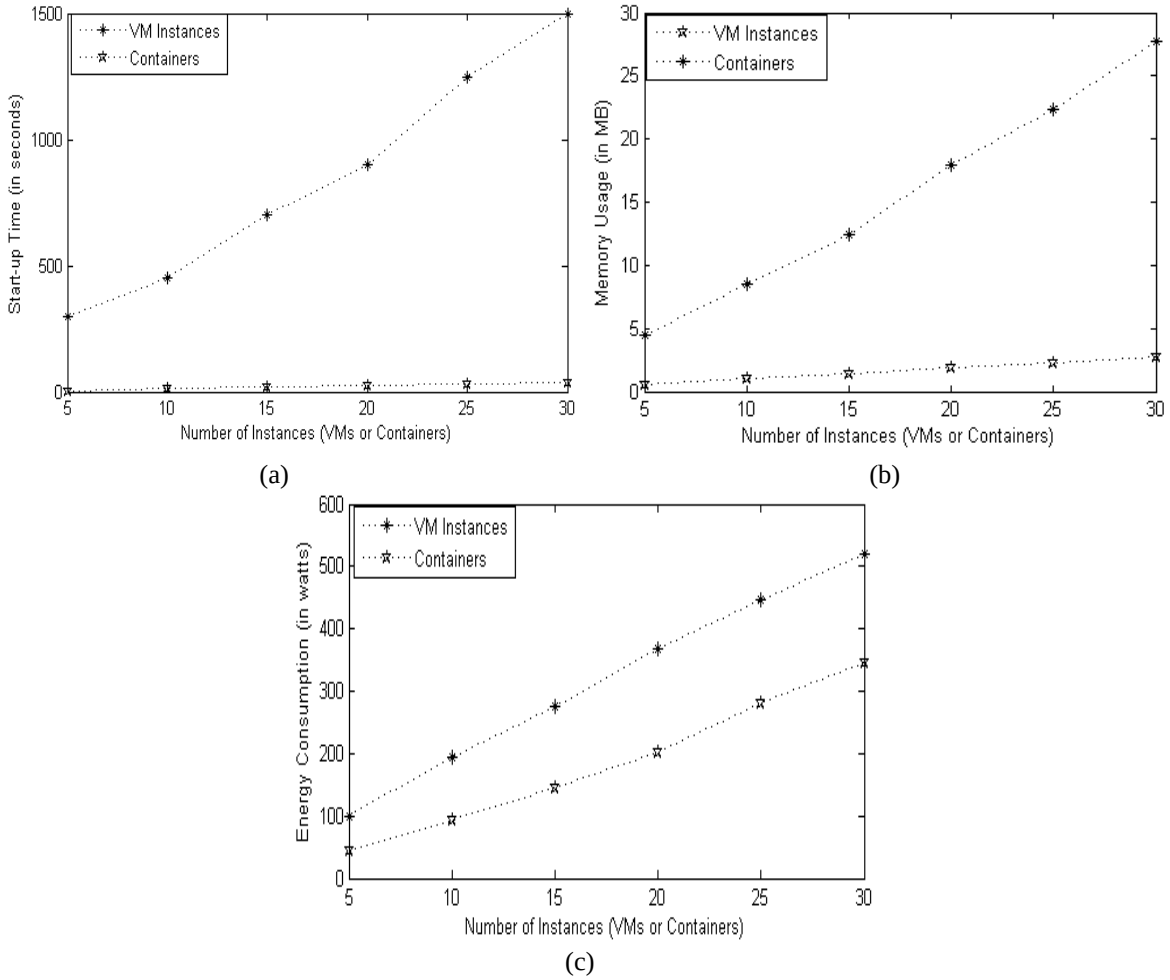


Fig. 4. Performance analysis between VM instances and containers: (a) Start-up Time; (b) Memory Usage; (c) Energy consumption

Kubernetes is an alternative of Docker container which provides a scalable, loosely coupled environment to run an application. A Kubernetes form a master-slave architecture where the cluster consists of a master node and multiple slave nodes. The master node schedules the tasks in order and deploys them to the slave nodes in the cluster. The slave nodes execute the tasks and return back the result to the master node. The master node works as a controller in the Kubernetes cluster. The Kubernetes architecture incorporates the concept of a pod which can host a set of containers with some shared resources at the same server. It plays an important role to maximize the performance of the Kubernetes. Kubernetes supports two types of pods- (i) Service pods- This can run permanently and helps to see the background workload of the cluster. (ii) Job/Batch pod- This helps to execute the tasks and terminate from the server on task completion. While launching a pod of a Kubernetes, it requests a set of resources. The Kubernetes scheduler is responsible to select the best-fit resources for the tasks. Docker and Kubernetes run on a different level of a server. Kubernetes can integrate the Docker engine for monitoring the scheduling and execution of the tasks. However, Docker can create its own container image to run an application using the docker build command.

5. Problem Formulation

The proposed EECS strategy requires addressing several research challenges for enabling effective and efficient operations. In this section, we discuss the system model followed by the problem statement to address the proposed method.

5.1 System Model

Here, we consider a container-based CDC model with a set of computing servers that can accommodate multiple numbers of containers and VM instances as per the requirements of the tasks. The container-based CDC model is shown in Fig. 5. Here, we consider two types of users such as IoT based and Non-IoT based who can transmit the various types of applications or tasks (e.g. event-driven tasks, request-based tasks, CPU-intensive tasks or memory-intensive tasks) to the CDC for further processing. However, most of the IoT devices have some processing capability (e.g. microprocessors, microcontrollers, FPGAs, SOCs) and storage for executing some micro-services applications or tasks. Otherwise, the IoT devices and all non-IoT devices can upload the applications or tasks to the CDC. The admission controller receives the tasks from the users and decides whether the tasks can be admitted or not. This decision is based on the availability of the computing resources of the servers. Another activity of the admission controller is to assign the control of the tasks either to the Container scheduler or VM manager for further processing. Based on our strategy, by default, the controls of the tasks come to the Container scheduler due to the benefits of the container deployment policy than the VM instances. The admission controller assigns the control of the tasks to the VM manager based on the requests of the users for executing the tasks on the VM instances.

The main responsibility of the Container scheduler is to create various types of containers based on the requirements of the tasks. However, the main activity of the VM manager is to select a VM instance from the pool of VMs for each task based on its resource requirements and characteristics. The tasks scheduler is responsible for holding detail information of the active servers in the CDC and receives the information about the availability of the resources in the servers in each time interval. This may help to find an optimal server with a minimum load for deploying the selected containers or the VM instances. The task scheduler is also responsible for auto-scaling the servers or the containers based on the overall resource availability and the tasks or applications arrival rate in the CDC. The symbols of the variables along with their descriptions are given in Table 2.

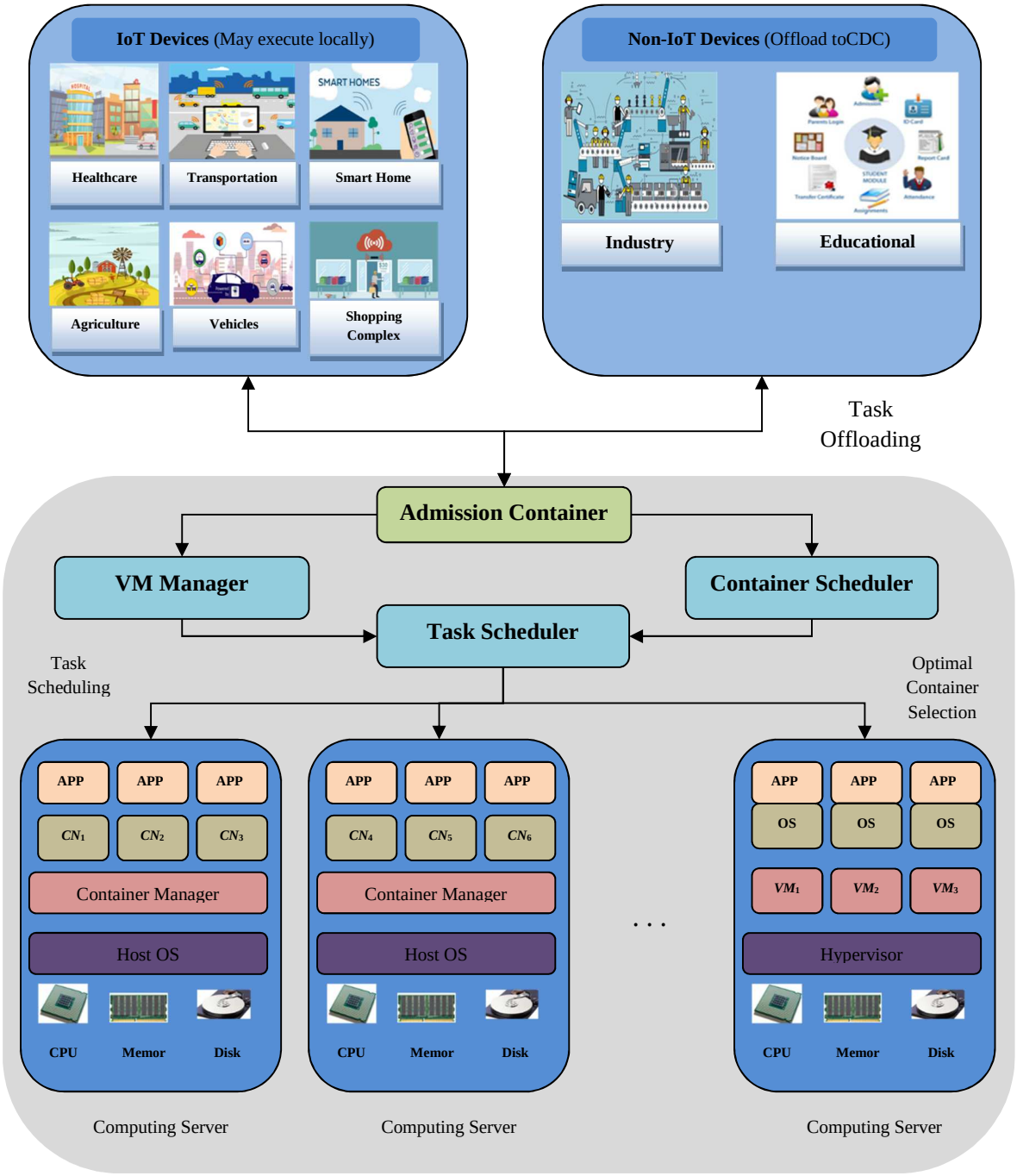


Fig. 5. Container-based cloud data center

Table 2. Computational elements of IoT

Symbols	Descriptions
CP_j^{CN}	CPU capacity of the container j
$\|CR_j\|$	Number of CPU cores
$sizeof(CR)$	Size of each CPU core
CM_j^{CN}	Memory capacity of the container j
$\|M_j\|$	Number of blocks in the memory
$sizeof(M)$	Size of each memory block
T_i^t	Total tasks are arrival rate at each server i
P_k^t	Rate of a Poisson process for task generation in each time slot
TT_{aj}	Transmission time between the user a and CDC j

SD_{aj}	Serialization delay between the user a and CDC j
PD_{aj}	Propagation delay between the user a and CDC j
SZ_{ka}	Sizes of a task k emitted from a device a
TR_{aj}	Transmission rate between the user a and CDC j
P_j	Transmission power
TT_{ja}^{d}	Total time requires to transmit the downlink traffic from CDC j to user a
TT_{ja}^{u}	Total time requires to transmit the uplink traffic from CDC j to user a
ET_{kc}^{a}	Execution time of a task k emitted from a device a in a computational device c
CT_{kc}^{a}	Computation time of a task k emitted from a device a in a computational device c
$C_c(t)$	CPU usage of a computational device c at time t
$M_c(t)$	Memory usage of a computational device c at time t
$L_c(C, M)$	Current load of a computational device c
$RC_c(C, M)$	Reaming load of a computational device c
$TL_k(C, M)$	Required load by a task k
RU_c	Resource utilization of the device c
$L_c^{max}(C, M)$	Maximum resource capacity of the device c
TP_c^{C}	Total power consumption of CPU by a computational device c
TP_c^{M}	Total power consumption of memory by a computational device c
DP_c^{C}	Dynamic power consumption of CPU by a computational device c
SP_c^{C}	Static power consumption of CPU by a computational device c
DP_c^{MR}	Dynamic power consumption of memory during read by a computational device c
DP_c^{MW}	Dynamic power consumption of memory during write by a computational device c
TP_c	Total power consumed by the device c
TCE_k	Total computational energy by a task T_k, executing in a computing device c
TEC_{kj}	Total energy consumes by a task while T_k executing in a container in a CDC j
$CE_c^{C}(t)$	The CO_2 emission rate of the CPU
$CE_c^{M}(t)$	The CO_2 emission rate of the memory
TCE_c	Total CO_2 emission rate of a device c
$T_c^{C}(t)$	The temperature emission rate of the CPU
$T_c^{M}(t)$	The temperature emission rate of the memory
TM_c	Total temperature emission rate of a device c

A. Network Model

We consider N computational servers, denoted by the set $S = \{S_1, S_2, S_3, ..., S_N\}$, deploying in a CDC and connected by the same Local Area Network. Each server can host U number of heterogeneous containers and H number of VM instances as per the requirements of the tasks and the resource availability of servers, represented as $CN = \{CN_1, CN_2, CN_3, ..., CN_U\}$, $VM = \{VM_1, VM_2, VM_3, ..., VM_H\}$ respectively. Let consider that there are M IoT users denoted by the set $I = \{I_1, I_2, I_3, ..., I_M\}$ and X Non-IoT users denoted by the set $NI = \{NI_1, NI_2, NI_3, ..., NI_X\}$. Here, we consider each IoT device has a processing unit (i.e. microprocessors, FPGAs, SOCs, microcontrollers) and software applications which has some computational ability. The processing units and the software of the IoT devices are shown in Table 3 [41].

Table 3. Computational elements of IoT

Computational Elements of IoT devices	Samples
Hardware	Arduino, Raspberry Pi, Smart Things, Gadgeteer, Smart Phones, Phidgets, Intel Galileo, BeagleBone, Cubieboard.
Software	Operating System (TinyOS, Contiki, Riot OS, LiteOS, Android) Cloud (Hadoop, Nimbits, etc).

Due to the processing ability, each IoT user $I_m \in I$ can execute the tasks T_k^{IoT} locally. However, due to limited processing capability, the IoT users can offload the tasks to the CDC for processing via wireless communications. The Non-IoT users $NI_x \in NI$ must not have any processing ability and offload the task T_k^{NI} to the CDC for processing or storage purpose using wireless networks. The servers of each CDC have heterogeneous computational resources (e.g. CPU, memory) with fixed capacities. Hence, the computational capabilities of the server i is characterized by its computational service rate C_i (CPU capacity), and the computational service rate of the servers available in a CDC, $C_{CDC} = \sum_{i=1}^{N} C_i$. Each server can deploy multiple containers based on the requirements of the tasks. Here we consider two types of resources such as CPU and memory for deploying the containers. The CPU capacity of the container j (CP_j^{CN}) is defined as follows.

$$CP_j^{CN} = | CR_j | \times sizeof(CR) \tag{12}$$

where ($|CR_j|$) is the number of cores and the size of each core, denoted as $sizeof(CR)$. The memory capacity of the container j (CM_j^{CN}) is defined as follows.

$$CM_j^{CN} = | M_j | \times sizeof(M) \tag{13}$$

where ($|M|$) is the number of blocks in the memory and the size of each block, denoted as $sizeof(M)$. Note that our proposed CBSS algorithm is compatible with other network structure until unless the IoT/Non-IoT to CDC association is unchanged in one peer offloading decision cycle.

B. Task Arrival Model

The computational tasks are categorized in two different way- (i) real-time tasks (also known as IoT tasks) which are generated by the IoT devices and (ii) batch tasks (also known as non-IoT tasks), which are generated by the non-IoT devices. IoT devices may execute the tasks locally or immediately offload the tasks to the CDC for faster processing. However, the non-IoT tasks offload the tasks to the CDC as a batch mode. The operational timeline is decentralized for making peer offloading for non-IoT based tasks. In each time slot t, a batch of non-IoT tasks k is generated from non-IoT devices (e.g Users own Laptop, Desktop, mobile, etc) according to the Poisson process. This is a common assumption on the batch tasks arrival in a computational server [42]. Let P_k^t denotes the rate of a Poisson process for task generation in each time slot. At time t, P_k^t is randomly drawn from $P_k^t \in [0, P_{max}]$ to generate the temporal variation in task arrival pattern. Let $P^t = \{P_k^t\}_{k \in T^{NI}}$, denotes the non-IoT based task arrival at time slot t. Here, we assume that the size of the IoT and Non-IoT based tasks are measured by Million Instructions (MI) and the required CPU and memory capacity for the tasks is represented as Million Instructions Per Second (MIPS) and MB respectively. The total tasks are arrival rate at each server i, denoted by T_i^t, is $T_i^t = \sum_{k \in T^I, k \in T^{NI}} P_k^t$. The task arrival rate of all N servers in a CDC is denoted as $T_{CDC} = \{T_i^t\}_{i \in N}$.

C. Transmission Model

The transmission time is incurred during IoT and Non-IoT based devices offloading the tasks to the CDC for further computation or storage through an uplink channel or CDC transmit the computation result to the requesting devices through a downlink channel. In a noise-free channel, the transmission time between users

and CDC depends on the distance and the bandwidth of the network. The transmission time between the user a and CDC j (TT_{aj}) is defined as follows.

$$TT_{aj} = PD_{aj} + SD_{aj} : a \in M, a \in X \tag{14}$$

where PD_{aj} and SD_{aj} represent the propagation delay and serialization delay. The propagation delay (DS_{aj}) is defined as the ratio between the distance among the user a and CDC j and the bandwidth of the network (BW_{aj}), i.e. $PD_{aj} = \frac{DS_{aj}}{BW_{aj}}$, where $DS_{aj} = \sqrt{(X_a - X_j)^2 + (Y_a - Y_j)^2}$ in a two-dimensional space (X, Y). The serialization delay is the ratio between the sizes of a task k emitted from a device a, $SZ_{ka} \in [0, SZ_{max}]$ to the transmission rate of the network (TR_{aj}), i.e. $SD_{aj} = \frac{SZ_{ka}}{TR_{aj}}$. However, during data transmission, some noise may be added with the actual data, which may change our assumption. So, we can consider that each CDC j consumes a fixed transmission power (P_j), and the data are transmitted over an orthogonal channel, and the achievable transmission rate TR_{aj} between user a and CDC j is given by Shannon Capacity,

$$TR_{aj} = BW_{aj} log_2(1 + \frac{P_j H_{aj}^t}{\delta^2}) \tag{15}$$

where H_{aj}^t is the channel gain between user a and CDC j and δ^2 represents the noise power. Then, the amount of time required to uplink the data from the user a to CDC j (TT_{aj}^u) is $TT_{aj}^u = \frac{P_j SZ_{ka}}{TR_{aj}}$. However, the downlink traffic, from CDC j to user a, consists of the computational result $SZ_{jk} \in [0, SZ_{max}]$ and a few communication data a $CD_{aj} \in [0, CD_{max}]$. Then, total time requires to transmit the downlink traffic from CDC j to user a is $TT_{ja}^d = \frac{P_j(SZ_{kj} + CD_{aj})}{TR_{ja}}$, where TR_{ja} is the transmission rate of the downlink channel from CDC j to user a and SZ_{kj} is the size of the task k, emitted from CDC j. So, the total transmission time is the combination of uplink time to transmit data from users to CDC and users and the downlink time to forward the data from CDC to users, which is formulated as-

$$TT_{ak} = TT_{aj}^u + TT_{ja}^d = \frac{P_j SZ_a}{TR_{aj}} + \frac{P_j(SZ_j + CD_{aj})}{TR_{ja}} \tag{16}$$

D. Computational Model

The computational time of an IoT or Non-IoT based task depends on the capacity of the computational device and total transmission time required for transmitting the tasks and receiving the tasks from CDC which is defined as follow.

$$CT_{kc}^a = ET_{kc}^a + TT_{ak} : c \in \{CN, I\}, a \in \{I, NI\} \tag{17}$$

Here, ET_{kc}^a represents the execution time of a task k emitted from a device a in a computational device c, which is defined as $ET_{kc}^a = \frac{SZ_{ka}}{CP_c}$, where CP_c is the CPU capacity of a computational device c. Here, the computational device for IoT based tasks is either the IoT device itself or the containers $l \in U$ of a server $i \in N$. IoT devices have some limited processing and storage capacity to execute the small task, also referred to as local computing; otherwise they can offload the tasks to the server $i \in N$ of a CDC for further processing, also denoted as remote computing. However, the Non-IoT based devices must offload the tasks to the server $i \in N$ for remote computing and assigned the tasks to a suitable container $l \in U$ as per the requirements of the tasks. A task can able to execute in a computational device if the remaining resource capacity (also referred as the remaining load capacity) of the device meets the resource requirements of the task. The current load of a device is defined as the percentage of the resources have been utilized within a specific time interval (T' to T''). Here we consider each computing device is configured based on two types of resources *i.e.* CPU ($C_c(t)$) and memory ($M_c(t)$). The current load of a computational device c at each time interval is defined as follows.

$$L_c(C, M) = \beta \times \sum_{t=T'}^{T''} C_c(t) + (1 - \beta) \times \sum_{t=T'}^{T''} M_c(t) : c \in \{I, CN\} \tag{18}$$

where, β is the constant in the interval [0, 1]. The value of $C_c(t)$ and $M_c(t)$ are represented in the interval [0,100]. If the remaining load of a computational device c, $RL_c(C,M) = (1 - L_c(C,M))$is less than the required load ($TL_k(C, M)$) by the task $T_k: k \in \{M, X\}$, *i.e.* $RL_c(C,M) \leq TL_k(C,M)$), then the tasks should be assigned to that device. Otherwise, the task should be offloaded to CDC or wait in a local queue for further execution. The aim should be to keep the resource utilization of an individual device within the permissible level. Utilization is another key decision parameter used to indicate whether a device capacity is adequate to assign a new task or not. However, very high resource utilization often compromises the performance of the system and may increase the completion time of the tasks. For a given device *c*, the resource utilization, RU_c, is calculated as follows.

$$RU_c = \frac{L_c(C,M)}{L_c^{\max}(C,M)} \times 100 \tag{19}$$

where $L_c^{max}(C, M)$ indicates the maximum resource capacity of the device *c*.

Case 1: Local Computing: The IoT devices have limited processing capacity which can execute the small tasks locally. If the remaining load of an IoT device $c \in \{M\}$, is less than the required load of a task $T_k: k \in \{M\}$, *i.e.* $RL_c(C,M) \leq TL_k(C,M)$), then the task T_k should be executed locally. As the task executes locally, so the total transmission time of the task is 0, i.e. $TT_{ak} = 0$. So the overall computational time required by the task to complete its execution is equal to the execution time, i.e. $CT_{kc}^a = ET_{kc}^a + 0 = ET_{kc}^a$.

Case 2: Remote Computing: In this case, the long task of the IoT devices and all Non-IoT based tasks should be offloaded to the CDC. Here, we consider the cloud data center has the maximum resource capacity to execute any kind of tasks. The tasks will transmit over a wireless user interface which requires an uplink and downlink transmission time. The cloud administrator develops a container based on the resource requirements by a task and assigns the container to an available server $i \in N$ in a CDC. For offloading a task, the computational time consists of the total transmission time by the task to the selected computation server and the execution time of the task on the selected container, i.e. $CT_{kc}^a = ET_{kc}^a + TT_{ak}$.

E. Energy Model

The energy consumption of an IoT or Non-IoT based task mainly depends on two factors- (i) Computational energy: consumes energy based on the amount of time the computing resources are busy to execute the task and (ii) Transmission energy: produces energy based on the time required to transmit the task to the CDC and receive the computational results from CDC. The IoT devices may execute the small tasks locally which minimize the transmission time as well as the energy consumption for transmission time. However, for the long tasks of the IoT devices and the Non-IoT based tasks should be offloaded to the CDC which consumes computational energy and transmission energy for executing the tasks in CDC.

1) ***Computational Energy:*** The computational energy consumption varies greatly depending on the workload of the computing devices. The power consumption of a device is determined mainly by the processors and memory and the power consumption of the resources is determined primarily by resource utilization. However, the energy consumption of a device is estimated based on the utilization of the resources and the execution time of the tasks. The amount of energy consumed by the IoT device or the selected container $c \in \{M, U\}$ is defined as follows.

$$TP_c^C = \begin{cases} DP_c^C + SP_c^C : When\ CPU\ executed\ a\ task \\ SP_c^C : When\ CPU\ is\ active\ state\ without\ executing\ a\ task \\ 0 : When\ CPU\ is\ in\ idle\ state \end{cases} \tag{20}$$

where, DP_c^C and SP_c^C represents the dynamic power and static power consumption by the processors respectively. The dynamic power of a device depends on the frequency (f) and the voltage (V_{dd}) of the cores of the CPU while executing a task. The dynamic power consumption of a device is defined as $DP_c^C = \sum_{a=1}^{CR} \beta f V_{dd}^2$. Here *CR* represents the number of cores of the processor and β is a constant, where $\beta = C_l N \alpha$.; C_l and *N* represent the capacity and the number of logic gates available in each

core and α is constant where $\alpha \leq 1$. Similarly, the static power consumption of a CPU is defined as $SP_c^C = dV_{dd}$, where d is a constant. The power consumption of the memory depends on the read write operation while executing a task on an IoT device or a container which is defined as follows

$$TP_c^M = \begin{cases} DP_c^{MW} : When\ memory\ performs\ write\ operation \\ DP_c^{MR} : When\ memory\ performs\ read\ operation \\ DP_c^{MW} + DP_c^{MR} + A_M : When\ memory\ performs\ read\ and\ write\ operation \\ A_M : When\ memory\ is\ in\ idle\ state \\ 0 : When\ memory\ is\ in\ idle\ state \end{cases} \quad (21)$$

Where, DP_c^{MR} and DP_c^{MW} represent the dynamic power consumption by the memory while performing reading and writing operation and A_M represents the static power required by the memory during an ideal state. The dynamic power consumption by the memory during read and write operation is defined as $DP_c^{MW} = DP_c^{MR} = \frac{1}{2} c V_{dd}^2 f$, where c is a constant. So, the total power consumed by the device c in each unit time is defined as follows.

$$TP_c = TP_c^C + TP_c^M \quad (22)$$

So, the total computational energy (TCE_k) by a task T_k, executing in a computing device c, a container is $TCE_k = TP_c \times ET_{kc}^a$.

2) ***Transmission Energy:*** The transmission energy of a task depends on the amount of bandwidth consumed by the tasks during offload the task to the CDC through an uplink network and transmits back the task to the user through a downlink channel. Here, we consider the bandwidth consumes by the task for uplink and downlink channel between user a and CSC j is BW_{aj} and BW_{ja} respectively. The total time requires to transmit the data from the user a to CDC j and vice-versa is $TT_{aj}{}^u$ and $TT_{ja}{}^d$ respectively. So the total transmission energy (TTE_k) consumed by a task T_k is.

$$TTE_k = \left(BW_{aj} \times TT_{aj}^u\right) + \left(BW_{ja} \times TT_{ja}^d\right) \quad (23)$$

So, the total energy consumes by a task while T_k executing in a container in a CDC j is

$$TEC_{kj} = (TCE_k + TTE_k) = (TP_c \times ET_{kc}^a) + \left(BW_{aj} \times TT_{aj}^u\right) + \left(BW_{ja} \times TT_{ja}^d\right) \quad (24)$$

The power consumption of a device also affects two parameters- the CO_2 emission rate and the temperature of the device. The relationship between power consumption and CO_2 emission is an important factor to account in consideration of the sustainability of a device. The CO_2 emission rate of the CPU ($CE_c^C(t)$)and memory ($CE_c^M(t)$) of a device c at time t is defined as follows.

$$\begin{aligned} CE_c^C(t) &= \gamma_{CO_2}^C \times TP_c^C \\ CE_c^M(t) &= \gamma_{CO_2}^{mm} \times TP_c^M \end{aligned} \quad (25)$$

where, γ_{CO2}^C and γ_{CO2}^M are two user-defined constants. So, the total CO_2 emission rate of a device c is defined as follow.

$$TCE_c = CE_c^C(t) + CE_c^M(t) \quad (26)$$

Similarly, the relationship between the temperature and power emission of the CPU ($T_c^C(t)$) and memory ($T_c^M(t)$) of a device is defined as follows.

$$\begin{aligned} T_c^C(t) &= \sqrt{\frac{TP_c^C}{\alpha V_{dd}}} \\ T_c^M(t) &= \sqrt{\frac{TP_c^M}{\alpha V_{dd}}} \end{aligned} \quad (27)$$

where α is a user-define constant. So, the total temparature (TM_c) emitted by a device c is defined as follow.

$$TM_c = T_c^C + T_c^M \quad (28)$$

5.2 Problem Statement

The main intention of the container-based scheduling strategy is to minimize the start-up time of the virtual resources as a form of containers for the tasks which can reflect the performance of the CDC including overall computation time, energy consumption and overall resource utilization. In this work, the tasks may execute locally (small IoT-based tasks) or should be offloaded (i.e. large IoT or Non-IoT based tasks) to the CDC for further processing and analyzing. A suitable container $l \in U$ is deployed for each task based on the multiple QoS parameters including computation time and energy consumption. The selected containers are assigned to an optimal computational server $i \in N$ based on the availability of the resources and the resource requested by the containers for processing. There are multiple indicators and objectives that can offload the tasks to the appropriate computational server and execute the tasks efficiently [43-44]. The main focus of the paper is to minimize the energy consumption and computational time of the tasks while maximizing the resource utilization of the computing devices. These objectives can be achieved by executing the tasks locally or offloading to the CDC. Consequently, the optimization of the tasks is formulated as a bi-objective optimization problem which is discussed below.

Minimize TEC_{kj}
Minimize $CT_{kc}{}^{a}$

Subject to

$TEC_{kj} \leq \theta_1$ .....(i)
$TCE_c \leq \theta_2$ ..... (ii)
$TM_c \leq \theta_3$ .....(iii)
$L_i(C, M) \leq \theta$ ..... (iv)
$L_c(C, M) \leq L_c^{max}(C, \mathrm{M})$ ..... (v)
$RU_i \leq 100$ ..... (vi)
$x_{kc} \in \{0, 1\};\ k \in \{M, X\},\ c \in \{M, U\}$ ..... (vii)
$\sum_{i=1}^{c} x_{ik}\ ;\ k \in \{M, X\}, c \in \{M, U\}$ .....(viii)

Constraint (i) indicates that the total power consumption of a device *c* must not beyond a threshold value θ_1. Constraint (ii) represents that the total CO_2 emission rate of a device *c* must not beyond a threshold value θ_2. The CO_2 emission rate is linearly proportion with the energy emission of the computing devices which is depicted in Eq. (25) and Eq. (26). Constraint (iii) indicates that the total temperature consumption of a device *c* must not beyond a threshold value θ_3. Temperature emission rate is linearly proportion with the energy emission of the computing devices which is depicted in Eq. (27) and Eq. (28). Constraint (iv) indicates that a load of a computing device *c* must not beyond a threshold value θ. Constraint (v) represents that a load of a computing device *c* must not beyond the maximum capacity of that server. Constraint (vi) indicates that the resource utilization of a computing device *c* must not beyond 100%. Constraint (vii), x_{kc} denotes whether a task T_k is assigned to a computational device *c*. Constraint (viii) describes that each computational server can execute multiple numbers of tasks by multiple containers $l \in U$ concurrently.

6. Energy-Efficient Container-based Scheduling (EECS) Strategy

In this section, the details of our proposed EECS strategy are described. Its three main components, i.e., selection of executing component and optimal server selection, are respectively introduced in the following sub-sections. At last the complete algorithm of the EECS strategy is also provided.

6.1 Selection of Executing Component

Here, we discuss a selection strategy of a suitable executing component for each task using the APSO technique based on the multiple objectives of the task scheduling strategy. As we discussed earlier that some of the IoT-based tasks may execute locally if the micro-services of the IoT devices meet the resource requirements of the tasks; otherwise the tasks should be offloaded to the CDC for further processing in a suitable container or VM instances. One of the common approaches for solving MOPs is a weighted-sum method, which is introduced by Zadeh [45]. This method linearly aggregates all the individual objective function of a MOP into one objective by using a weight vector. Here, we apply a weighted-sum method to

design an objective function to find a suitable executing device (i.e. containers or VM instances) for each task based on the APSO technique. Let's assume that the executing devices are represented as particles which are represented as $D = \{d_1, d_2, d_3, ..., d_Z\}$ includes Z particles. Each particle has a position x_z and velocity v_z $\{z= 1, 2, 3, ..., Z\}$ in a 2-D space. Initially, this phase generates a multi-objective optimization function (referred as Fitness Function) based on the two QoS parameters i.e. total energy consumption and the computational time, defined in Section 5.2. The total energy consumption of the executing devices are denoted as

$$\boldsymbol{TEC_z}(d_z, D) = \{TEC_1, TEC_2, TEC_3, TEC_4, ..., TEC_z\}, \text{ where } z \in \{U, H\}$$

and the computational time of those are represented as

$$\boldsymbol{CT_z}(d_z, D) = \{CT_1, CT_2, CT_3, CT_4, ..., CT_z\}, \text{ where } z \in \{U, H\}$$

The Fitness Function places all the two QoS parameters into a single one based on weighted-sum approached which is defined below.

$$fit(d_z, D) = \alpha_1 . TEC_z(d_z) + \alpha_2 . CT_k(d_z) \tag{29}$$

Thus the scheduling objectives are aggregated in a single Fitness Function, which measures the degree of the optimal executing device. In Eq. (29), $fit_z(d_z, D)$ represents the Fitness solution of the selection of the suitable executing device $k \in \{U, H\}$ for each requested task or application. In the above equation, the coefficients α_1, and α_2 are the weights which are used to indicate the priority of the objectives whose value lies between [0, 1]. During finding an optimal executing device each objective of particle d_z is normalized based on the maximum and minimum values of the corresponding objective function. Such normalized objective function helps to eliminate the impact of various amplitudes on multi-objectives. The normalized objective $FN_r(d_i)$ (where r represents the total number of objectives) of d_i are obtained using

$$FN_r(d_i) = \frac{f_r(d_i) - f_r^{\min}}{f_r^{\max} - f_r^{\min}} \tag{30}$$

Where f_r^{max} and f_r^{min} represent the maximum and minimum values of the r^{th} objective which are obtained from the non-dominated solutions. The particles are placed in a 2-D space based on the two objectives (total energy consumption and computational time) of the given problem using Eq. (29), shown in Fig. 6(a). While running the APSO technique, all the normalized particles are generated in the 2-D plane and converge to a single point of the plane, which is shown in Fig. 6(b) and Fig. 6(c) respectively. Finally, the gbest location of the converged particle finds the best-fit executing device using the Euclidean distance (ED) among the nearest neighboring particles in that plane which is shown in Fig. 6(c). Finally, the APSO algorithm pointed the best-fit executing device in the plane with minimum ED value, shown in Fig. 6(d) and is defined as

$$gbest(d_i) = \min_{j \neq i} \sqrt{\sum_{x=1}^{2} ED(f_r(d_i), f_r(d_j))} = \sqrt{\sum_{x=1}^{2} (f_r(d_i) - f_r(d_j))^2} \tag{31}$$

where ED value $ED(d_i, d_j)$ between two particles d_i and d_j in 2-D space is defined as

$$ED(f_r(d_i), f_j(d_i)) = \begin{cases} f_r(d_j) - f_r(d_i) \ \textit{if} \ (f_r(d_j) > f_r(d_i)) \\ 0 \qquad\qquad\qquad \textit{Otherwise} \end{cases} \tag{32}$$

The above total energy consumption and computational time are dynamically balancing two weight factors α_1 and α_2. Those factors are adaptively adjusted for different particles based on the objectives. For this purpose, the average value of all $TEC_z(d_z, D)$ and $CT_z(d_z, D)$ in the particle swarm d_z are calculated by

$$\overline{TEC}(d_z, D) = \frac{\sum_{i=1}^{Z} TEC(d_i, D)}{Z} \ \textit{and} \ \overline{CT}(d_z, D) = \frac{\sum_{i=1}^{Z} CT(d_i, D)}{Z} \tag{33}$$

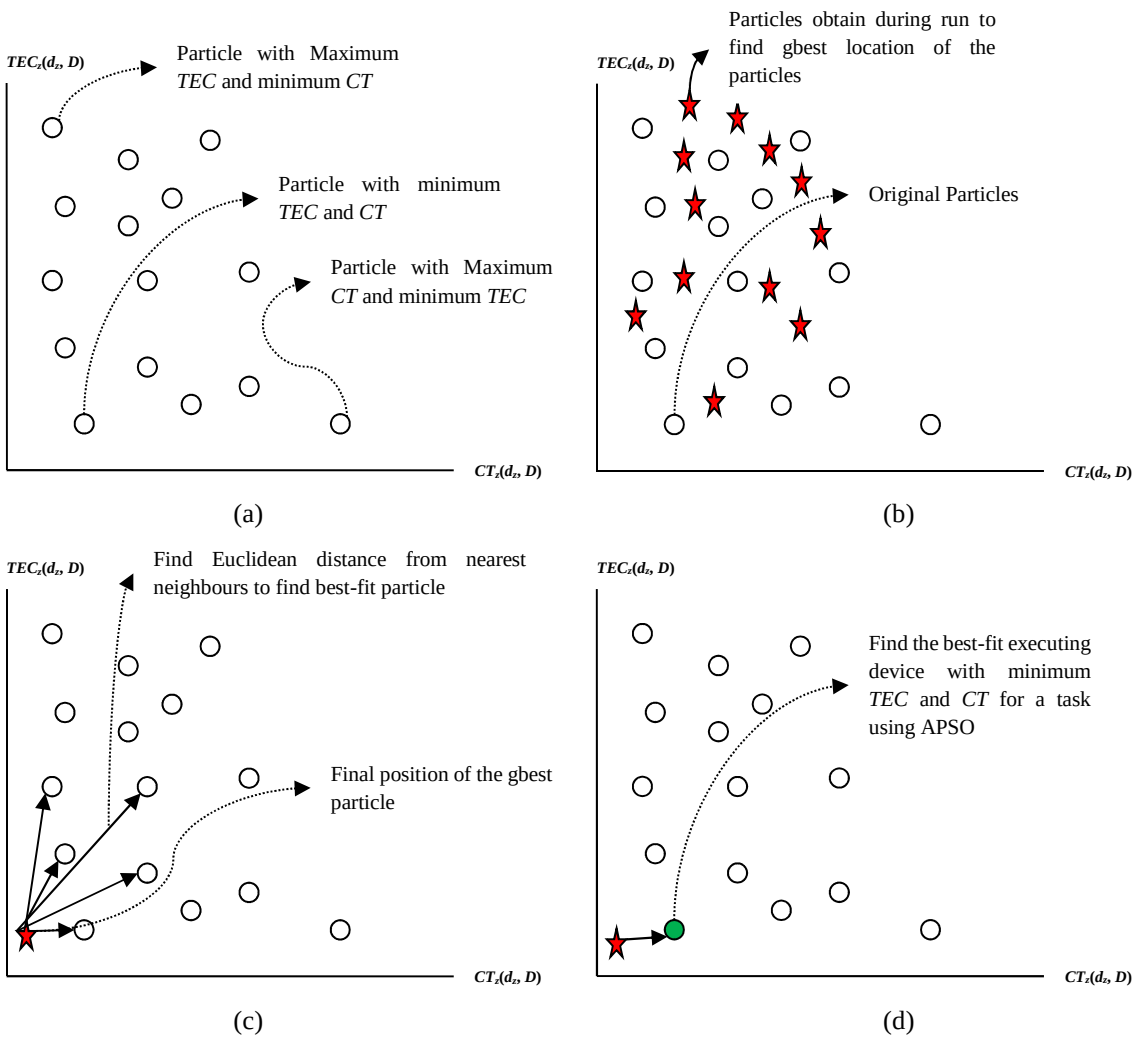


Fig. 6. The process of finding the best-fit executing device using multiple objectives based on APSO technique

The main idea of this phase of EECS strategy is to use a weighted-sum approach to define the fitness function, defined in Eq. (29). However, different particles should be assigned in the 2-D plane based on the various weight of α_1 and α_2, in order to improve some potentiality for a superior solution instead of the poorly converged ones. Therefore, using $\overline{TEC}(d_z, D)$ and $\overline{CT}(d_z, D)$, four situations of the particles are considered for adjusting the values of α_1 and α_2, aiming to properly balance the value of $TEC_z(d_z, D)$ and $CT_z(d_z, D)$. The example of different weights of α_1 and α_2 for various cases is shown in Fig. 7.

Case I: {The particle d_i with $TEC_z(d_z, D) < \overline{TEC}(d_z, D)$ and $CT_z(d_z, D) > \overline{CT}(d_z, D)$ }: In this category, all the normalized particles have minimum total energy consumption with maximum computational time. For better convergence, the values of the weighted factors set as $\alpha_1 = [0.7, 1]$ and $\alpha_2 = [0, 0.3]$.

Case II: {The particle d_i with $TEC_z(d_z, D) > \overline{TEC}(d_z, D)$ and $CT_z(d_z, D) < \overline{CT}(d_z, D)$ }: In this category, all the normalized particles have maximum total energy consumption with minimum computational time. For better convergence, the values of the weighted factors set as $\alpha_1 = [0, 0.3]$ and $\alpha_2 = [0.7, 1]$.

Case III: {The particle d_i with $TEC_z(d_z, D) < \overline{TEC}(d_z, D)$ and $CT_z(d_z, D) < \overline{CT}(d_z, D)$ }: In this category, all the normalized particles have minimum total energy consumption and computational time. For better convergence, the values of the weighted factors are equally distributed among them and set as $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$.

Case IV: {The particle d_i with $TEC_z(d_z, D) > \overline{TEC}(d_z, D)$ and $CT_z(d_z, D) > \overline{CT}(d_z, D)$ }: In this category, all the normalized particles have maximum total energy consumption and computational time. For better convergence, the values of the weighted factors are set as $\alpha_1 = [0, 0.5]$ and $\alpha_2 = [0, 0.5]$.

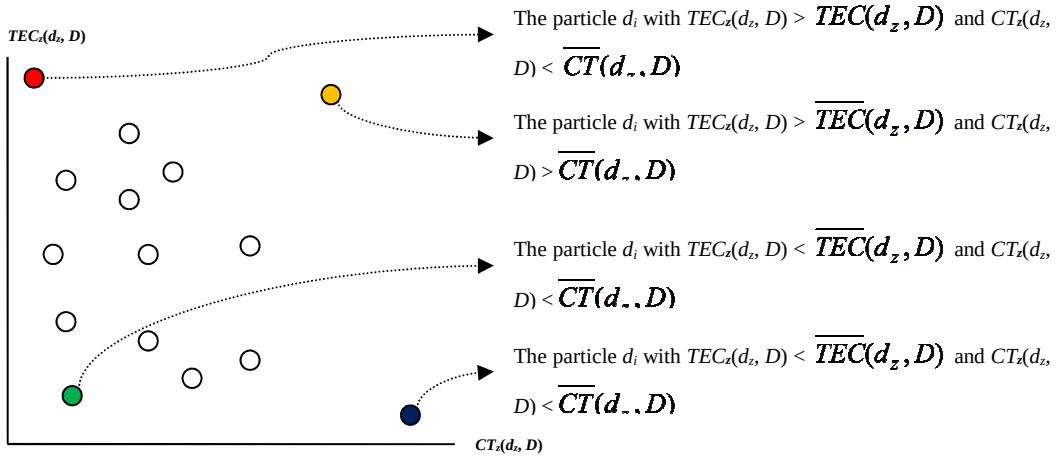


Fig. 7. Example of various cases for adjusting the values of α_1 and α_2

Finally, the velocity and the position of the particles of the APSO technique need to be updated using their positional information of the local and global best positions. To reduce the evolutionary distance between the local-best particles and global-best one while making more disturbances, a velocity updated equation is designed as follows.

$$V_{z+1} = V_z + \alpha\varepsilon + \beta(X_z^* - X_z) \tag{34}$$

where, $\boldsymbol{\varepsilon}$ is a random vector uniformly distributed in the range [0, 1] and α, β are the user-defined constraints. X_z^* and X_z represents the global-best position of a swarm and the position of the swarm in the previous iteration respectively. This velocity update helps to guide the particles to search toward the global-best particles. The updated location of a single particle to increase the convergence even further using one step is formulated as

$$X_{z+1} = (1-\beta)X_z + \beta X^* + \alpha\varepsilon^l \tag{35}$$

Thus, it is expected to enhance the convergence speed of selection of executing component of EECS strategy. The pseudo code of selection of executing component of EECS strategy is shown in Fig. 8.

EECS Strategy: Selection of Executing Component

Input: Set the objectives of the computational devices; Set the user defined constants: n, α, β, and $\boldsymbol{\varepsilon}$
Output: Find a suitable computational device

```
1:  Begin
2:  For each j: 1 to z  do                    // r = Maximum number of iterations
3:      fit(dz, D)  = Initial Solution()
4:  End For
5:  While (fit(dz, D)  <= Δ) do               // Δ= Threshold value
6:      min = arg min fit(dz, D)
7:        For each i: 1 to z do
8:            For each  j: 1 to z do
9:                \\ Calculate the modified attractiveness function
                        FN_r(d_i) = (f_r(d_i) − f_r^min) / (f_r^max − f_r^min)
10:               If (fj(di) >  fi(di))
```

11: \\ Find the Euclidean distance between the servers

$$ED(f_r(d_i), f_j(d_i)) = \begin{cases} f_r(d_j) - f_r(d_i) \ if \ (f_r(d_j) > f_r(d_i)) \\ 0 \qquad\qquad\qquad Otherwise \end{cases}$$

12: Set the values of α_1 and α_2 for the particles.

13: Compare the Pareto dominance relationship between each pair of the computational devices and put the non-dominated ones.

14: Randomly select the executing devices from the set of swarms.

15: \\ Update the velocity and position of the particles

$$V_{z+1} = V_z + \alpha\varepsilon + \beta(X_z^* - X_z)$$

$$X_{z+1} = (1-\beta)X_z + \beta X^* + \alpha\varepsilon^l$$

16: **End if**

17: **End for**

18: **End for**

19: Find the best-fit executing device for each task or application

20: **End for**

21: **End**

Fig. 8. The pseudo code of selection of executing component of EECS strategy

6.2 Optimal Server Selection

The main contribution of this phase is to select an optimal loaded computing server with available resources for deploying the selected executing devices (containers or VM instances) $c \in \{U, H\}$. The current load of each computing server $i \in N$ depends on the current CPU and memory usage by various executing devices which is defined in Eq. (18). The remaining load of the computing server $i \in N$ is defined as $RL_i(C, M) = (1 - L_i(C, M))$ and the remaining CPU and memory usage of the computational servers is represented as $C_i^R(t)$ and $M_i^R(t)$ respectively. This phase sort the remaining loads of the computing servers and select the server with minimum load, i.a e.

$S_i^{optimal}$ = **MAX** {$RL_1(C, M)$, $RL_1(C, M)$, $RL_1(C, M)$, ..., $RL_N(C, M)$}

A selected executing device $c \in \{U, H\}$ is assigned to the optimal server with required resources ($C_c(t)$, $M_c(t)$) at time t based on the following rules:

Rule 1: {If (**MAX**$(RL_i(C, M): i \in N) \wedge C_i^R(t) \geq C_c(t) \wedge M_i^R(t) < M_c(t))$}: The computing server $i \in N$ is a minimum loaded server which satisfies the CPU requirements of the executing device $c \in \{U, H\}$, however, fails to meet memory usage requirements. The executing device $c \in \{U, H\}$ needs to wait until sufficient memory is available.

Rule 2: {If (**MAX**$(RL_i(C, M): i \in N) \wedge C_i^R(t) < C_c(t) \wedge M_i^R(t) \geq M_c(t))$}: The computing server $i \in N$ is a minimum loaded server which satisfies the memory requirements of the executing device $c \in \{U, H\}$, however, fails to meet CPU usage requirements. The executing device $c \in \{U, H\}$ needs to wait until sufficient CPU is available.

Rule 3: {If (**MAX**$(RL_i(C, M): i \in N) \wedge C_i^R(t) < C_c(t) \wedge M_i^R(t) < M_c(t))$}: The computing server $i \in N$ is a minimum loaded server which does not has sufficient CPU and memory usage for assigning the executing device $c \in \{U, H\}$, The executing device $c \in \{U, H\}$ needs to wait until sufficient CPU and memory are available.

Rule 4: {If (**MAX**$(RL_i(C, M): i \in N) \wedge C_i^R(t) \geq C_c(t) \wedge M_i^R(t) \geq M_c(t))$}: The computing server $i \in N$ is a minimum loaded server which satisfies the CPU and memory requirements both of the executing device $c \in \{U, H\}$. The executing device $c \in \{U, H\}$ is immediately assigned to the computing server $i \in N$.

The pseudo code of the optimal server selection of EECS strategy is shown in Fig. 9.

EECS: Optimal Server Selection

Input: Set of executing devices $c \in \{U, H\}$, remaining loads $RL_i(C, M)$ of the computational servers $i \in N$

Output: Select the optimal server

1. **Begin**
2. **For each** computational servers *i*: 1 to *N*
3. **For each** computational servers *j*: 1 to *N*
4. Calculate loads of the computational servers

$$L_c(C,M) = \beta \times \sum_{t=T'}^{T''} C_c(t) + (1-\beta) \times \sum_{t=T'}^{T''} M_c(t)$$

5. Calculate remaining loads of the servers

$$RL_i(C,M) = (1 - L_i(C,M))$$

6. Sort the servers and selects the minimum loaded servers with maximum reaming load

$$S_i^{optimal} = \mathbf{MAX}\ \{RL_1(C, M), RL_1(C, M), RL_1(C, M), ..., RL_N(C, M)\}$$

7. **End for**
8. **If** (MAX($RL_i(C,M)$: $i \in N$) $\wedge\ C_i^R(t) \geq C_c(t) \wedge M_i^R(t) < M_c(t)$)
9. Executing device $c \in \{U, H\}$ needs to wait
10. **If** (MAX($RL_i(C,M)$: $i \in N$) $\wedge\ C_i^R(t) < C_c(t) \wedge M_i^R(t) \geq M_c(t)$)
11. Executing device $c \in \{U, H\}$ needs to wait
12. **If** (MAX($RL_i(C,M)$: $i \in N$) $\wedge\ C_i^R(t) < C_c(t) \wedge M_i^R(t) < M_c(t)$)
13. Executing device $c \in \{U, H\}$ needs to wait
14. **If** (MAX($RL_i(C,M)$: $i \in N$) $\wedge\ C_i^R(t) \geq C_c(t) \wedge M_i^R(t) \geq M_c(t)$)
15. Immediately deploy the executing device $c \in \{U, H\}$ to the server $i \in N$
16. **End for**
17. Find CT_k: $c \in \{M, X\}$ and RU_i: $i \in N$
18. **End**

Fig. 9. The pseudo code of optimal server selection of EECS strategy

Lemma: *The worst case and the best case time complexity of the EECS strategy is* $O(K^2r)$ *and* $O(Kr \log(K))$ *respectively.*

Proof: Let *K* be the total number of executing devices such as containers and VM instances. Step 1 to Step 21 of the selection of executing component of EECS strategy requires $O(K^2r)$ time (in the worst case) or $O(Kr \log(K))$ time (in the best case) to find the optimal executing device on the fly. This algorithm has two inner loops when going through population *K* and one outer loop for iteration *r*. So the complexity of the extreme case is $O(K^2r)$. As *K* is small (typically, *K*= 40 to 100) and *r* is large (*r*= 5000), the computation cost of this phase is relatively inexpensive because the complexity of this phase is linearly proportional to *r*. If *K* is relatively large, the complexity of the strategy is going to be $O(Kr \log(K))$. Let, *N* is the total number of computing servers in a CDC. So, for the optimal server selection of EECS strategy, the total time complexity of Step 1 to Step 18 is $O(N^2)$. The total time complexity of EECS strategy for best case is- $O(Kr \log(K)) + O(N^2) = O(Kr \log(K))$ (value of $K >> N$). The total time complexity of EECS strategy for worst case is- $O(K^2r) + O(N^2) = O(K^2r)$ (value of $K >> N$)

7. Performance Analysis

In this section, we investigate the performance of the proposed EECS strategy by evaluating various QoS parameters such as computational time, energy consumption, CO_2 emission, Temperature emission, and resource utilization. We further compare the proposed method with existing algorithms proposed in [20], [21], and [32].

7.1 Simulation Setup

The simulation parameters of the proposed EECS strategy are summarized in Table 4. Here we consider 100 IoT and 100 Non-IoT devices and each the devices are capable to generate multiple tasks or applications

simultaneously. Each IoT device has some processing and storage capacity which is randomly and uniformly taken as [1000-4000] MIPS and [256-512] MB respectively. Each IoT and Non-IoT device randomly generates multiple numbers of tasks with sizes lay in [7000-100000] MI. We assume that each computing device is equipped with three different wireless interfaces, Long Term Evolution (LTE), Wifi and Bluetooth. The IoT and Non-IoT devices may use the LTE connection for long-term communication such as the cloud data center, while they use Wifi and Bluetooth interfaces to connect with the other IoT and Non-IoT devices. Here, we assume that the transmission rate of the LTE and Wifi interfaces are randomly and uniformly distributed over [4.85, 6.85] Mbps and [2.01, 4.01] Mbps respectively. Here, we consider two cloud data center where each of them has enough processing and storage capacity. The CPU capacity and memory usage of each cloud server lay in [12000-30000] MIPS and [2018-4096] MB respectively.

Table 4: Simulation parameters

Parameter Description	Values
Number of IoT devices	100
Number of Non-IoT devices	100
Number of tasks or applications	1000
Sizes of the IoT applications	7000- 100000 MI
Number of Containers	100
Number of Servers	10
Number of cloud data centers	2
CPU Capacity of IoT devices	1000- 4000 MIPS
RAM Size of IoT devices	256-512MB
CPU Capacity of each computing server	25000- 45000 MIPS
RAM Size of each computing server	2048-4096MB
Storage capacity of each computing server	100 GB

7.2 Dataset Used

In this section, we discuss the dataset used to evaluate the performance of the proposed method. Here we generate six different synthetic datasets where each of the datasets randomly generates the energy consumption and the computational time for the executing devices as a form of containers and VM instances. The two important parameters of the APSO algorithm are α, β and we assumed their possible values of the parameters are in the range of [0, 1], where the expected possible values of those parameters are selected randomly such as 0.25, 0.45, 0.55, 0.65, 0.85 and 1.0. In the experimental purpose, we consider the size of the populations lie in [0, 100] and the possible values of the population values are selected randomly such as 20, 40, 50, 60, 80, and 100. An empirical test is performed to fix the values of the parameter. The proposed algorithm is simulated with a fixed value of each parameter for 100 iterations and minimum, maximum and average error are recorded with 1000 independent runs. In this experiment, for finding the best-possible value of a single parameter of APSO technique other parameters are in the constant state.

\

A fitness value of each pattern in a dataset is calculated based on two QoS parameters, shown in Eq. (29) and performs a significance test between the dataset to measure the reality of the data. The significant level of the datasets is measured based on *P*-value analysis with an unpaired *t*-test. The unpaired *t*-test assumes that the data have been sampled from the normally distributed population. The *P*-value is the probability of finding a result equal to or more extreme than the observed data when the null hypothesis (H_0) is true. *P*-value is less than the selected significance level then the null hypothesis is rejected and supported the alternative hypothesis with proper evidence. The choice of significance level at which we reject H_0 is arbitrary. Conventionally the 5%, 1% and 0.1% ($P < 0.05$, 0.01 and 0.001) levels have been used. Most researchers refer to statistically significant $P < 0.05$ and statistically highly significant $P < 0.001$. From Table 5, it is clearly observed that all the datasets are highly significant. So we can conclude that all the datasets are valid.

Table 5. Significant test of the datasets (DSs) based on *P*-value

	DATASET- 1	DATASET- 2	DATASET- 3	DATASET- 4	DATASET- 5	DATASET- 6
DS- 1	0	4.16421E-08	3.5324E-08	6.64E-07	0.000287	3.1346E-07

DS- 2	1.2539E-07	0	2.35E-07	2.46E-07	7.37E-10	6.64E-07
DS- 3	4.22E-11	1.24E-07	0	1.25395E-07	3.43E-10	3.1346E-07
DS- 4	1.24E-07	2.46E-07	3.24E-07	0	3.64221E-08	4.16421E-08
DS- 5	3.43E-10	3.64221E-08	6.12E-11	7.37E-10	0	0.000287
DS- 6	1.2539E-07	2.46E-07	2.35E-07	6.64E-07	7.37E-10	0

7.3 Parameter analysis and discussion

In this section, we analyze the values of the parameters of the proposed EECS strategy for observing the better quality of the solutions. For fixing the values of the parameters of the EECS strategy, we calculate the minimum, mean and maximum error of the populations. Here, we use Euclidean distance for calculating the minimum error (*MinErr* (D^*, d_z)) and maximum error (*MaxErr* (D^*, d_z)) between the best-position of a particle (D^*) and the other particles (d_z), $z \in Z$ in the dataset, which is shown below.

$$MinErr(D^*, d_z) = \min(ED_i(D^*, d_z)) = \min(\sqrt{\sum_{i=1}^{2}(D^* - d_z)})$$
$$MaxErr(D^*, d_z) = \max(ED_i(D^*, d_z)) = \max(\sqrt{\sum_{i=1}^{2}(D^* - d_z)}) \quad (36)$$

The mean error ($\overline{Err}(D^*, d_z)$) of the c, $c \in \{U, H\}$ number of executing devices is calculated as follow.

$$\overline{Err}(D^*, d_z) = \frac{(ED_z(D^*, d_z)}{c} = \frac{\sqrt{\sum_{i=1}^{2}(D^* - d_z)}}{c} \quad (37)$$

Analysis of β value: Beta parameter of the proposed algorithm is one of the most important parameters and the performance of the proposed EECS strategy varies for the different values of the parameter. The fluctuation in the performance due to the different values of the beta parameter can be easily observed in Table 6 and Fig. 10. In Table 6, the minimum error for different values of Beta is shown. All the dataset used for the experiment with the proposed method produces the minimum value if the beta value is set to 0.55. Hence, for the rest of the experiment, the beta value is fixed at 0.55.

Table 6. Minimum, mean and maximum error of β

Dataset		**Error**					
		β = 0.25	**β = 0.45**	**β = 0.55**	**β = 0.65**	**β = 0.85**	**β = 1.0**
DS- 1	**Minimum**	2.77127E-05	9.02E-05	7.66E-06	0.000268	0.000291	0.000427
	Mean	0.60433687	0.693567	**0.584586**	0.567229	0.684708	0.610959
	Maximum	0.90433678	1.293556	0.784575	1.167231	0.984712	0.920769
DS- 2	**Minimum**	7.45E-05	0.000327	6.38E-05	7.27E-05	0.000215	1.22E-05
	Mean	2.386025	2.192138	**2.524406**	2.142301	2.698928	2.479959
	Maximum	2.871234	2.834512	2.613425	2.943161	2.814531	2.825432
DS- 3	**Minimum**	0.000301	6.47E-06	1.18E-05	8.22E-06	3.28E-05	8.9E-05
	Mean	1.594098	1.685053	**1.748093**	1.613907	1.685876	1.547047
	Maximum	1.892132	2.0564321	1.814321	1.919071	2.134325	2.013245
DS- 4	**Minimum**	0.000126	2.71E-05	1.31E-05	3.13E-05	0.000361	0.000168
	Mean	0.446152	0.735002	**0.542211**	0.530446	0.735819	0.428055
	Maximum	1.032456	1.412313	0.900123	1.212365	.9934210	1.1045236
DS- 5	**Minimum**	4.16E-05	3.375E-05	3.75E-05	0.000348	0.000142	0.000149
	Mean	0.295197	0.333111	**0.173401**	0.307142	0.479175	0.126886
	Maximum	1.234123	1.412423	0.985423	.9912341	1.104231	1.045231
DS- 6	**Minimum**	5.22E-05	2.47E-05	2.18E-05	0.000191	7.16E-05	6.61E-05
	Mean	0.40563	0.691009	**0.407218**	0.377485	0.443640	0.443048
	Maximum	1.102341	0.994216	.9721650	1.132456	0.991234	1.012363

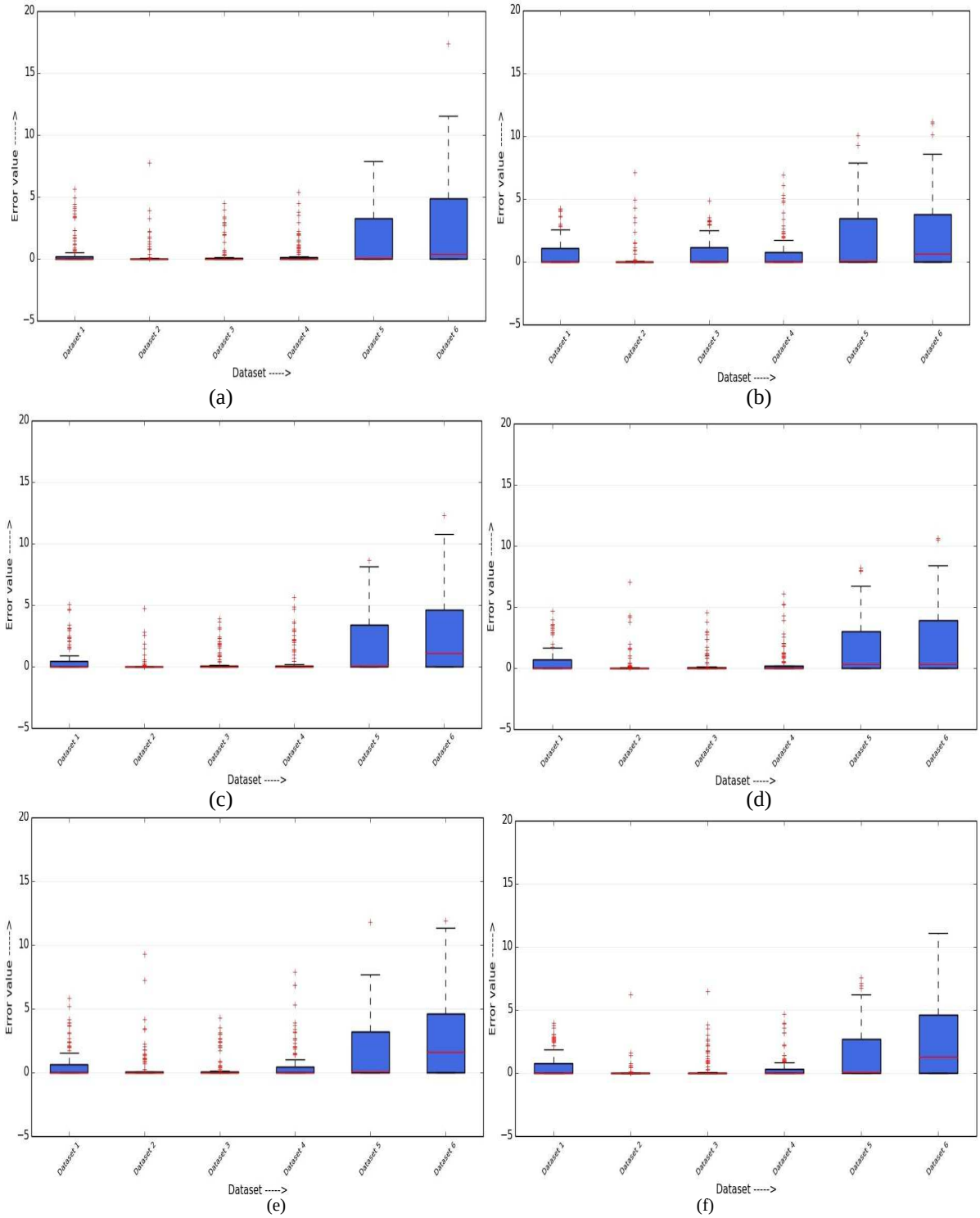
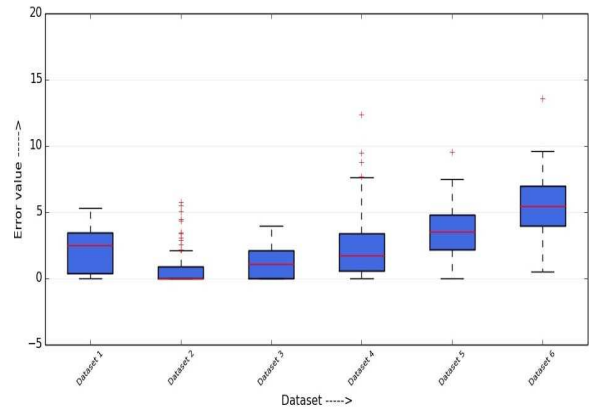


Fig. 10. Box Plot of Error for different value of β (a) β =.25 (b) β =.45 (c) β =.55 (d) β =.65 (e) β =.85 (f) β =1.0

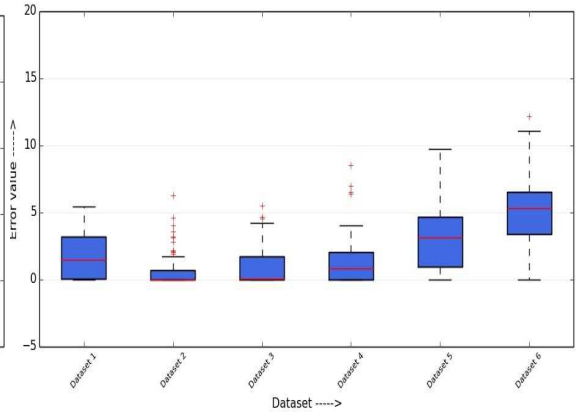
Analysis of α value: Alpha parameter of the proposed algorithm is one of the most important parameters and the performance of the proposed EECS strategy varies for the different values of this parameter. The fluctuation in the performance due to the different values of the Alpha parameter can be easily observed from Table 7 and Fig. 11. In Table 7, the minimum error for different values of Alpha is shown. All the dataset used for the experiment with the proposed method produces the minimum value if the alpha value is set to 1.0. Hence, for the rest of the experiment, the alpha value is fixed at 1.0.

Table 7. Minimum, mean and maximum error of α

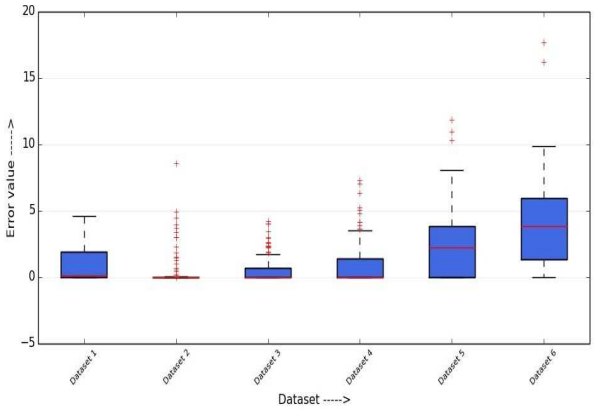
Dataset		Minimum Error					
		α = 0.25	α = 0.45	α = 0.55	α = 0.65	α = 0.85	α = 1.0
DS- 1	**Minimum**	0.000124	7.84E-05	0.00023	6.94E-05	0.000178	**5.14E-06**
	Mean	1.298154	0.824462	0.62336	0.742566	0.474447	0.357432
	Maximum	1.534123	1.412423	0.985423	.9912341	1.104231	1.045231
DS- 2	**Minimum**	3.38124E-05	3.81E-05	4.08E-05	6.41E-05	4.38E-05	**3.26E-05**
	Mean	2.263859796	1.340141	1.112783	1.19726	1.09794	0.910373
	Maximum	2.90433678	1.693556	1.784575	1.217231	1.584712	0.940769
DS- 3	**Minimum**	0.001088	0.000267	0.000427	5.4E-05	0.000153	**2.18E-05**
	Mean	3.508792	2.700784	2.489331	1.985265	1.899956	1.628176
	Maximum	4.102341	2.994216	2.9721650	2.132456	1.991234	1.912363
DS- 4	**Minimum**	4.91E-05	0.000134	8.24E-05	4.76E-05	2.49E-05	**4.82E-06**
	Mean	0.714127	0.735838	0.474563	0.339181	0.094042	0.190216
	Maximum	1.271234	1.034512	0.913425	0.943161	0.914531	0.825432
DS- 5	**Minimum**	0.528811	0.00031	0.000177	0.000149	0.000304	**0.000255**
	Mean	5.531986	4.43705	4.015667	3.73585	3.175658	2.928514
	Maximum	5.892132	5.0564321	4.814321	3.919071	3.634325	3.013245
DS- 6	**Minimum**	0.00026	6.95E-05	0.000212	7.41E-05	1.56E-05	**5.28E-06**
	Mean	2.33300	1.327611	1.052162	0.746555	0.552767	0.706386
	Maximum	2.932456	1.412313	1.400123	1.212365	.9934210	0.9045236



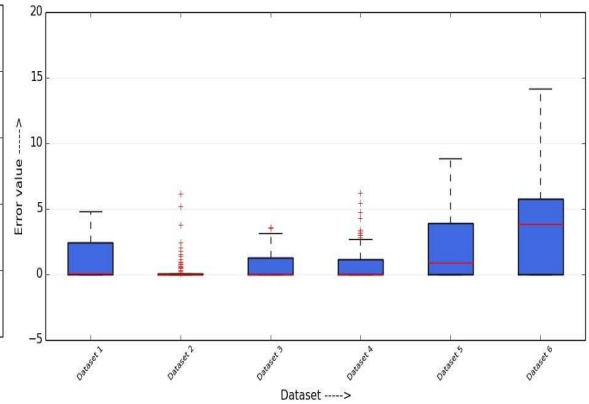
(a)



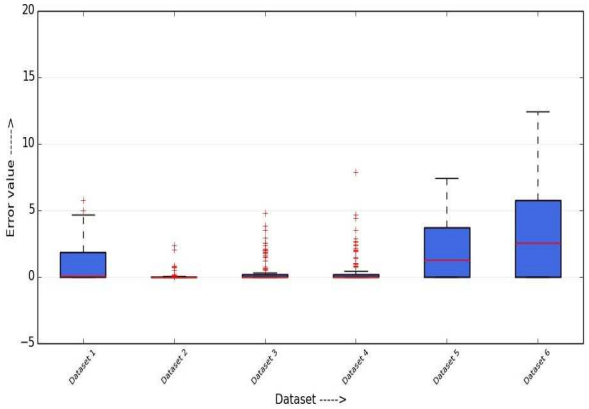
(b)

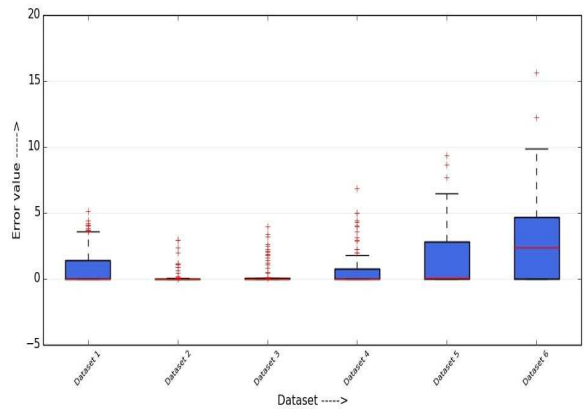


(c)



(d)





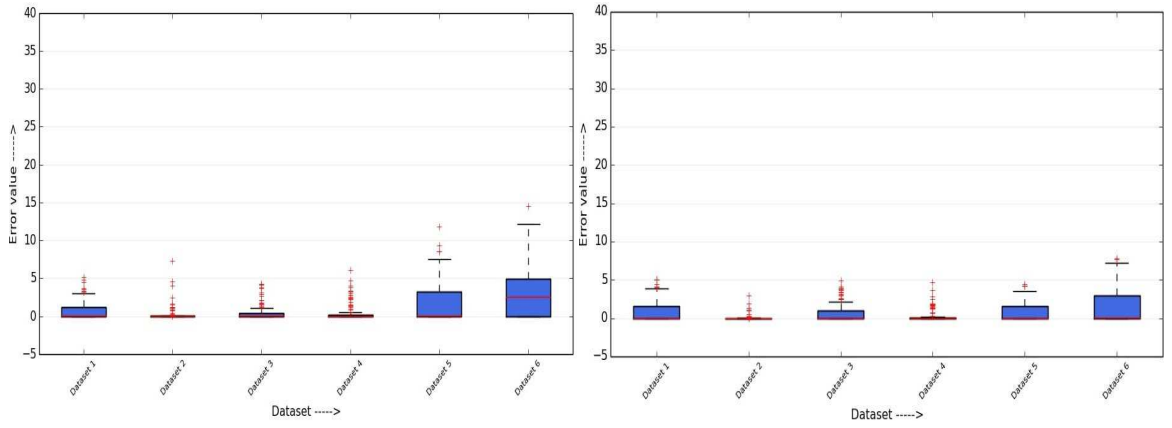
(e) (f)

Fig. 11. Box plot of error for different value of α (a) α = 0.25 (b) α = 0.45 (c) α = 0.55 (d) α = 0.65 (e) α =0.85 (f) α =1.0

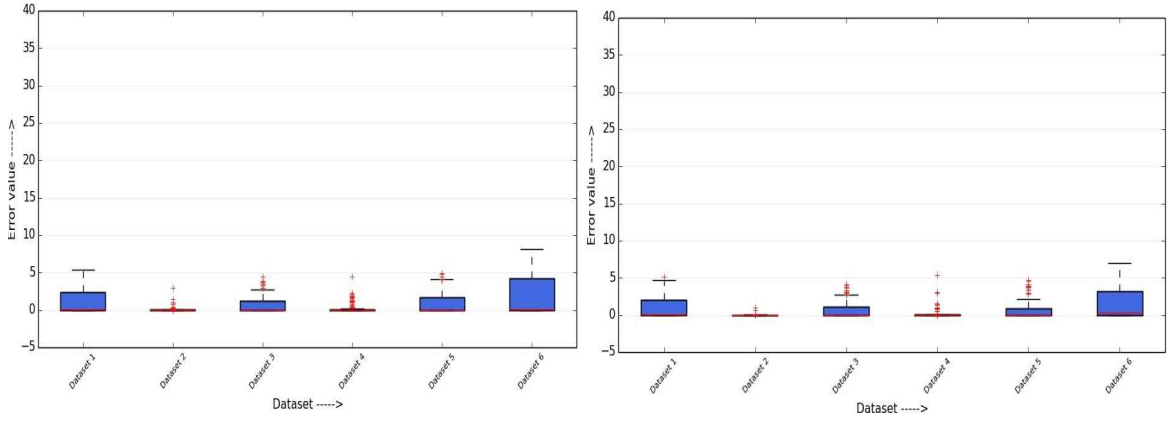
Analysis of Population Size: Population Size parameter of the proposed algorithm is another important parameter and the performance of the proposed algorithm varies for the different values of this parameter. The fluctuation in the performance due to the different values of the *Pop_Size* parameter can be easily observed in Table 8 and Fig. 12. In Table 8, the minimum error for different values of *Pop_Size* is shown. All the dataset used for the experiment with the proposed method produces the minimum value if the *Pop_Size* value is set to 60. Hence, for the rest of the experiment, the Pop_Size value is fixed to 60.

Table 8. Minimum, mean and maximum error of Pop_Size (PS)

Dataset		Minimum Error					
		PS= 20	**PS= 40**	**PS= 50**	**PS= 60**	**PS= 80**	**PS= 100**
DS- 1	**Minimum**	**5.14E-06**	6.94E-05	0.00023	0.000291	0.000178	7.66E-06
	Mean	0.357432	0.742566	0.62336	0.684708	0.474447	**0.584586**
	Maximum	1.045231	.9912341	0.985423	0.984712	1.104231	0.784575
DS- 2	**Minimum**	**3.26E-05**	6.41E-05	4.08E-05	0.000215	4.38E-05	6.38E-05
	Mean	0.910373	1.19726	1.112783	2.698928	1.09794	**2.524406**
	Maximum	0.940769	1.217231	1.784575	2.814531	1.584712	2.613425
DS- 3	**Minimum**	**2.18E-05**	5.4E-05	0.000427	3.28E-05	0.000153	1.18E-05
	Mean	1.628176	1.985265	2.489331	1.685876	1.899956	**1.748093**
	Maximum	1.912363	2.132456	2.9721650	2.134325	1.991234	1.814321
DS- 4	**Minimum**	**4.82E-06**	4.76E-05	8.24E-05	0.000361	2.49E-05	1.31E-05
	Mean	0.190216	0.339181	0.474563	0.735819	0.094042	**0.542211**
	Maximum	0.825432	0.943161	0.913425	.9934210	0.914531	0.900123
DS- 5	**Minimum**	**0.000255**	0.000149	0.000177	0.000142	0.000304	3.75E-05
	Mean	2.928514	3.73585	4.015667	0.479175	3.175658	**0.173401**
	Maximum	3.013245	3.919071	4.814321	1.104231	3.634325	0.985423
DS- 6	**Minimum**	**5.28E-06**	7.41E-05	0.000212	7.16E-05	1.56E-05	2.18E-05
	Mean	0.706386	0.746555	1.052162	0.443640	0.552767	**0.407218**
	Maximum	0.9045236	1.212365	1.400123	0.991234	.9934210	.9721650



(a) (b)



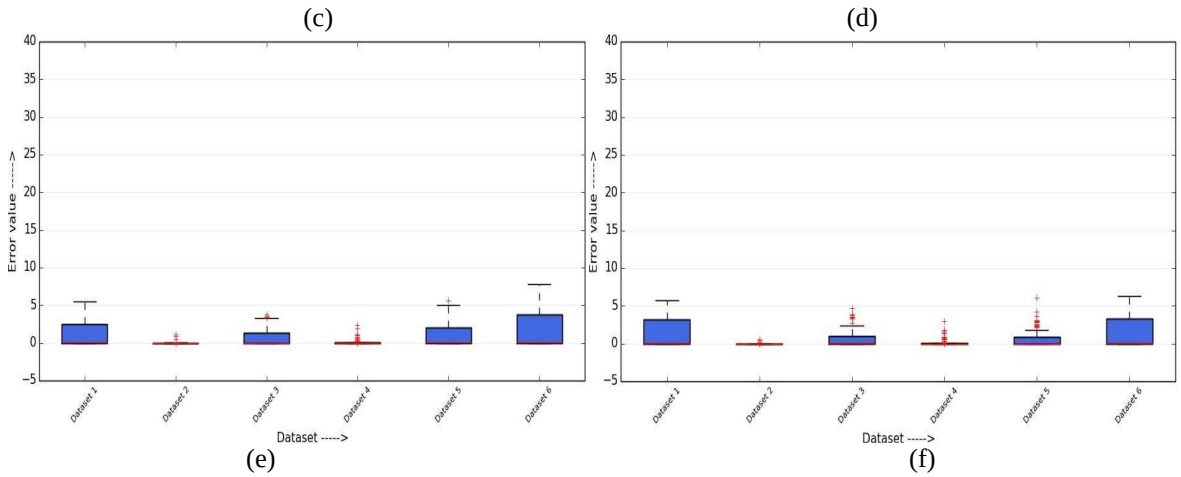


Fig. 12. Box Plot of Error for different value of Pop_Size (a) Pop_Size =20 (b) Pop_Size =40 (c) Pop_Size =50 (d) Pop_Size =60 (e) Pop_Size =80 (f) Pop_Size =100

Result analysis: The experiment was conducted with the fixed parameter values of alpha, beta, and Pop_Size which are decided by the analysis. The experiment was conducted to prove the convergence speed, stability and solution quality of the proposed EECS strategy. The convergence of the proposed EECS strategy over different synthetic datasets is presented in Table 9 and Fig. 13. The convergence is the minimum Euclidean distance between the target result and the optimized result produced by the proposed method. So, the main target of all the conducted experiment is to achieve the minimum error. It is observed from the experiment, which is presented in Table 9 that the error of the proposed algorithm is almost close to zero for all datasets. The performances of the proposed method over the extremely significant dataset are less than the predefined threshold value 0.0001 and the proposed method achieved the target level for all the datasets.

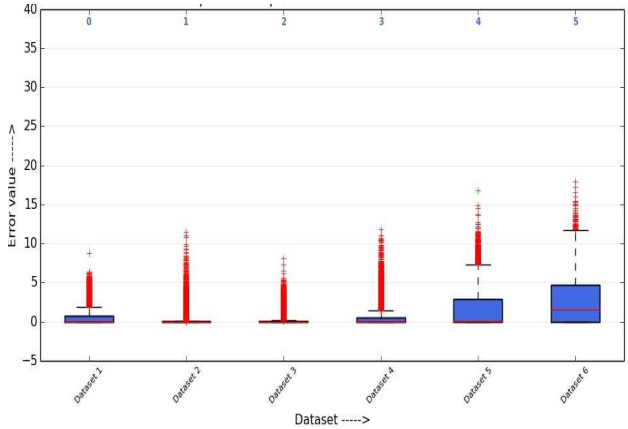


Fig. 13. Comparison of performance over different synthetic datasets

Table 9. Minimum and mean error over different synthetic datasets

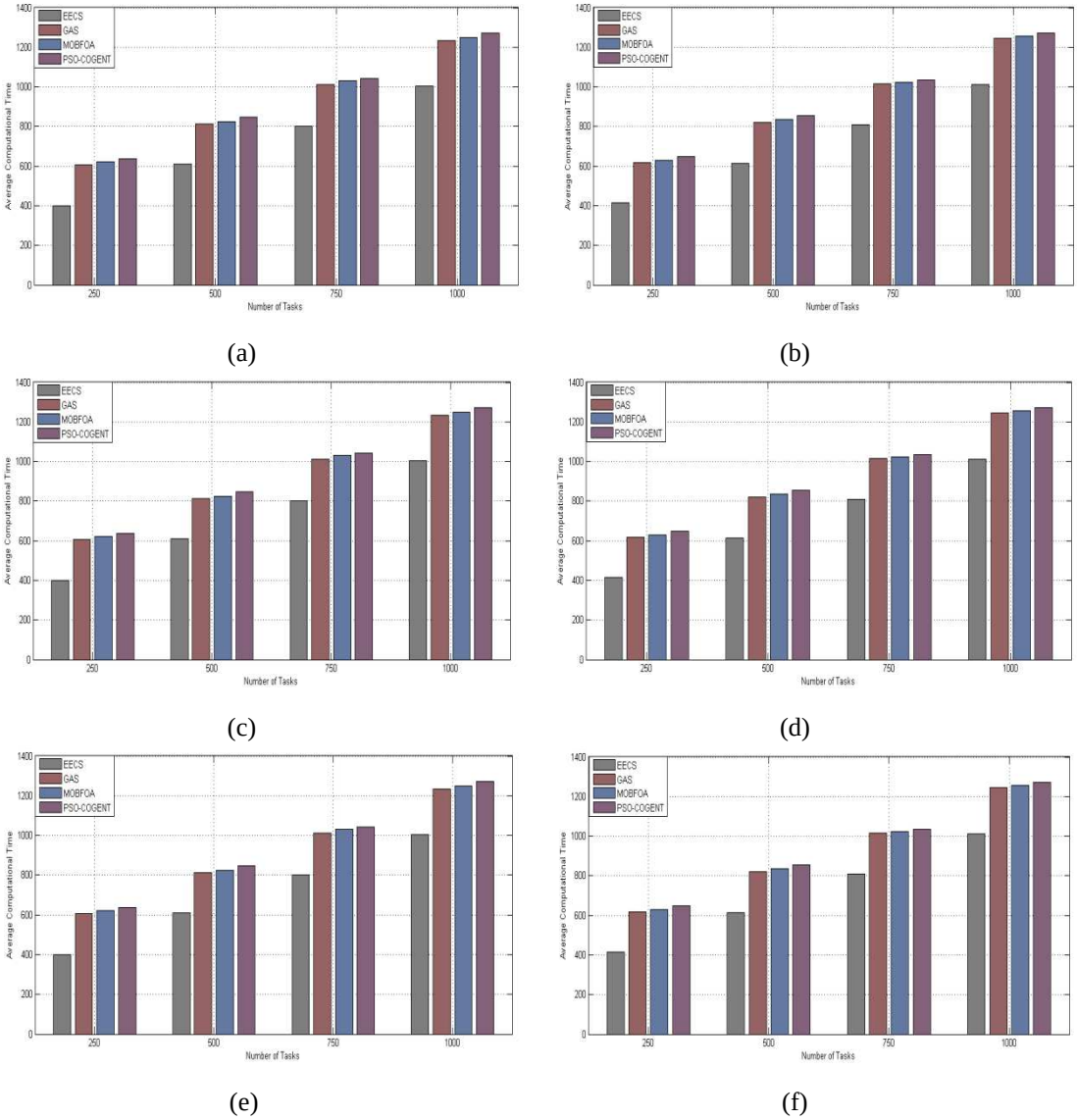
Datasets	Minimum Error	Mean Error	Maximum Error
DS 1	4.72758E-07	0.677862757	1.472345
DS 2	4.70339E-07	0.280634301	1.104523
DS 3	**1.36224E-08**	**0.396771942**	1.002345
DS 4	2.53E-07	0.690372	1.324561
DS 5	7.64E-07	1.597311	1.931023
DS 6	4.26E-06	2.589816	2.934127

7.4 Comparison Analysis

In this section, we evaluate the proposed EECS strategy via simulation runs over the six synthetic datasets and compare with the existing algorithms such as MOBFOA [20], PSO-COGENT [21] and GAS [32]. Here,

we consider various performance metrics to evaluate the proposed strategy such as computational time, energy consumption, CO_2 emission, Temperature emission, and resource utilization.

A) Computational Time: Computational time of a task is defined as the amount of time requires to complete its execution within an executing device. The computational time depend on the executing time of a task in the assigned resources, the time requires to transmit the task from the computational devices to the CDC and the time requires to deploy an executing device. As in Section 4, we already discussed that the deployment time of the VM instances is much higher than the containers. This may affect the computational time of the tasks. The proposed EECS strategy finds a suitable container for each task and assigns the container to the best-fit server. However, the existing MOBFOA and PSO-COGENT multi-objective scheduling strategies deploy the tasks to the suitable VM instances which may take more deployment time and consume the maximum amount of resources. However, the GAS deployed the tasks to the suitable containers based o single objective optimization strategy which may reduce the overall performance of the servers. The comparative analysis between the EECS algorithm and the existing state-of-art-algorithms in term of computational time over various numbers of tasks are shown in Fig. 14. The average computational time of EECS strategy is better than the GAS 21%, the MOBFOA algorithm by 26%, and the PSO-COGENT algorithm by 30%. Based on the experiments, the EECS strategy outperforms than other scheduling algorithms for different synthetic datasets.



(a) (b)

(c) (d)

(e) (f)

Fig. 14. Average computational Time of different state-of-arts-algorithms: (a) Dataset- 1; (b) Dataset- 2; (c) Dataset- 3; (d) Dataset- 4; (e) Dataset- 5; (f) Dataset- 6

The statistical analysis of the proposed EECS strategy and the existing multi-objective scheduling algorithms in a cloud environment is shown in Table 10. This may prove the efficiency of the EECS strategy over the existing ones in term of computational time. The metrics of minimum value, mean, maximum value and standard deviation (SD) are specified and computed for the algorithms based on the results of the computational time of various datasets. It was found that the MOBFOA, PSO-COGENT and GAS algorithms all performed significantly worse than the proposed EECS, especially when the algorithms need to schedule and execute a maximum number of tasks in CDC.

Table 10. Statistical analysis of computational time of EECS strategy and existing algorithms

Datasets		**Computational Time for IoT and Non-IoT based tasks**			
		EECS	**GAS**	**MOBFOA**	**PSO_COGENT**
DS- 1	**Minimum**	10	21	24	29
	Mean	35.12	47.17	49.34	55.45
	Maximum	65	91	94	104
	SD	5.495657	10.43240	11.34250	14.56231
DS- 2	**Minimum**	11	24	27	33
	Mean	37.54	49.23	51.92	58.12
	Maximum	71	92	97	110
	SD	4.215431	11.43245	12.43516	16.32187
DS- 3	**Minimum**	14	27	32	37
	Mean	40.32	53.29	59.17	61.75
	Maximum	71	96	101	118
	SD	5.912652	13.12543	15.32546	18.32187
DS- 4	**Minimum**	9	19	22	27
	Mean	36.12	45.86	54.73	59.56
	Maximum	65	85	92	102
	SD	4.126543	9.985641	11.43276	14.65234
DS- 5	**Minimum**	12	25	28	35
	Mean	40.54	42.34	50.23	56.10
	Maximum	69	87	90	107
	SD	5.103424	10.12875	12.12654	13.62342
DS- 6	**Minimum**	15	28	30	37
	Mean	42.44	56.12	53.56	58.07
	Maximum	72	98	94	114
	SD	6.123451	14.01235	16.12654	15.65213

B) Energy Consumption: Energy consumption of a task is defined as the amount of energy consumed by the resources of executing device which is assigned for that task (defined in Eq. (24)). The energy consumption of a task depends on the energy consumption of the transmitting channel and the energy consumption of the resources during processing a task in a computing server. Here, we considered that a suitable container is assigned for each task due to its minimum resource usage and limited deployment time for running a task. As in Section 4, we discussed that a container must consume a minimum amount of resources which may reduce the overall energy consumption for executing a task. The proposed EECS strategy finds a suitable container for each task and assigns the container to the best-fit server. However, the existing MOBFOA and PSO-COGENT multi-objective scheduling strategies deploy the tasks to the suitable VM instances which may consume the maximum amount of resources due to their own OS and consume maximum energy for running the assigned task. On the other hand, the GAS algorithm did not consider the energy consumption parameter for minimizing the overall energy consumption of the CDC. The comparative analysis between the EECS algorithm and the existing state-of-art-algorithms in term of energy consumption over various numbers of tasks are shown in Fig. 15. The average energy consumption of EECS strategy is better than the GAS 24%, the MOBFOA algorithm by 29%, and the PSO-COGENT algorithm by 33%.

Based on the experiments, the EECS strategy outperforms than other scheduling algorithms for different synthetic datasets.

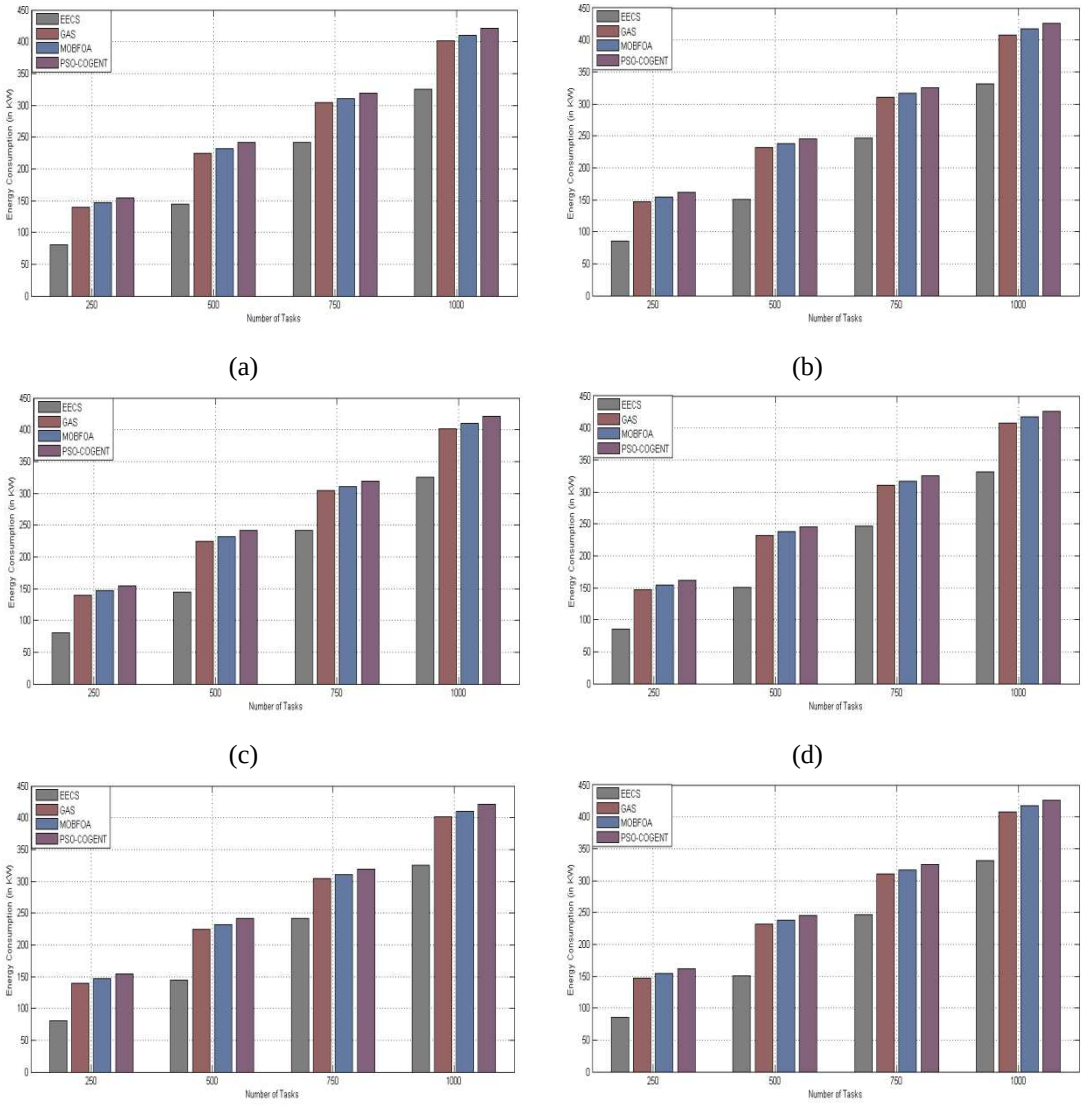


Fig. 15. Average energy consumption of different state-of-arts-algorithms: (a) Dataset- 1; (b) Dataset- 2; (c) Dataset- 3; (d) Dataset- 4; (e) Dataset- 5; (f) Dataset- 6

The statistical analysis of the proposed EECS strategy and the existing multi-objective scheduling algorithms in a cloud environment is shown in Table 11. This may prove the efficiency of the EECS strategy over the existing ones in term of energy consumption. The metrics of minimum value, mean, maximum value and standard deviation (SD) are specified and computed for the algorithms based on the results of the energy consumption of various datasets. It was found that the MOBFOA, PSO-COGENT and GAS algorithms all performed significantly worse than the proposed EECS, especially when the algorithms need to schedule and execute a maximum number of tasks in CDC.

Table 11. Statistical analysis of energy consumption of EECS strategy and existing algorithms

Datasets		**Energy Consumption for IoT and Non-IoT based tasks**			
		EECS	**GAS**	**MOBFOA**	**PSO_COGENT**
DS- 1	**Minimum**	7.12	15.34	18.32	21.53
	Mean	25.32	37.65	39.51	42.56
	Maximum	42.87	57.23	65.02	71.98

	SD	3.145657	3.84320	3.94520	4.14234
DS- 2	**Minimum**	8.37	17.45	20.11	22.76
	Mean	26.54	38.76	40.12	42.95
	Maximum	47.54	57.76	60.43	67.13
	SD	3.215431	4.33245	4.87516	5.13187
DS- 3	**Minimum**	7.86	13.54	16.78	15.65
	Mean	23.45	33.56	35.65	38.97
	Maximum	39.98	45.56	50.98	57.34
	SD	2.912652	3.72543	3.97546	4.13287
DS- 4	**Minimum**	9.11	11.43	15.32	19.11
	Mean	22.12	25.86	27.73	30.56
	Maximum	32.23	40.31	45.65	49.67
	SD	2.426543	2.985641	3.123276	3.75234
DS- 5	**Minimum**	12.12	19.13	21.32	24.34
	Mean	25.54	29.34	31.23	35.10
	Maximum	36.65	41.34	45.54	49.98
	SD	2.103424	2.82875	3.12654	3.72342
DS- 6	**Minimum**	11.12	16.43	19.67	21.34
	Mean	21.44	27.12	29.56	34.07
	Maximum	37.65	41.23	43.45	46.32
	SD	2.123451	2.81235	3.12654	3.85213

C) CO_2 Emission: CO_2 emission of a task is defined as the amount of CO_2 emitted by the resources of executing device which is assigned for that task (defined in Eq. (26)). The CO_2 emission of a task depends on the CO_2 emitted of the transmitting channel and the CO_2 emitted of the resources during processing a task in a computing server. Here, we considered that a suitable container is assigned for each task due to its minimum resource usage and limited deployment time for running a task. As in Section 4, we discussed that a container must consume a minimum amount of resources which may reduce the overall CO_2 emission for executing a task. The proposed EECS strategy finds a suitable container for each task and assigns the container to the best-fit server. However, the existing MOBFOA and PSO-COGENT multi-objective scheduling strategies deploy the tasks to the suitable VM instances which may consume the maximum amount of resources due to their own OS and emit maximum CO_2 for running the assigned task. On the other hand, the GAS algorithm did not consider the energy consumption parameter for minimizing the overall CO_2 emission of the CDC. The comparative analysis between the EECS algorithm and the existing state-of-art-algorithms in term of CO_2 emission over various numbers of tasks are shown in Fig. 16. The average CO_2 emission of EECS strategy is better than the GAS 25%, the MOBFOA algorithm by 30%, and the PSO-COGENT algorithm by 34%. Based on the experiments, the EECS strategy outperforms than other scheduling algorithms for different synthetic datasets.

The statistical analysis of the proposed EECS strategy and the existing multi-objective scheduling algorithms in a cloud environment is shown in Table 12. This may prove the efficiency of the EECS strategy over the existing ones in term of CO_2 emission. The metrics of minimum value, mean, maximum value and standard deviation (SD) are specified and computed for the algorithms based on the results of the CO_2 emission of various datasets. It was found that the MOBFOA, PSO-COGENT and GAS algorithms all performed significantly worse than the proposed EECS, especially when the algorithms need to schedule and execute a maximum number of tasks in CDC.

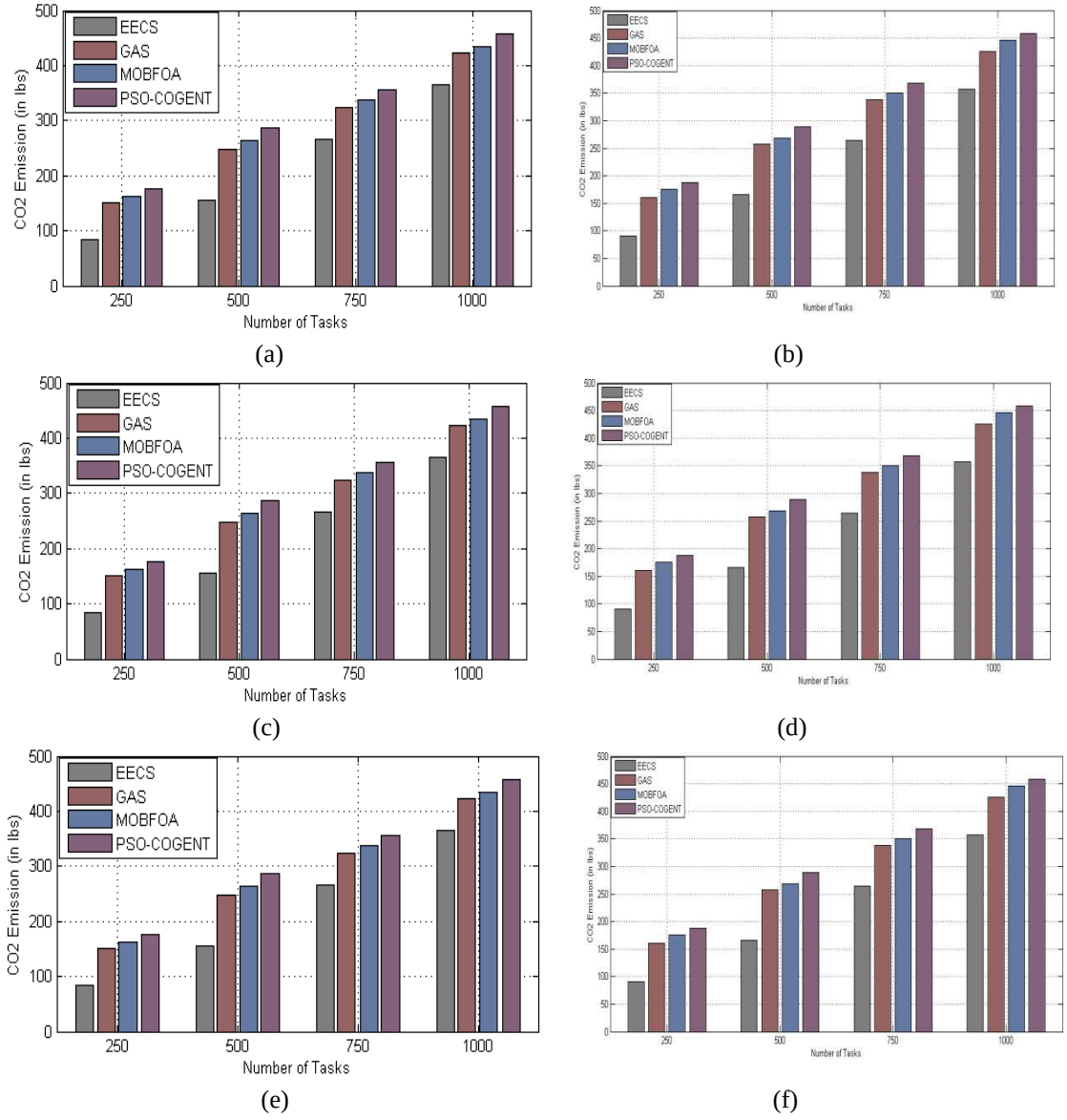


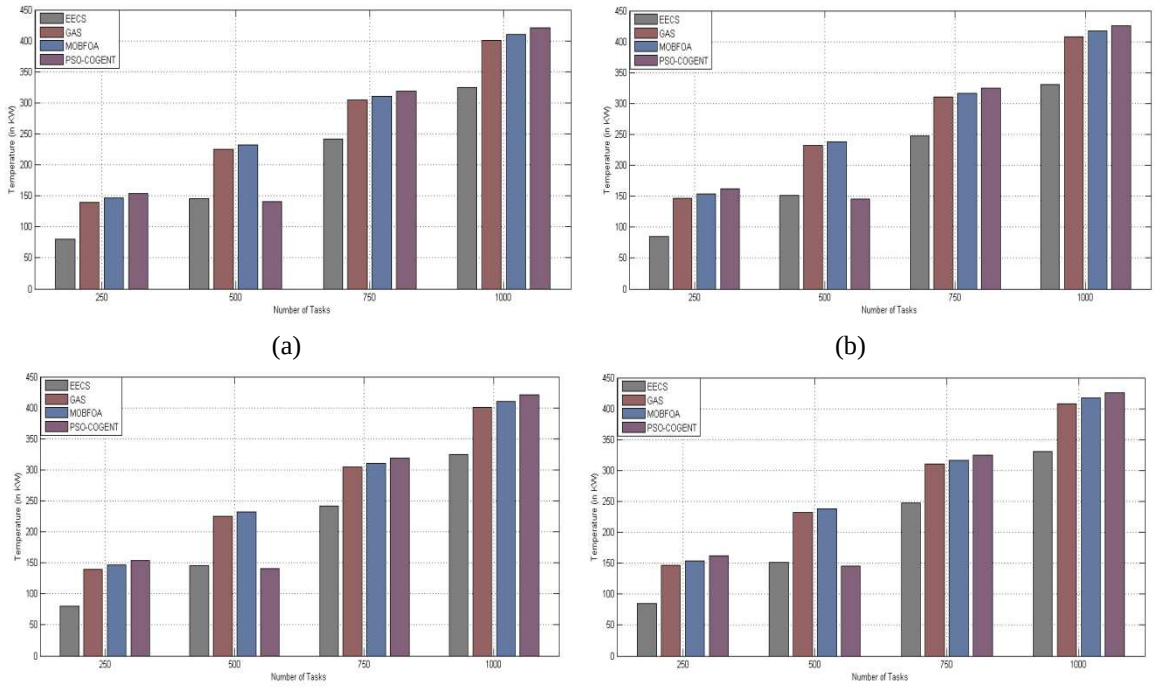
Fig. 16. Average CO_2 emission of different state-of-arts-algorithms: (a) Dataset- 1; (b) Dataset- 2; (c) Dataset- 3; (d) Dataset- 4; (e) Dataset- 5; (f) Dataset- 6

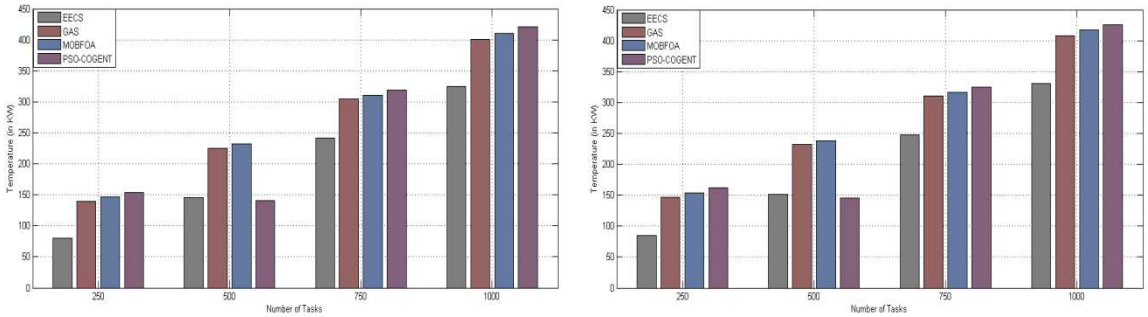
Table 12. Statistical analysis of CO_2 emission of EECS strategy and existing algorithms

Datasets		**Energy Consumption for IoT and Non-IoT based tasks**			
		EECS	**GAS**	**MOBFOA**	**PSO_COGENT**
DS- 1	**Minimum**	8.12	16.64	19.72	23.63
	Mean	25.32	38.65	40.53	43.58
	Maximum	47.87	59.23	68.02	73.98
	SD	3.245657	3.74320	3.94520	4.14234
DS- 2	**Minimum**	9.47	16.55	21.01	23.75
	Mean	27.54	38.98	41.17	41.75
	Maximum	47.84	59.76	61.73	66.53
	SD	3.205431	3.71245	3.97516	4.13187
DS- 3	**Minimum**	7.96	13.85	16.89	16.75
	Mean	23.35	33.46	35.75	38.86
	Maximum	39.98	45.56	50.98	57.34
	SD	2.912652	3.62543	3.87546	4.13287
DS- 4	**Minimum**	9.51	11.89	15.72	19.91
	Mean	22.89	26.76	28.96	31.76

	Maximum	33.43	41.51	46.75	50.07
	SD	2.456343	2.975641	3.343276	3.84234
DS- 5	**Minimum**	12.12	19.13	21.32	24.34
	Mean	25.54	29.34	31.23	35.10
	Maximum	36.65	41.34	45.54	49.98
	SD	2.103424	2.82875	3.12654	3.72342
DS- 6	**Minimum**	11.82	16.75	19.76	21.54
	Mean	22.65	28.82	30.12	35.18
	Maximum	38.75	42.45	44.56	48.45
	SD	2.223451	2.91235	3.62654	3.95213

D) Temperature Emission: Temperature emission of a task is defined as the amount of heat emitted by the resources of executing device which is assigned for that task (defined in Eq. (28)). The Temperature emission of a task depends on the amount of heat emitted of the transmitting channel and the heat emitted of the resources during processing a task in a computing server. Here, we considered that a suitable container is assigned for each task due to its minimum resource usage and limited deployment time for running a task. As in Section 4, we discussed that a container must consume a minimum amount of resources which may reduce the overall Temperature emission for executing a task. The proposed EECS strategy finds a suitable container for each task and assigns the container to the best-fit server. However, the existing MOBFOA and PSO-COGENT multi-objective scheduling strategies deploy the tasks to the suitable VM instances which may consume the maximum amount of resources due to their own OS and emit maximum heat for running the assigned task. On the other hand, the GAS algorithm did not consider the energy consumption parameter for minimizing the overall temperature emission of the CDC. The comparative analysis between the EECS algorithm and the existing state-of-art-algorithms in term of Temperature emission over various numbers of tasks are shown in Fig. 17. The average Temperature emission of EECS strategy is better than the GAS 24%, the MOBFOA algorithm by 29%, and the PSO-COGENT algorithm by 33%. Based on the experiments, the EECS strategy outperforms than other scheduling algorithms for different synthetic datasets.





(e) (f)

Fig. 17. Average CO_2 emission of different state-of-arts-algorithms: (a) Dataset- 1; (b) Dataset- 2; (c) Dataset- 3; (d) Dataset- 4; (e) Dataset- 5; (f) Dataset- 6

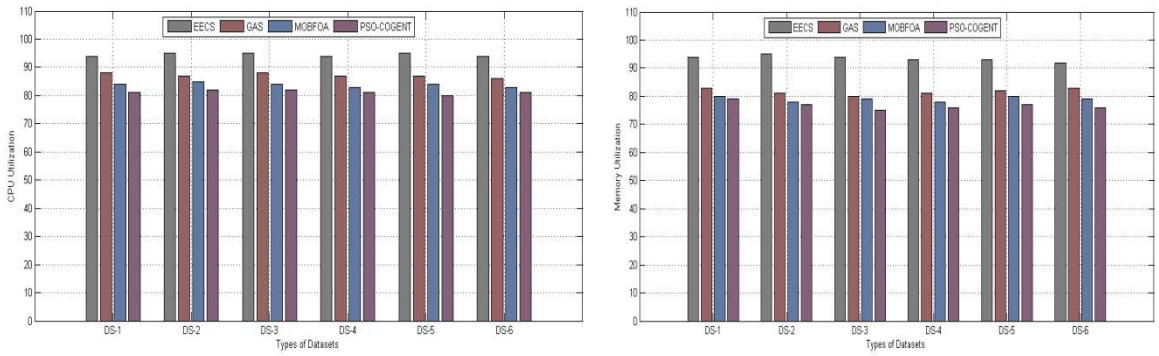
The statistical analysis of the proposed EECS strategy and the existing multi-objective scheduling algorithms in a cloud environment is shown in Table 13. This may prove the efficiency of the EECS strategy over the existing ones in term of Temperature emission. The metrics of minimum value, mean, maximum value and standard deviation (SD) are specified and computed for the algorithms based on the results of the Temperature emission of various datasets. It was found that the MOBFOA, PSO-COGENT and GAS algorithms all performed significantly worse than the proposed EECS, especially when the algorithms need to schedule and execute a maximum number of tasks in CDC.

Table 13. Statistical analysis of Temperature emission of EECS strategy and existing algorithms

Datasets		**Energy Consumption for IoT and Non-IoT based tasks**			
		EECS	**GAS**	**MOBFOA**	**PSO_COGENT**
DS- 1	**Minimum**	7.12	15.34	18.32	21.53
	Mean	25.32	37.65	39.51	42.56
	Maximum	42.87	57.23	65.02	71.98
	SD	3.145657	3.84320	3.94520	4.14234
DS- 2	**Minimum**	8.37	17.45	20.11	22.76
	Mean	26.54	38.76	40.12	42.95
	Maximum	47.54	57.76	60.43	67.13
	SD	3.215431	4.33245	4.87516	5.13187
DS- 3	**Minimum**	7.86	13.54	16.78	15.65
	Mean	23.45	33.56	35.65	38.97
	Maximum	39.98	45.56	50.98	57.34
	SD	2.912652	3.72543	3.97546	4.13287
DS- 4	**Minimum**	9.11	11.43	15.32	19.11
	Mean	22.12	25.86	27.73	30.56
	Maximum	32.23	40.31	45.65	49.67
	SD	2.426543	2.985641	3.123276	3.75234
DS- 5	**Minimum**	12.12	19.13	21.32	24.34
	Mean	25.54	29.34	31.23	35.10
	Maximum	36.65	41.34	45.54	49.98
	SD	2.103424	2.82875	3.12654	3.72342
DS- 6	**Minimum**	11.12	16.43	19.67	21.34
	Mean	21.44	27.12	29.56	34.07
	Maximum	37.65	41.23	43.45	46.32
	SD	2.123451	2.81235	3.12654	3.85213

E) Resource Utilization: Resource utilization of a server is defined as the number of executing devices executing the tasks and reuses the resources to schedule the tasks. As a container is a lightweight instance compared with the VM instances, the container must utilize the resources better than the VM instances. The containers are deployed over the host OS which must not require additional memory and CPU cycle to execute the processes of OS. However, the VM instances have their own OS and deployed over the

computing hardware and require additional memory to store the processes of OS image with extra CPU cycle. This may reduce the parallelism among the tasks in a server and minimizes the resource utilization in terms of CPU and memory. However, the containers may execute a number of tasks and increase the parallelism among the tasks. This strategy again improves the CPU and memory utilization of the servers. The proposed EECS strategy finds a suitable container instead of the VM instances for each task and assigns the container to the best-fit server. However, the existing multi-objective scheduling strategies deploy the tasks to the suitable VM instances which may consume the maximum amount of resources and minimize the CPU and memory utilization. On the other hand, GAS should assign the tasks to the suitable server without considering its resource availability which may reduce the overall performance. The comparative analysis between the EECS algorithm and the existing state-of-art-algorithms in term of CPU and memory utilization over various synthetic datasets are shown in Fig. 18.



(a) (b)

Fig. 18. Average resource utilization: (a) CPU Utilization; (b) Memory Utilization

The percentage of resource (CPU and memory) utilization of the proposed EECS strategy and the existing algorithms by the tasks on the servers is shown in tabular form (Table 14) for better understanding and discussion. The average CPU utilization of EECS strategy is better than the GAS 10%, the MOBFOA by 12%, and the PSO-COGENT by 15%. Similarly, the average memory utilization of EECS strategy is better than the GAS 19%, the MOBFOA algorithm by 23%, and the PSO-COGENT algorithm by 28%. Based on the experiments, the EECS strategy outperforms than other scheduling algorithms for different synthetic datasets.

Table 14. Resource utilization of EECS strategy and existing algorithms

CPU Utilization				
Datasets	**EECS**	**GAS**	**MOBFOA**	**PSO-COGENT**
DS- 1	94.75%	88.25%	84.35%	81.15%
DS- 2	95.66%	87.45%	85.15%	82.23%
DS- 3	95.88%	88.48%	84.27%	82.45%
DS- 4	94.32%	87.15%	83.24%	81.43%
DS- 5	95.93%	87.25%	85.22%	80.45%
DS- 6	94.99%	86.65%	83.18%	81.46%
Memory Utilization				
DS- 1	94.32%	83.45%	80.85%	79.22%
DS- 2	95.93%	81.78%	78.24%	77.15%
DS- 3	94.99%	80.45%	79.17%	75.15%
DS- 4	93.80%	81.87%	78.34%	76.35%
DS- 5	93.42%	82.85 %	80.65 %	77.26%
DS- 6	92.96%	83.92%	79.45%	76.85%

8. Conclusion

In this work, we have developed an energy-efficient container based strategy in a cloud environment, namely EECS to tackles MOPs. The main contribution of the algorithm is to find a suitable lightweight container for each task based on multiple-objectives such as energy consumption and computational time. Here, we have applied the APSO technique for finding a suitable container based on a weighted-sum approach. This may minimize the overall computation time and energy consumption of the CDC due to minimum deployment time and resource consumptions of the containers. Due to minimize energy consumption, the proposed EECS strategy also minimizes the overall CO_2 emission and temperature of the computing servers. The algorithm also finds a suitable computing server based on a rule-based strategy for the containers for better resource utilization of the multiple resources of the computing servers such as CPU and memory. We have conducted the simulation runs using six synthetic datasets. Through comparisons, we have established the superior performance of the proposed algorithm over the existing ones using various statistical and comparative analyses.

Our future research plan is to methodically scrutinize the trade-off between the qualities of container-based scheduling with different heuristic and meta-heuristic algorithms for different types of QoS parameters of various types of applications along with their penalties. Further, we will develop a dynamic container-based Cloud environment for IoT applications and assign the applications on an online basis to the best-fit containers for meeting various QoS constraints.

Reference

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility", *Future Generation Computer Systems*, vol. 25, pp. 599–616, 2009.

[2] T. Chatterjee, V. K. Ojha, M. Adhikari, S. Banerjee, U. Biswas and V. Snasel (2014), "Design and Implementation of a New Datacenter Broker policy to improve the QoS of a Cloud", *Proceedings of ICBIA 2014, Advances in Intelligent Systems and Computing*, vol. 303, pp- 281-290, 2014

[3] S. Banerjee, M. Adhikari, S. Kar, and U. Biswas, "Development and Analysis of a New Cloudlet Allocation Strategy for QoS Improvement in Cloud", *Arabian Journal for Science and Engineering*, Springer, vol. 40, pp- 1409-1425, 2014.

[4] Saurabh Kumar Garg, Adel Nadjaran Toosi, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud data center", *Journal of Network and Computer Applications*, vol. 45, pp- 108–120, 2014.

[5] Mainak Adhikari, and Tarachand Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud", *Future Generation Computer Systems*, vol. 81, pp. 156–165, 2018.

[6] S K Garg, A N Toosi, S K Gopalaiyengar, and R Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud data center", *Journal of Network and Computer Applications*, vol: 45, pp- 108–120, 2014.

[7] J. Stankovi, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1(1), pp. 3-9, 2014.

[8] R. DelValle, G. Rattihalli, A. Beltre, M. Govindaraju and M. J. Lewis, Exploring the Design Space for Optimizations with Apache Aurora and Mesos, *In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco*, CA, pp. 537-544, 2016.

[9] "Docker", *https://www.docker.com*, Accessed on November 2018.

[10] A. Zanella et al, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1(1), pp. 22-32, 2014.

[11] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. "Context-aware Dynamic Discovery and Configuration of Things in Smart Environment", *Springer International Publishing, Cham*, pp. 215–241, 2018.

[12] Charith Perera, Yongrui Qin, Julio C Estrella, Stephan Reiff-Margnaiec and Athanasios V Vasilakos, "Fog Computing for Sustainable Smart Cities: A Survey", *ACM Computing Surveys*, vol. 50(3), pp. 1-43, 2017.

[13] K Hightower, B Burns, and J Beda, “Kubernetes: Up and Running: Dive Into the Future of Infrastructure”, *O'Reilly Media, Inc*., 2017.

[14] V. Medel, O. Rana, J. Á. Bañares and U. Arronategui, “Modelling Performance & Resource Management in Kubernetes”, *In Proceedings of the 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), Shanghai*, pp. 257-262, 2016.

[15] Partha Pratim Ray, “A survey of IoT cloud platforms”, *Future Computing and Informatics Journal*, vol. 1, pp. 35-46, 2016.

[16] K. Miettinen, “Nonlinear Multiobjective Optimization”, *International Series in Operations Research & Management Science*, F. S. Hillier, Ed. Boston, MA: Springer US, vol. 12, 1998.

[17] K. Deb, “Multi-Objective Optimization Using Evolutionary Algorithms”, 1st ed., ser. *Wiley-Interscience series in systems and optimization*, Chichester, New York: John Wiley & Sons, 2001.

[18] Qingfu Zhang and Hui Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition,” *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 712–731, 2007.

[19] X S Yang, “Nature-Inspired Metaheuristic Algorithms”, *Luniver Press*, 2008.

[20] Xin-She Yang, Suash Deb and Simon Fong, “Accelerated Particle Swarm Optimization and Support Vector Machine for Business Optimization and Applications**”,** *International Conference on Networked Digital Technologies NDT 2011: Networked Digital Technologies*, pp 53-66, 2011.

[21] Mandeep Kaur, and Sanjay Kadam, “A novel multi-objective bacteria foraging optimization algorithm (MOBFOA) for multi-objective scheduling”, *Applied Soft Computing*, vol. 66, pp. 183–195, 2018.

[22] Mohit Kumar, and S.C. Sharma, “PSO-COGENT: Cost and energy efficient scheduling in cloud environment with deadline constraint”, *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 147–164, 2018.

[23] D. Bernstein, “Containers, and Cloud: LXC to Docker to Kubernetes”, *IEEE Cloud Computing*, vol. 1(3), pp. 81-84, 2014.

[24] C Kaewkasi, and K Chuenmuneewong, “Improvement of container scheduling for docker using ant colony optimization”, *Proceedings in 9th IEEE International Conference on Knowledge and Smart Technology (KST)*, pp. 254-259, 2017.

[25] L Yin, J Luo, and H Luo, “Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing”, *IEEE Transactions on Industrial Informatics*, vol. 14(10), pp. 4712-4721, 2018.

[26] W Li, A Kanso, and A Gherbi, “Leveraging Linux containers to achieve high availability for cloud services”, *Proceeding in IEEE International Conference on Cloud Engineering (IC2E)*, pp. 76-83, 2015.

[27] Alfonso Pérez, Germán Moltó, Miguel Caballer, and Amanda Calatrava, “Serverless computing for container-based architectures”, *Future Generation Computer Systems*, vol. 83, pp. 50-59, 2018.

[28] X. Tang, F. Zhang, X. Li, “Quantifying cloud elasticity with container-based auto-scaling”, *Future Generation Computer Systems*, DOI: https://doi.org/10.1016/j.future.2018.09.009, 2018.

[29] Ruiting Zhou, Zongpeng Li, and Chuan Wu, “Scheduling Frameworks for Cloud Container Services”, *IEEE/ACM Transactions on Networking*, vol. 26(1), pp. 436-450.

[30] D. Zhang, B.-H. Yan, Z. Feng, C. Zhang, and Y.-X. Wang, “Container Oriented Job Scheduling Using Linear Programming Model”, *Proceeding in 3rd International Conference on Information Management (ICIM), Chengdu*, 2017.

[31] Y. Tao, X. Wang, X. Xu, and Y. Chen, “Dynamic Resource Allocation Algorithm for Container-Based Service Computing”, *Proceeding in 13th International Symposium on Autonomous Decentralized System (ISADS), Bangkok*, 2017.

[32] C. Guerrero, I. Lera, and C. Juiz, “Genetic Algorithm for Multi-Objective Optimization of Container Allocation in Cloud Architecture”, *Journal of Grid Computing*, vol. 1(4), pp. 1-23, 2017.

[33] Y. Li, J. Zhang, W. Zhang and Q. Liu, “Cluster resource adjustment based on an improved artificial fish swarm algorithm in Mesos”, *Proceeding in 2016 IEEE 13th International Conference on Signal Processing (ICSP), Chengdu*, 2016.

[34] Y. Moa, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, “DRAPS: Dynamic and Resource-Aware Placement Scheme for Docker Containers in a Heterogeneous Cluster”, *Proceeding in 36th IEEE International Performance Computing and Communications Conference (IPCCC'2017)*, 2017.

[35] A. Havet, V. Schiavoni, P. Felber, M. Colmant, R. Rouvoy, and C. Fetzer, "GENPACK: A Generational Scheduler for Cloud Data Centers", *Proceeding in 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver*, 2017.

[36] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open Issues in Scheduling Microservices in the Cloud", Proceeding in Cloud *Computing*, 2016.

[37] X. Wan et al., "Application deployment using Microservice and Docker containers: Framework and optimization", *Journal of Network and Computer Applications*, vol. 119, pp. 97-109, 2018.

[38] Mainak Adhikari, Sudarshan Nandy, and Tarachand Amgoth, "Meta heuristic-based task deployment mechanism for load balancing in IaaS Cloud", *Journal of Network and Computer Applications*, vol. 128, pp. 64–77, 2019.

[39] X S Yang, "Engineering Optimization: An Introduction with Meta-heuristic Applications", *John Wiley & Sons, Chichester*, 2010.

[40] Xin-She Yang, Suash Deb, and Simon Fong, "Accelerated Particle Swarm Optimization and Support Vector Machine for Business Optimization and Applications", *NDT 2011, CCIS 136*, pp. 53–66, 2011.

[41] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communication Surveys and Tutorials*, vol. 17(4), pp. 2347-2376, 2015.

[42] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communication Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[43] L. Li, S. C. Li, S. S. Zhao, "QoS-aware scheduling of services-oriented Internet of Things," IEEE Transaction on Industrial Information, vol. 10(2), pp. 1497-1505, 2014.

[44] Salman Taherizadeha, Andrew C. Jones, Ian Taylor, Zhiming Zhao, and Vlado Stankovsk, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review", *Journal of Systems and Software*, vol. 136, pp. 19-38, 2018.

[45] L. Zadeh, "Optimality and non-scalar-valued performance criteria," IEEE Transaction on Automatic Control, vol. 8(1), pp. 59–60, 1963.