TÜBİTAK

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 1406 – 1427
© TÜBİTAK
doi:10.3906/elk-1810-47

Low-latency and energy-efficient scheduling in fog-based IoT applications

Dadmehr RAHBARI, Mohsen NICKRAY[*]
Department of Computer Engineering and Information Technology, University of Qom, Qom, Iran

Received: 05.10.2018 • **Accepted/Published Online:** 13.01.2019 • **Final Version:** 22.03.2019

Abstract: In today's world, the internet of things (IoT) is developing rapidly. Wireless sensor network (WSN) as an infrastructure of IoT has limitations in the processing power, storage, and delay for data transfer to cloud. The large volume of generated data and their transmission between WSNs and cloud are serious challenges. Fog computing (FC) as an extension of cloud to the edge of the network reduces latency and traffic; thus, it is very useful in IoT applications such as healthcare applications, wearables, intelligent transportation systems, and smart cities. Resource allocation and task scheduling are the NP-hard issues in FC. Each application includes several modules that require resources to run. Fog devices (FDs) have the ability to run resource management algorithms because of their proximity to sensors and cloud as well as the proper processing power. In this paper, we review the scheduling strategies and parameters as well as providing a greedy knapsack-based scheduling (GKS) algorithm for allocating resources appropriately to modules in fog network. Our proposed method was simulated in iFogsim as a standard simulator for FC. The results show that the energy consumption, execution cost, and sensor lifetime in GKS are better than those of the first-come-first-served (FCFS), concurrent, and delay-priority algorithms.

Key words: Fog computing, heuristic, greedy knapsack, resource scheduling

1. Introduction

In the modern world, many communication devices have built-in wireless sensors. Wireless sensor networks (WSNs) collect data in applications including healthcare, transportation, smart cities, smart energy grid, urgent computing. These networks as the infrastructure of IoT require real-time processing and decision-making. In a cloud-based system, data collected by WSNs should be transmitted to cloud over a period of time and processed there. In fact, processing in cloud has a lot of delays and bottlenecks for a large amount of collected data by end devices and sensors. Data transmission from end-sensor nodes to cloud by passing several midsensor nodes, routers, and gateways has high total network power consumption and long delays [1, 2]. In many sensitive cases such as medical care and transportation systems, high number of delays in IoT applications can lead to a patient's death or cause an accident. Edge computing has been the next generation of cloud computing, which performs computations in the vicinity of sensors and does not send the data to cloud [3]. In many applications, computing or storage is required in the absence of sufficient resources at the edge of the network, so fog computing (FC) is appropriate for this work.

FC, as a new paradigm, is located at the middle level of cloud and sensor nodes [4] so that data collection, processing, and storage can be done locally and data are sent to cloud only if needed. Processing in the fog device (FD), due to the proximity of sensors to the network edge, increases the speed and also reduces network traffic.

[*]Correspondence: M.Nickray@qom.ac.ir

FC with a large number of nodes has lower energy consumption than centralized cloud computing systems [5, 6]. Many challenges of the big data processing can be solved by FC [7]. Also, FC has many challenges as large-scale and distributed sensors and fog nodes (FNs), dynamically changing due to the on-off switching of IoT applications and the mobility of FNs, and quality of service (QoS) requirements [8].

An application in the fog network includes some modules. The modules in FDs can do a variety of computations, filter data to remove unnecessary items, and also assign a label to the data [9]. The FDs, as microdatacenters (MDCs), provide the resources for application modules. An optimal allocation policy of processing elements (PEs) to application modules will increase the productivity of resources inside the MDC [10]. In FC, scheduling is assigning resources to modules in a specified order. As there are many resources and modules in a fog architecture, scheduling is a serious challenge. The resource allocation is based on a number of QoS parameters [11]. Knapsack is an optimization problem in two ways, which includes the arbitrary and 0-1. Knapsack 0-1 is more important. It offers many practical applications such as capital budgeting, project selection, resource allocation, cutting stock, investment decision-making [12]. The greedy solution of the knapsack algorithm uses a simple sort with high execution speed. Thus, we used it to reduce the time delay in optimal allocation of PEs to modules in the fog network.

Our scheduling strategy was simulated by greedy knapsack algorithms in the fog network with two case studies. These case studies were based on the collection of data related to the tracking of objects and humans, data transmission to the edge of the network, processing, and sending them to cloud. Our key contributions in this paper are as follows. **1-** We formulatde the resource scheduling problem in FC using a greedy knapsack-based method. To create the objective function, the weighted sum method was used rather than the multiobjective algorithm. This method causes fast processing of the proposed algorithm. As greedy knapsack-based scheduling (GKS), after module placement in FDs then the best PEs of FDs are allocated to the requested modules. **2-** Two case studies as EEG (Electroencephalography) beam tractor and intelligent surveillance were analyzed by some parameters as the number of executed modules in FDs, transmission time, and resource management interval. **3-** The proposed method is compared with other algorithms based on different configurations by areas, departments, cameras, mobiles, and the number of users. The simulation results show that the quality of GKS as the energy consumption, delay, network usage, and total execution cost is better than first-come-first-served (FCFS), concurrent, and delay-priority algorithms [13] for two case studies.

The rest of this paper is organized as follows. Section 2 summarizes and categorizes the related works with scheduling methods in cloud and fog networks. In Section 3, we explain our proposed scheduling in detail as GKS. Two case studies on tracking applications and experiment results are presented in Section 4. The conclusion of this work is provided in Section 5.

2. Related work

Scheduling in distributed environments is generally divided into three categories: resource scheduling, workflow scheduling, and task scheduling [14]. Scheduling is defined as follows: finding an optimal solution for allocation of a set of resources $R = \{R_1, R_2, ..., R_m\}$ to a set of tasks $T = \{T_1, T_2, ..., T_n\}$ or workflow. Scheduling can be done by the deployment of a set of predefined constraints and objective functions [15]. The task is a small part of the work that must be performed within a specified time. One of the goals of task scheduling is to maximize the use of existing resources and minimize the waiting time of jobs [16]. In scheduling problems, the service user interest corresponds to optimal makespan, budget, deadline, security, and cost. Also, the service provider's objective is load balancing, resource utilization, and energy efficiency.

The optimization strategies include heuristics, metaheuristics [12, 17], and other methods. The resource models include virtual machine (VM) leasing model, single or multiple VM types, single or multiple providers, intermediate data sharing model, data transfer and storage cost awareness, static, dynamic, subscription, time unit VM pricing model, and VM provisioning delay [17, 18]. Scheduling strategies have different parameters and algorithms. We categorized these methods as follows:

Traditional methods are easy to use for scheduling. In the scheduling of multilevel deadline-constrained scientific workflows, each provider offers a few heterogeneous VMs and global storage service for data sharing [19]. One of the scheduling methods is the earliest finish time, in which cost and makespan objectives are optimized. The authors in [20] analyzed commercial infrastructure as a cloud service. They used the Pareto front as a decision support tool for the trade-off between appropriate solutions. They reduced the scheduling cost by half, but the makespan increased by 5%. In [21], a fault-tolerant task scheduling is considered by bidding strategy for spot VMs and latest time to on-demand instances. This method has low efficiency due to the cheapest spot VM. In [22], the authors presented a real-time workflow scheduler based on hardware failures. Their algorithm by three steps as backward shifting, resource scaling up and shrinking of resources optimized resource utilization than other baseline algorithms.

In [13], the authors studied the QoS parameters on three scheduling methods, i.e. concurrent, FCFS, and delay-priority in fog network. In the concurrent method, the arrival tasks are allocated to a cloudlet regardless of usage capacity. In the FCFS method, the tasks run in the order of entry, also if the processing power of the data center is less than the task request, then that program is placed in the scheduler queue. In the delay-priority method, tasks are scheduled based on lower delay. The researcher used iFogsim library with two games as video surveillance/object tracking application and EEG tractor beam game (EEGTBG). The results show that the concurrent method has a greater delay than FCFS and delay-priority methods. In EEGTBG, the number of modules in each device for the concurrent method is greater than those of the other two, also this parameter in FCFS method is greater than those of others in cloud. In the final comparison, the number of modules per device for delay priority method is approximately equal in all three methods.

Heuristic methods solve problems by a number of rules. The classic heuristics include the first fit, best fit, and worst fit. The cloud, fog [17], and edge providers [23] can run applications and tasks in large-scale by heuristic methods. In [24], the tasks scheduled by a heuristic algorithm, such that the objective function includes the makespan and the execution cost of the tasks. The results show the higher efficiency and lower mandatory cost than other methods. In [25], a coalitional game based on cluster formation is proposed for radio access networks in the fog. It optimized the throughput of the network with a distributed user scheduling algorithm. Another method for vehicular cloud scheduling is a dynamic algorithm [26] by parameters of queue's length and response time. In this work, the Markov single server system scheduled tasks and modeled them by stochastic Petri net and OpenStack. In [27], parallel real-time tasks in the heterogeneous network are scheduled by a heuristic method to optimize the energy consumption. In this research, frequency selection and thread allocation are done by nonlinear programming.

In [28], the researchers have proposed the knapsack algorithm for task scheduling for parallel video transferring in cloud by minimum complete time (MCT) objective. They executed the max-min algorithm on the powerful computers for task's mapping to a number of segments and scheduling segments by MCT algorithm. The results show that the max-min algorithm is better than MCT in the execution time and the number of segments. Researchers in [29] solved the offloading in mobile edge computing (MEC). They minimized the energy consumption under the time-sharing constraint. Their method decomposed computations onto the mobile devices and the MEC servers and make them autonomously collaborate to iteratively reach the optimum. In another research, task offloading of DAG-based applications is done in [30]. In this work, a cost-makespan aware scheduling (CMaS) algorithm was proposed. The results satisfy the user-defined deadline constraints and improve the QoS.

In [31] and [32], two heuristic algorithms proposed in fog-cloud environment for solving load balancing and VM live migration, respectively. They used min-conflicts optimizing and best fit decreasing (BFD) algorithms. The heuristic-based algorithm results show improvements compared to the traditional algorithms. In another work [33], to load balancing in FC, scheduling problem is moved from devices to server. They minimized deadline misses and total runtime for connected car systems in FC. Researchers in [34] provided a maximal energy-efficient task scheduling in FC with homogeneous nodes. Their selection strategy and the offloading time slots allocation decreased the total energy consumption.

Metaheuristic methods are used for resource allocation in parallel computing [35] and distributed computing [36]. The authors in [37] studied the task scheduling based on the metaheuristics approach in clouds. Each algorithm has its advantages and disadvantages. Scheduling solutions have issues like resource scaling, failure handling, security and storage-aware, dependent tasks, data transfer cost, and dynamic resource provisioning for the IoT. The hybrid solutions can be improved by changing of selecting initial population, diversity of the solution, or modifying the operators.

In the metaheuristics model, general algorithms are designed to solve optimization problems [11]. Particle swarm optimization (PSO) is used for resource provisioning and scheduling that considers features such as the elastic provisioning and heterogeneity of unlimited compute resources as well as VM performance variation. In [38], a hybrid method of PSO with cat swarm optimization is used for task scheduling and VM management in cloud. This method has reduced the average response time, also it has increased resource utilization by up to 12% in compared with other benchmark algorithms. Ant colony optimization (ACO) as a metaheuristic algorithm is used for mobile cloud computing [39] that requires specific resources. This method executes offloaded Tasks in FDs by delay, complete time, and energy consumption objectives. Another research [10] used ACO for scheduling and resource allocation in Cloud with trust and deadline objectives. As results, ACO minimized the throughput and the number of request failure and maximized the computation power. In [40], bee's life algorithm (BLA) was proposed for the job scheduling in the fog network. It is based on the optimized distribution of tasks in the FNs. The researchers using BLA find an optimal trade-off between runtime and memory allocation for mobile users. The results show that runtime and memory allocation values by BLA are lower than those of PSO algorithms and genetic algorithm (GA).

The authors in [41], found the optimal orders of running tasks based on deadline and minimum cost using knapsack with dynamic programming. The different scheduling problem solved in [42] by knapsack-based ACO to find the best solution as a mapping between multiple knapsacks and load scheduling. In [43], a resource scheduling in cloud solved by the combination of the knapsack and GA with the fitness function include utilization of CPU, network throughput, and input/output rate of the disk. They decreased the energy consumption of the physical machine and the number of migration compared to the standard methods. In [44], resources were allocated to tasks in FC using the NSGA-II method. This work used MATLAB for simulation. This scheduling method reduced the latency and improved the stability of task execution than random allocation method.

Machine learning is another method for solving the scheduling problem in IoT. Researchers in [45] combined reinforcement learning with the quality of experience (QoE) from the users to build prestored cost mapping table for optimal resource allocation. Their value function used energy and response time. In [46], energy-efficient scheduling is proposed in edge computing. To produce Q-value for each dynamic voltage and frequency scaling (DVFS) algorithms, a Q-learning algorithm was used. Also, the rectified linear units (ReLU) function was used rather than Sigmoid function to produce more objective Q-value.

Cloudsim [47] is the most widely used simulator in scheduling projects. iFogsim as an extension of Cloudsim is very suitable for IoT. Also, many of the metaheuristic scheduling algorithms are very timely; thus, we use a greedy knapsack algorithm for fast execution and best results.

3. The proposed approach

3.1. System model

FC architecture has a hierarchical arrangement of sensor nodes, edge devices, and cloud. Sensors are located in the lower geographic location at the bottom of the architecture and send the collected data to the up-level by gateways. The actuators at the bottom of the fog architecture control the environment or change it. The fog network process includes sensing and sending data, processing in the FDs, dividing an application into several modules, and allocating resources to them for execution. Applications are used to collect and store data in MDCs as well as future analyses and processes.

In Figure 1, the received data are processed in FDs or send to cloud. The FD and application properties are explained as follows. An FD is an MDC that analyzes, filters, and stores the received data from the sensors. The FD's properties include million instruction per second (MIPS), RAM, up bandwidth, down bandwidth, the level number in the topology, rate per MIPS, power in the busy state, and idle power. Each FD includes hosts as $\{Host_1, Host_2, ..., Host_n\}$. A host's properties include RAM, bandwidth, storage, and PEs. In a host, $FB_{Lower} \leq \sum_{i=1}^{N} HB_i \leq FB_{Upper}$, where FB is fog's bandwidth, HB is host's bandwidth before FB_{Lower} is the lower bandwidth and FB_{Upper} is the upper bandwidth of each FD. HB_i is the bandwidth of ith host. N is the number of hosts. The total bandwidth of all hosts in each FD is between FB_{Lower} and FB_{Upper}.

In FD, PEs of hosts are allocated to application modules and execute them. The most important feature of a PE is MIPS. These value sets at the start of simulation for all FDs. After allocation of a PE to an application module, the total allocated MIPS of all PEs is updated. Thus, we have $TAM = \sum_{i=1}^{N} \sum_{j=1}^{M} PEM_{ij}$, where TAM is the total allocated MIPS of a FD that is less than or equal to MIPS of that FD ($TAM \leq FD_{MIPS}$) and PEM is the PE's MIPS. N is the number of hosts in the FD, M is the number of PEs in a host, and PEM_{ij} is the MIPS of jth PE in ith host. The application module is a type of VM. The module's properties include MIPS, size, bandwidth, and the number of PEs. The number of modules in each FD is more than the number of PEs. In fact, $\sum_{i=1}^{C} Module_i > \sum_{j=1}^{K} FD_j$, where C is the total number of modules and K is the total number of FDs.

3.2. Scheduling schema

The scheduling is NP-hard problem and has high execution time. In this paper, we propose a fast method for allocation of PEs to the application modules. The default resource scheduler equally divides the host resources in FDs among all active application modules. In fact, by running simulation as Algorithm 1, N modules are

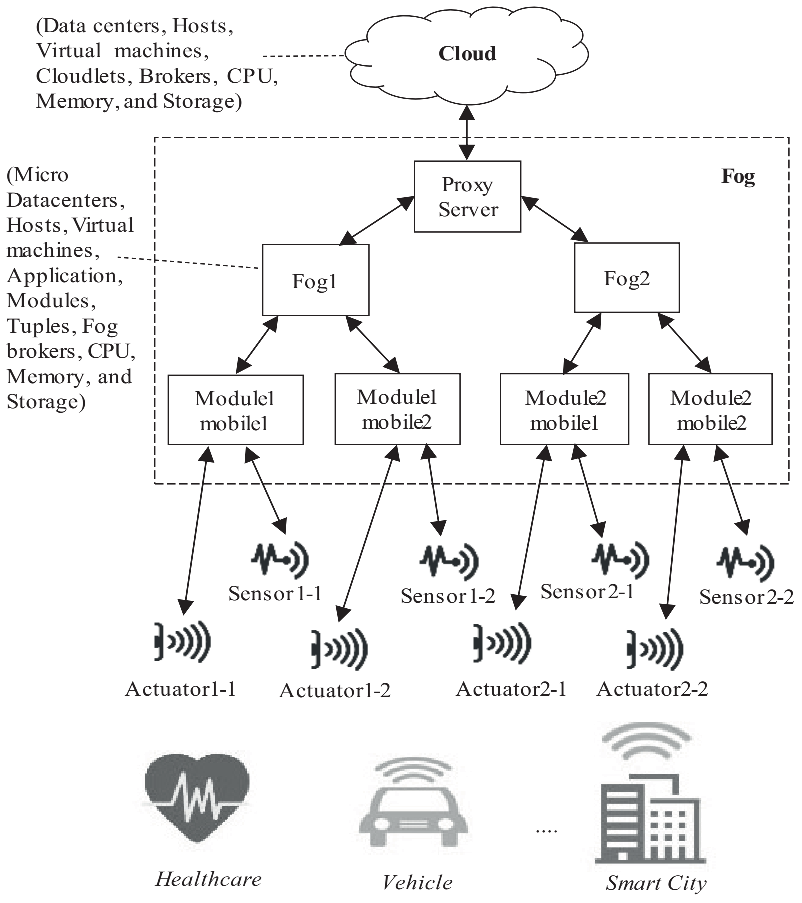


Figure 1. Fog computing architecture.

created as $\{m_1, m_2, m_3, ..., m_n\}$, and also there are K PEs in host's FD as $\{PE_1, PE_2, PE_3, ..., PE_K\}$. We consider the resource allocation of PEs to modules as a knapsack problem. As knapsack problem, each object has a weight as w and a profit as p. Thus, we have weights as $\{w_1, w_2, w_3, ..., w_n\}$ and profits as $\{p_1, p_2, p_3, ..., p_n\}$. In this paper, there are two objectives as total utilization of CPU and bandwidth, where w_{1i} is the MIPS of $module_i$ and w_{2i} is the bandwidth of $module_i$. An object is placed in knapsack if it has two conditions as $\sum w_{1i} * \delta_i \leq FDe_{MIPS}$ and $\sum w_{2i} * \delta_i \leq FD_{Bandwidth}$, where $\delta_i = 0{,}1$. A knapsack has the limited capacity. This capacity is considered by MIPS and bandwidth of FDs. The total MIPS of modules is less than the MIPS of FD and the total bandwidth of modules is less than FD's bandwidth. In greedy knapsack, the goal is to fill the knapsack with more objects by maximum profit and minimum weight $Maximize(\sum p_i * \delta_i)$.

3.3. GKS algorithm

Our proposed algorithm allocates PEs in the hosts to best modules. We updated the original application scheduling policy by changing the updateAllocatedMips method in FogDevice class in iFogsim [1] with the GKS algorithm. The profit of our problem is as follows. $Profit_i = k_1 * TUC_i + k_2 * BW_i..$, where TUC_i is the total utilization of CPU for allocated PEs to application modules and BW_i is the bandwidth of the application module. Since we have two objectives, the weighted sum method [48] was used. k_1 and k_2 are the weighted coefficients with equal value by 0.5. We complete this equation for knapsack problem as $\frac{p_i}{w_i} = \frac{k_1 * TUC_i}{w_{1i}} + \frac{k_2 * BW_i}{w_{2i}}$. As shown in Figure 2, the modules are entered into the system. In this figure, M_i is the ith module or object that will place in the knapsack. FD include many PEs. These PEs are allocated to modules based on the knapsack algorithm. According to iFogsim, if a module fails to fit into the array of modules in the knapsack then it will be transmitted to another FD. Thus, the wait time of the modules for PEs is optimized. The flowchart of GKS is shown in Figure 3.

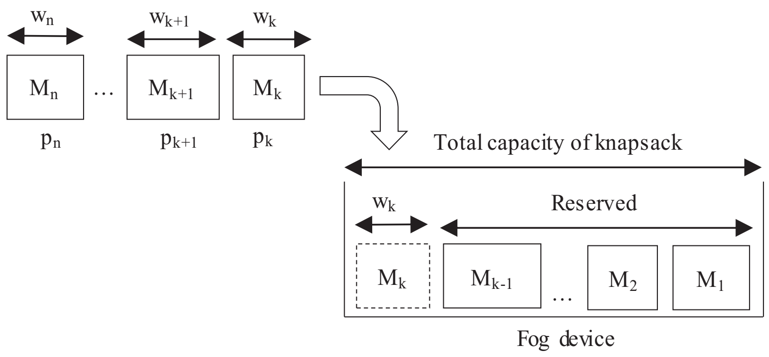


Figure 2. Scheduling by greedy knapsack.

The pseudo code of our proposed method is shown Algorithm 1. In this algorithm, all steps are defined to execute our method for each case study as the following. In line 1, the broker is created. An application is built in line 2. In this line, all modules and edges between them are added to the application. Also, all tuples (the communication units between modules) map between the modules. The workflow between all modules is added. In lines 3 to 7, FD is built for each area and camera. FD as an MDC is a class in iFogsim. Its attributes include memory, processor, storage, upper link and lower link bandwidths. The module mapping is initialized in line 8 and run by lines 9 to 11 as the modules are added to FDs. The created application with all their modules is submitted to network in line 12. The iFogsim engine starts in line 13. In line 14, the CPU utilization and bandwidth of all modules are calculated. The value of $\frac{p_i}{w_i}$ is calculated for all modules and is placed in a list. This list sorts in line 15. In line 16, the counter, the sum of MIPS, and the sum of BW for all modules are initialized by zero value. If enough resources are provided in FD, then the PEs are allocated to modules as lines 17 to 25. As line 19, if the input module is not running in current FD then PE is allocated to this module. The sum values of MIPS and BW are updated in lines 21 and 22. The module counter is increased by one in line 24. The cost, network usage, and energy are calculated in line 26. Finally, the iFogsim is stopped in line 27.

After changing updateAllocatedMips class, we ran two case studies and analyzed the iFogsim outputs. The GKS algorithm helped us to decrease the runtime using sorting of module's list with low time order. The time order of GKS is $O(n + n \log_2 n + m)$ as n is the number of the modules and m is the number of PEs.

4. Evaluation

4.1. Experiment setup

The simulation environment in this research was iFogsim library [1]. Our system for execution was a PC with properties as CPU (include Intel Core i5 2.67 Giga Hertz), RAM (3 Gigabyte), and OS as Microsoft Windows 10 32-bit. We simulated the GKS algorithm and compared the results with other scheduling methods as FCFS, concurrent, delay priority algorithms [13]. We ran the simulation by 6 states of areas/departments,

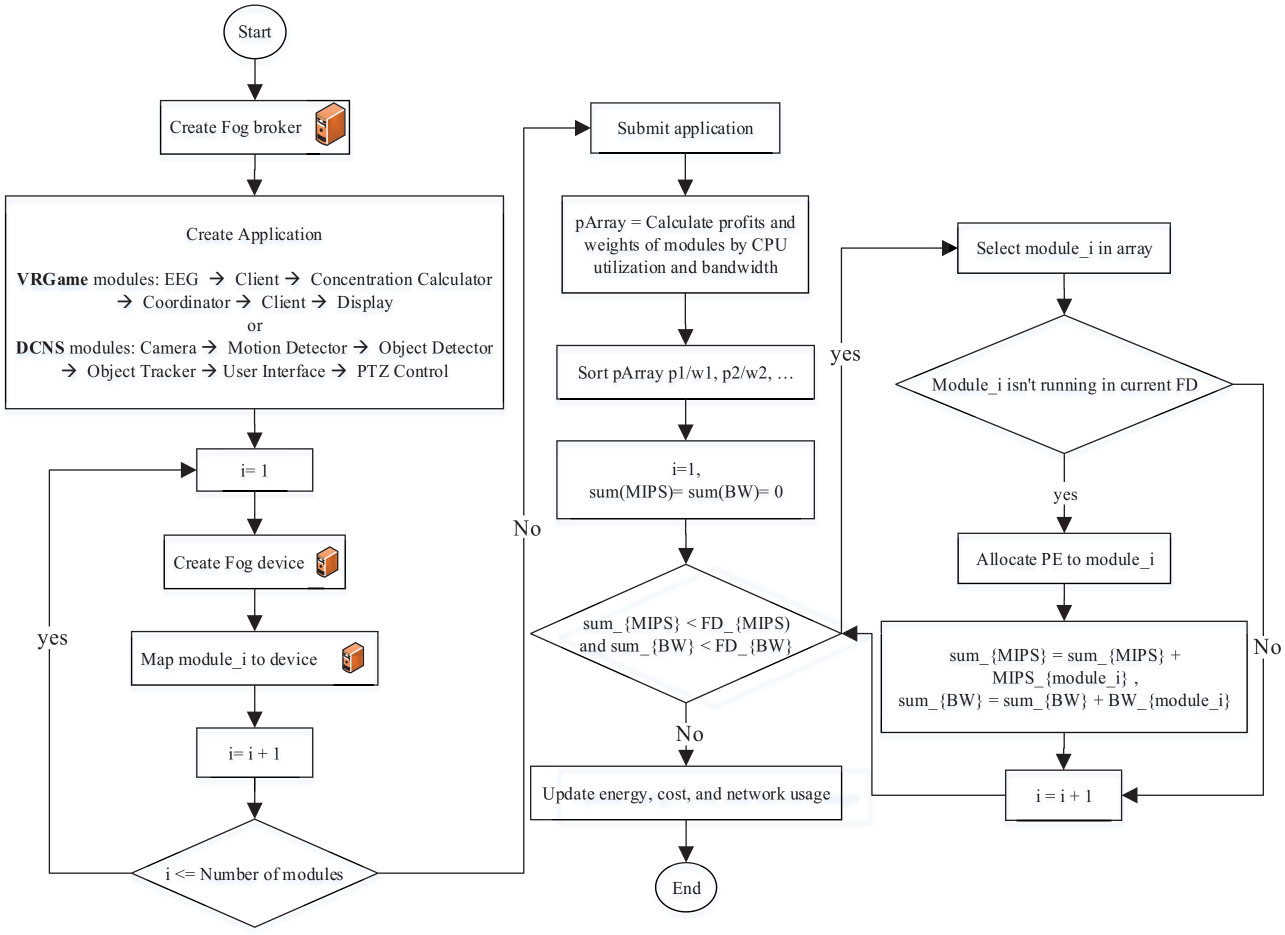

Start
Create Fog broker
Create Application
VRGame modules: EEG → Client → Concentration Calculator → Coordinator → Client → Display
or
DCNS modules: Camera → Motion Detector → Object Detector → Object Tracker → User Interface → PTZ Control
i= 1
Create Fog device
Map module_i to device
i= i + 1
i <= Number of modules
yes
No
Submit application
pArray = Calculate profits and weights of modules by CPU utilization and bandwidth
Sort pArray p1/w1, p2/w2, ...
i=1, sum(MIPS)= sum(BW)= 0
sum_{MIPS} < FD_{MIPS} and sum_{BW} < FD_{BW}
yes
No
Update energy, cost, and network usage
End
Select module_i in array
Module_i isn't running in current FD
yes
Allocate PE to module_i
sum_{MIPS} = sum_{MIPS} + MIPS_{module_i} , sum_{BW} = sum_{BW} + BW_{module_i}
i = i + 1
No

Figure 3. The flowchart of GKS.

cameras/mobiles, and FDs as $\{1 = (2,6,16), 2 = (2,7,18), 3 = (3,6,23), 4 = (3,7,26), 5 = (4,6,30), 6 = (4,7,34)\}$. The comparisons were based on the best results of algorithms with the same configuration for each one of case studies.

4.2. Case study 1: EEG beam tractor game

This game application is between two humans [1, 49] that involves augmented brain-computer interaction. The EEG headsets are connected to the smartphone. The EEG headset sends the real-time signals to the gateway and calculates the brain state of the user. Application model has three functions. The client module receives raw EEG signals and sends those to concentration calculator and display modules. The concentration calculator module receives and calculates the sensed EEG signal values. The coordinator module locates the game between players in distributed positions. The physical topology of case study 1 as VRGame (Virtual Reality Game) is shown in Figure 1. In this architecture, all cases in the dotted box are FDs, M_i is ith module, and m_i is ith mobile. The workflow loop of this case study is as $EEG \rightarrow Client \rightarrow ConcentrationCalculator \rightarrow Coordinator \rightarrow Client \rightarrow Display$. The application edge configuration of VRGame is shown in Table 1, where the periodicity (ms) in case it is periodic and tuple CPU or new length (byte) are the properties of a tuple.

Algorithm 1 GKS: Greedy Knapsack Scheduling

```
1: Create fog broker.
2: Create application (Modules, Edges, Tuples, Workflow).
3: for $i = 1$ to $Area_{max}$ do
4:     for $j = 1$ to $Camera_{max}$ do
5:         Create FD (Node name, MIPS, Ram, Storage, upper BW, lower BW, busy power, and idle power).
6:     end for
7: end for
8: Initialize module's mapping.
9: for $i = 1$ to $Fogdevice_{max}$ do
10:     Add module to $Fogdevice_i$.
11: end for
12: Submit applications.
13: Start iFogsim.
14: $pArray$ = Calculate profits and weights of modules by CPU utilization and bandwidth.
15: Sort the $pArray$ as $\{\frac{p_1}{w_1}, \frac{p_2}{w_2}, ..., \frac{p_n}{w_n}\}$.
16: $i = 1, sum_{MIPS} = sum_{BW} = 0$.
17: while $sum_{MIPS} \leq FD_{MIPS}$ and $sum_{BW} \leq FD_{BW}$ do
18:     Select $module_i$ in array.
19:     if Input application module isn't running in current FD then
20:         Allocate PE to $module_i$.
21:         $sum_{MIPS} = sum_{MIPS} + MIPS_{module_i}$.
22:         $sum_{BW} = sum_{BW} + BW_{module_i}$.
23:     end if
24:     $i = i + 1$.
25: end while
26: Update energy and cost using Eqs. (1) and (2) respectively.
27: Stop iFogsim.
```

Table 1. Application edge configuration of VRGame.

Source module	Destination module	Periodicity (mS)	Tuple CPU length (B)	Tuple new length (B)
EEG	Client	0	3000	500
Client	Concentration calculator	0	3500	500
Concentration calculator	Connector	100	1000	1000
Concentration calculator	Client	0	14	500
Connector	Client	100	28	1000
Client	Display	0	1000	500

4.3. Case study 2: intelligent surveillance through distributed camera networks

This case study was based on a distributed monitoring system of cameras in areas of healthcare, transportation, security, and manufacturing [1]. The requirements of this system were low-latency communication, handling massive data, and heavy long-term processing. In this case study, **object tracker** module tracks objects and calculates an optimal PTZ configuration. PTZ configuration adjusts the physical camera and serves as the actuator. The user interface module sends a fraction of the tracked object to the user's device. The physical topology of case study 2 as DCNSGame is exactly equal with VRGame but the cameras are used as FDs instead of mobiles as shown in Figure 1. The workflow loop of this case study is as $Camera \rightarrow MotionDetector \rightarrow ObjectDetector \rightarrow ObjectTracker\&UserInterface \rightarrow PTZControl$. The application edge configuration of DCNS is shown in Table 2.

Table 2. Application edge configuration of DCNS.

Source module	Destination module	Periodicity (ms)	Tuple CPU length (B)	Tuple new length (B)
Camera	Motion detector	0	1000	20,000
Motion detector	Object detector	0	2000	2000
Object detector	User interface	0	500	2000
Object detector	Object tracker	0	1000	100
Object tracker	PTZ control	100	28	100

4.4. Simulation configuration

As shown in Table 3, each FD as an MDC has many parameters including MIPS, RAM (KB), UpBW (Upper bandwidth by kilobyte per second), DownBW (Down bandwidth by kilobyte per second), level in the hierarchical topology, rate per MIPS, busy, and idle power (MW). The application module only has a bandwidth feature that is set in UpBW column. Table 4 shows the host's configuration. Host's properties include storage, bandwidth, architecture, operating system, VM model, time zone, cost, cost per memory, and cost per storage. In Table 5, module's properties are provided as RAM, MIPS, size of module, and bandwidth.

Table 3. FD configuration.

Name	MIPS	RAM	UpBw	DownBw	Level	Rate per MIPS	Busy power	Idle power
Cloud	44,800	40,000	100	10,000	0	0.01	1648	1.332
Proxy-server	2800	4000	10,000	10,000	1	0	107.339	83.4333
Area/department	2800	4000	10,000	10,000	1	0	107.339	83.4333
Camera/mobile	500	1000	10,000	10,000	3	0	87.53	82.44

Table 4. Host configuration.

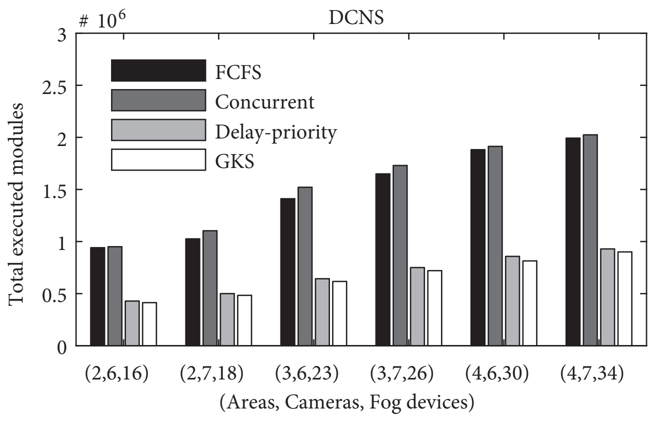
Storage	BW	Architecture	OS	VM model	Time zone	Cost	Cost per memory	Cost per storage
1,000,000 B	10,000 B/S	x86	Linux	Xen	10	3	0.05	0.01

Table 5. Application module configuration.

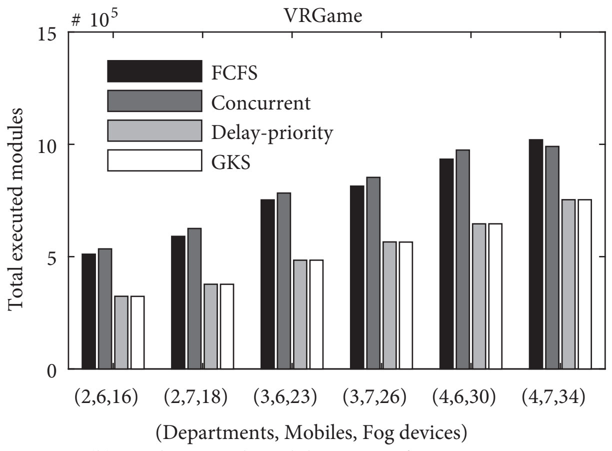
RAM	MIPS	Size	BW
10 B	1000	10,000 B	1000 B/S

4.5. Executed modules in FDs

The number of executed modules is a good criterion for load measuring in FDs. Figures 4a and 4b show this parameter for DCNS and VRGame case studies. The horizontal axis shows the number of areas/departments, cameras/mobiles, and FDs for each topology. The vertical axis is the total number of executed modules in all of the devices. As these figures, concurrent method has the maximum number of executed modules and GKS has the minimum number of it. These results are shown in DCNS and VRGame case studies.



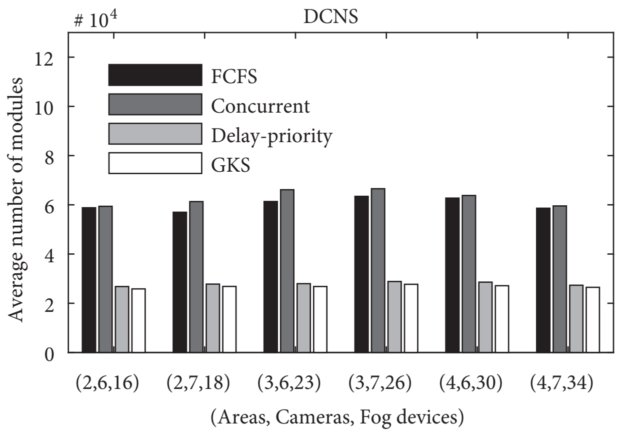
(a) Total executed modules in FDs for DCNS



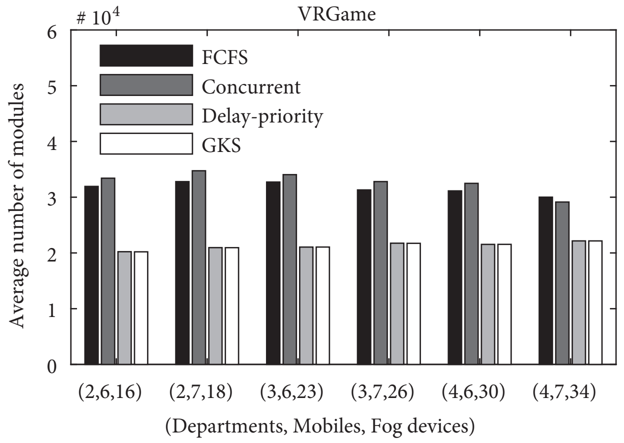
(b) Total executed modules in FDs for VRGame

Figure 4. Total executed modules in FDs.

The average number of modules in FDs is shown in Figures 5a and 5b. The GKS method has the best result compared to FCFS, concurrent, and delay-priority methods in two case studies. Based on the obtained values of the executed modules in FDs, the traditional methods are not better than heuristic and GKS methods. Table 6 show the simulation results of DCNS and VRGame. These metrics include the energy consumption, total execution cost, total and average number of executed modules in FDs.



(a) Average number of modules in FDs for DCNS



(b) Average number of modules in FDs for VRGame

Figure 5. Average number of modules in FDs.

Table 6. Simulation results. (Number of users = 1).

Application	Area	Camera	FD	FCFS	Concurrent	Delay-priority	GKS
				Total executed modules / 10^5			
DCNS	2	6	16	9.41	9.50	4.29	4.13
	2	7	18	10.3	11.0	5.0	4.83
	3	6	23	14.1	15.2	6.43	6.17
	3	7	26	16.5	17.3	7.50	7.20
	4	6	30	18.8	19.1	8.57	8.14
	4	7	34	19.9	20.2	9.29	9.0
VRGame	2	6	16	5.11	5.35	3.24	3.23
	2	7	18	5.90	6.25	3.77	3.77
	3	6	23	7.53	7.83	4.84	4.84
	3	7	26	8.14	8.53	5.65	5.65
	4	6	30	9.34	9.75	6.46	6.46
	4	7	34	10.2	9.91	7.53	7.53
				Average number of modules / 10^4			
DCNS	2	6	16	5.88	5.94	2.68	2.58
	2	7	18	5.70	6.13	2.78	2.68
	3	6	23	6.13	6.61	2.80	2.68
	3	7	26	6.34	6.65	2.89	2.77
	4	6	30	6.27	6.38	2.86	2.71
	4	7	34	5.86	5.95	2.73	2.65
VRGame	2	6	16	3.19	3.34	2.02	2.02
	2	7	18	3.28	3.47	2.10	2.09
	3	6	23	3.27	3.40	2.11	2.11
	3	7	26	3.13	3.28	2.17	2.17
	4	6	30	3.11	3.25	2.15	2.15
	4	7	34	3.0	2.91	2.22	2.22
				Energy consumption / 10^7			
DCNS	2	6	16	1.47	1.54	1.41	1.33
	2	7	18	1.49	1.54	1.45	1.33
	3	6	23	1.54	1.64	1.51	1.33
	3	7	26	1.57	1.66	1.49	1.33
	4	6	30	1.60	1.63	1.49	1.33
	4	7	34	1.60	1.67	1.54	1.33
VRGame	2	6	16	1.63	1.70	1.33	1.29
	2	7	18	1.63	1.72	1.33	1.28
	3	6	23	1.63	1.73	1.34	1.19
	3	7	26	1.63	1.79	1.34	1.23
	4	6	30	1.63	1.81	1.34	1.27
	4	7	34	1.64	1.83	1.34	1.24
				Total execution cost of DCNS / 10^6 (VRGame / 10^4)			
DCNS	2	6	16	1.77	1.97	1.58	0.94
	2	7	18	2.05	2.28	1.60	1.43
	3	6	23	2.31	2.89	2.31	1.39
	3	7	26	3.35	3.35	3.02	1.88
	4	6	30	3.06	3.82	3.06	2.45
	4	7	34	3.48	3.86	3.09	1.85
VRGame	2	6	16	4.17	4.59	2.10	1.90
	2	7	18	4.19	4.70	3.64	3.13
	3	6	23	4.22	5.08	6.0	4.09
	3	7	26	4.24	5.41	7.03	3.90
	4	6	30	4.28	6.01	7.40	3.71
	4	7	34	4.31	6.48	8.35	4.24

4.5.1. Resource management interval in DCNS

Figure 6 shows the results of our analysis of the GKS algorithm based on the interval of resource management in DCNS, where horizontal axis is the number of areas, mobiles, and fog devices. Three interval values equal to 200, 500, and 1000 were considered. As shown in Figure 6a, the interval 200 cause to minimize energy consumption. In Figure 6b, total execution cost of DCNS has maximum value by interval 1000. Also, Figures 6c and 6d show that interval 200 is suitable for minimum network usage and delay. Table 7 shows the simulation results of resource management interval in DCNS by metrics includimg energy consumption, total execution cost, and network usage.

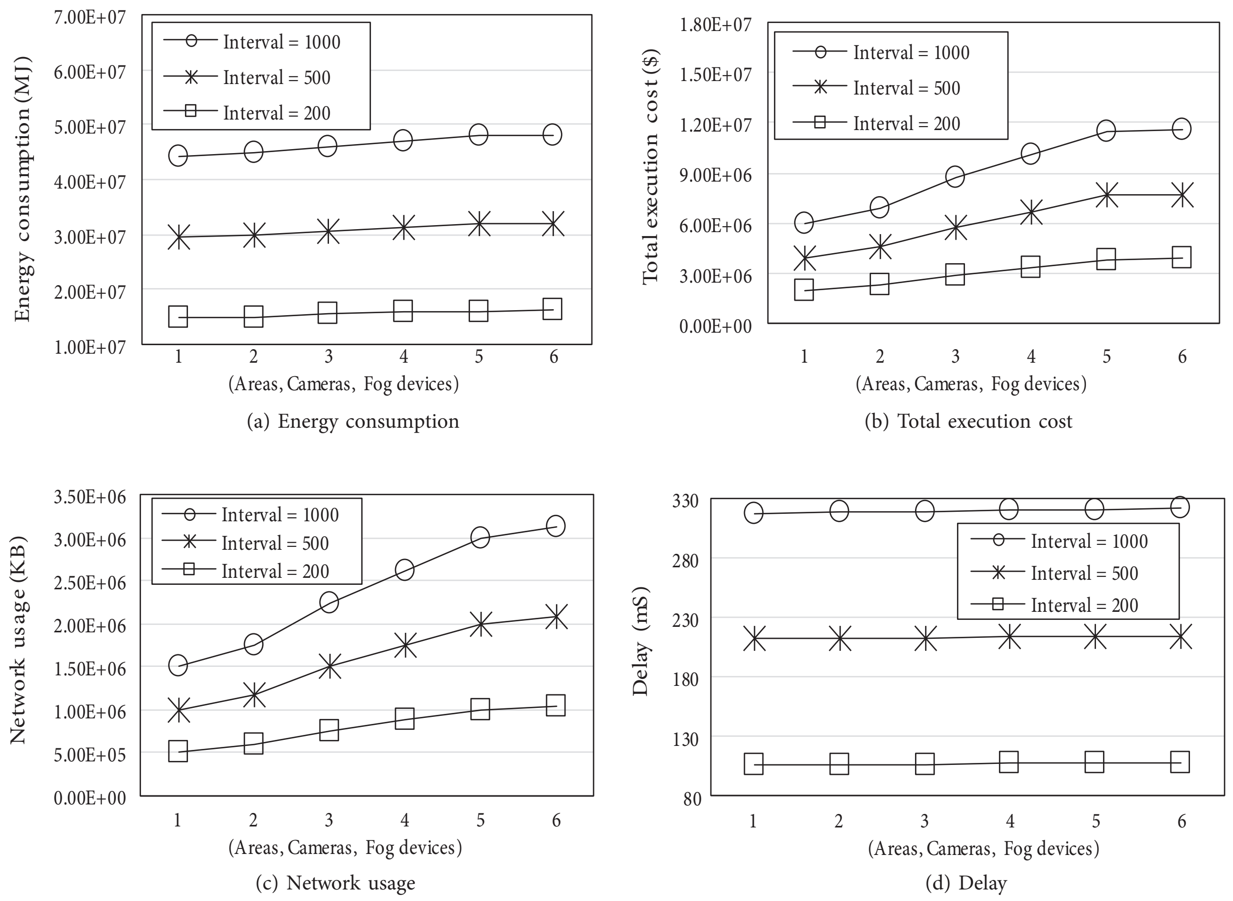


Figure 6. Analysis GKS algorithm based on resource management intervals of DCNS.

4.5.2. EEG transmission time in VRGame

The speed of EEG signal transmission to modules is a very important parameter in the quality of objectives. We consider EEG transmission time by 3, 4, and 5 values. Analysis of these results in Figure 7 indicates that the energy consumption (Figure 7a), total execution cost (Figure 7b), network usage (Figure 7c), and delay (Figure 7d) are decreased by EEG transmission time equal to 5. In fact, low value of this parameter can increase the quality of scheduling. Table 8 shows the simulation results of EEG transmission time in VRGame by metrics including the energy consumption, total execution cost, and network usage.

Table 7. Simulation results of resource management interval in DCNS. (Number of users = 1).

Interval	Area	Camera	FD	Energy consumption/10^7	Total execution cost/10^6	Network usage/10^5	Delay
200	2	6	16	1.47	1.98	4.99	106.04
	2	7	18	1.49	2.28	5.82	106.24
	3	6	23	1.54	2.89	7.48	106.64
	3	7	26	1.57	3.35	8.73	106.94
	4	6	30	1.60	3.82	9.97	106.24
	4	7	34	1.60	3.86	1.04	107.40
500	2	6	16	1.47	1.96	4.99	106.04
	2	7	18	1.49	2.28	5.82	106.24
	3	6	23	1.54	2.89	7.48	106.64
	3	7	26	1.57	3.35	8.73	106.94
	4	6	30	1.60	3.82	9.97	107.24
	4	7	34	1.60	3.87	1.04	107.50
1000	2	6	16	1.47	1.97	4.99	106.04
	2	7	18	1.49	2.28	5.82	106.24
	3	6	23	1.54	2.89	7.48	106.64
	3	7	26	1.57	3.25	8.73	106.94
	4	6	30	1.60	3.82	9.97	107.24
	4	7	34	1.60	3.86	1.04	107.91

4.5.3. Energy consumption

The energy consumption is calculated for the full topology of the network with Eq. (1).

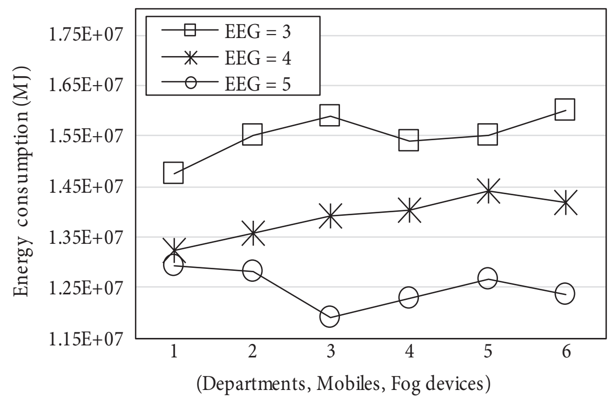
$$Energy = CEC + (NT - LUUT) * HP. \tag{1}$$

The energy consumption of FD is calculated by the power of all hosts in a certain time frame of execution, where CEC is the current energy consumption, NT is the current time, $LUUT$ is the last utilization update time, and HP is the host power in LU (Eq. (2)).

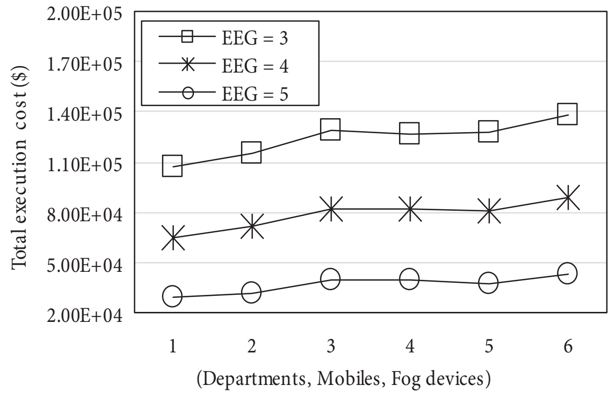
The simulation results show that our proposed method obtains less energy consumption than other scheduling policies. The average value of the energy consumption by $1.25 * 10^7$ belongs to GKS in VRGame and the concurrent method has maximum value in average state by 1.76. Also, this parameter value is equal to $1.33 * 10^7$ by GKS in DCNS. GKS algorithm reduces the energy consumption in DCNS by 13.8%, 17.5%, and 9.97% compared to FCFS, concurrent, and delay-priority methods respectively. Also, GKS algorithm reduces the energy consumption in VRGame by 23.4%, 29.1%, and 6.46% compared to FCFS, concurrent, and delay-priority methods respectively. Considering the mentioned values and Figures 8a and 8b, for the two case studies, VRGame has more optimization than DCNS.

4.5.4. Total execution cost

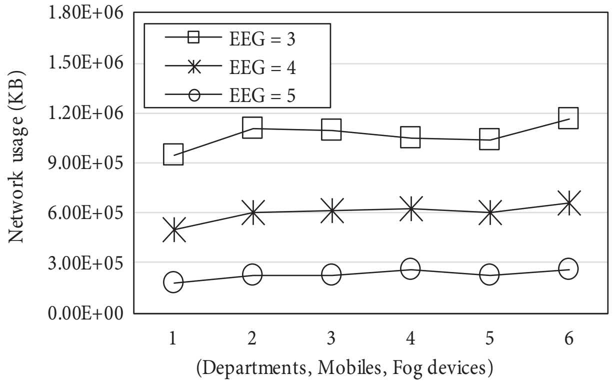
$$Cost = \sum_{i=1}^{F} [TC + (CC - LUUT) * RPM * LU * TM]. \tag{2}$$



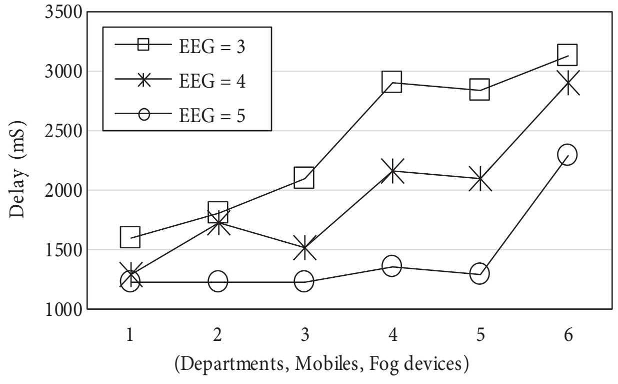
(a) Energy consumption



(b) Total execution cost

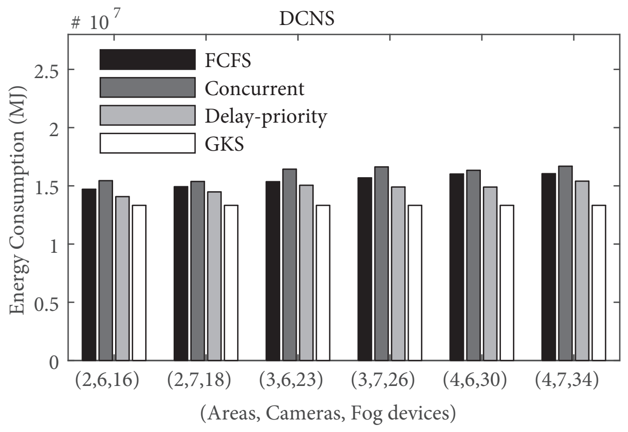


(c) Network usage

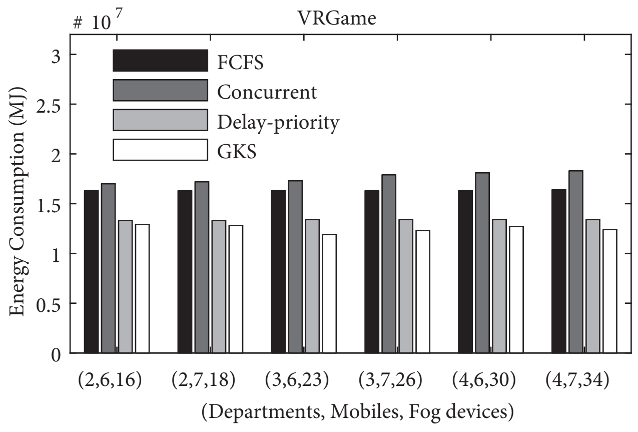


(d) Delay

Figure 7. Analysis GKS algorithm based on EEG transmission times of VRGame.



(a) Energy consumption of DCNS



(b) Energy consumption of VRGame

Figure 8. Energy consumption of FCFS, concurrent, delay-priority, and GKS

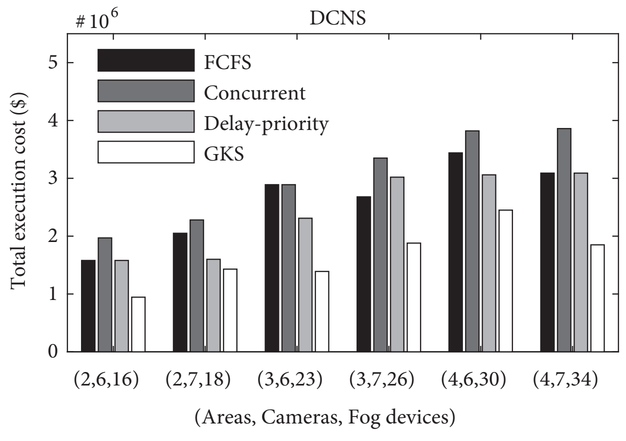
To calculate the execution cost, the total MIPS of hosts is calculated by time frame. The time frame is different between the current time of simulation and the last utilization time. In Eq. (2), F is the number of FDs, TC is the execution cost, CC is the CloudSim clock or current time of simulation, $LUUT$ is the last utilization

Table 8. Simulation results of EEG transmission time in VRGame. (Number of users = 1).

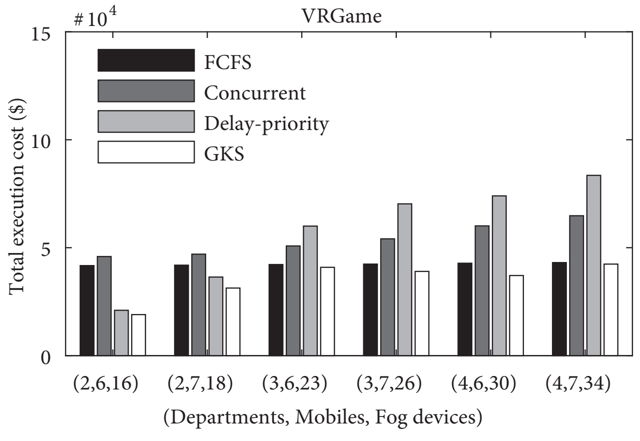
Interval	Area	Camera	FD	Energy consumption/10^7	Total execution cost/10^6	Network usage/10^5	Delay
3	2	6	16	1.48	4.32	4.43	1.61
	2	7	18	1.55	4.37	5.11	1.80
	3	6	23	1.59	4.63	4.76	2.10
	3	7	26	1.54	4.47	4.26	2.91
	4	6	30	1.55	4.77	4.36	2.85
	4	7	34	1.60	4.91	5.03	3.13
4	2	6	16	1.32	3.54	3.18	1.30
	2	7	18	1.36	4.0	3.76	1.72
	3	6	23	1.39	4.26	3.98	1.51
	3	7	26	1.40	4.28	3.70	2.17
	4	6	30	1.44	4.35	3.70	2.10
	4	7	34	1.42	4.63	3.93	2.91
5	2	6	16	1.29	2.90	1.83	1.23
	2	7	18	1.28	3.13	2.22	1.23
	3	6	23	1.19	3.99	2.19	1.23
	3	7	26	1.23	3.90	2.57	1.35
	4	6	30	1.27	3.71	2.26	1.29
	4	7	34	1.24	4.24	2.63	2.30

update time, RPM is the rate per MIPS that is different for each intermodule edges, and TM is the total MIPS of the host. Also, LU is the last utilization (LU) that is calculated as $LU = Min(1, TMA/TM)$, where TMA is the total allocated MIPS of the host.

As results, the best average value of this parameter for DCNS is obtained by GKS equal to $1.66 * 10^6$. The comparison of GKS, FCFS, concurrent, and delay-priority methods is also presented in Figures 9a and 9b.



(a) Total execution cost of DCNS



(b) Total execution cost of VRGame

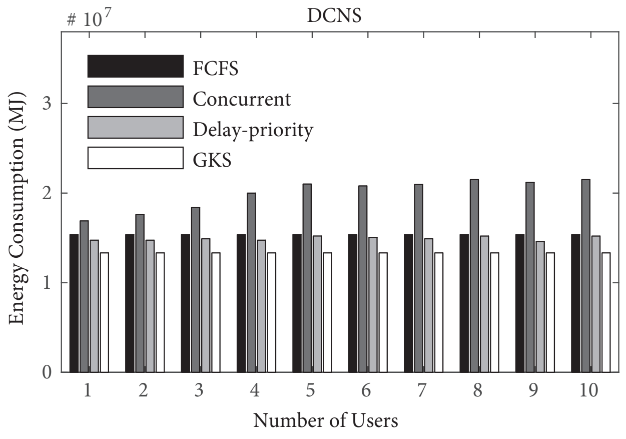
Figure 9. Comparison of total execution cost by FCFS, concurrent, delay-priority, and GKS

The GKS optimizes the total execution cost by 40.5%, 45.3%, and 32.1 compared to FCFS, concurrent, and delay-priority in DCNS. Also, this parameter is optimized by GKS than FCFS, concurrent, and delay-priority by 17.5%, 35.1%, and 39.3 in VRGame.

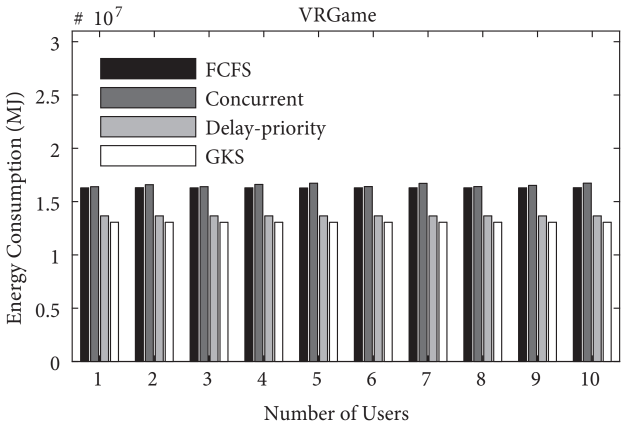
4.6. Analysis based on the number of users

4.6.1. Energy consumption based on the number of users

Another parameter for comparison of methods in this paper is the number of users. We consider 10 different numbers for this parameter and ran simulation based on it. The results indicate that GKS algorithm by $1.33 * 10^7$ in DCNS and $1.33 * 10^7$ in VRGame is better than FCFS, concurrent, and delay-priority methods. Figures 10a and 10b show the comparison of scheduling methods. GKS algorithm has better results than others. In DCNS, GKS reduces the energy consumption then FCFS, concurrent, and delay-priority by 13.2%, 33.3%, and 10.7% respectively. Also in VRGame, GKS reduces the energy consumption compared to FCFS, concurrent, and delay-priority by 19.8%, 21.1%, and 4.39%, respectively.



(a) Energy consumption of DCNS



(b) Energy consumption of VRGame

Figure 10. Comparison of energy consumption by FCFS, concurrent, delay-priority, and GKS based on the number of users.

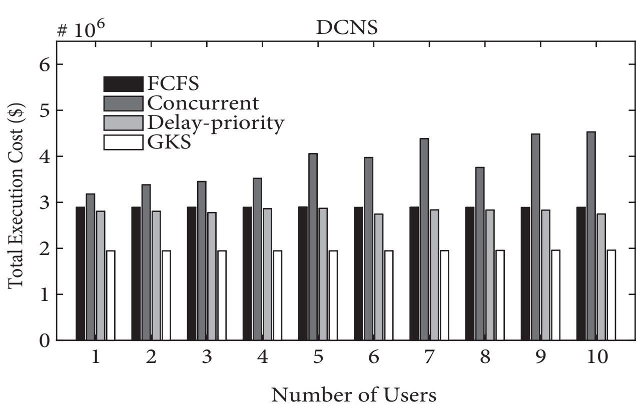
4.6.2. Total execution cost based on the number of users

We present total execution cost based on the number of users for FCFS, concurrent, delay-priority, and GKS methods.The results indicate that the GKS has the best values in DCNS by $.95 * 10^6$. Figures 11a and 11b show the comparison of the mentioned methods for DCNS and VRGame. In DCNS, the GKS is better than FCFS by 32.6%, concurrent by 49.6%, and delay-priority by 30.6%. Also in VRGame, the GKS is better than FCFS by 22.6%, concurrent by 30.3%, and delay-priority by 18.4%.

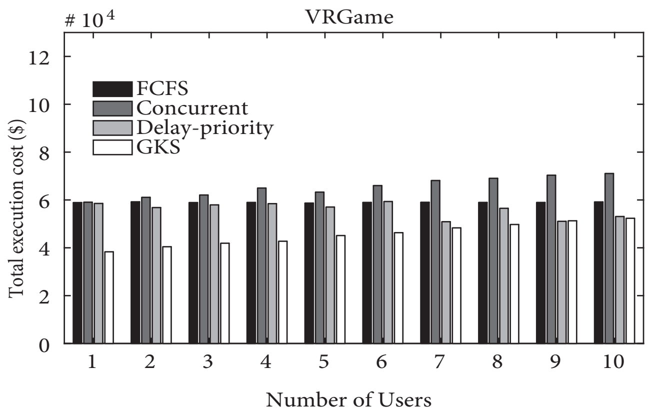
4.6.3. Delay of application loop based on the number of users

The application loop delay is calculated by Cloudsim clock and the tuple's end time.

$$TupleEndTime = \begin{cases} CC - T1 & \text{if } A = False, \\ \frac{T1*C1+(CC-T1)}{C1+1} & \text{if } A = True \end{cases} \tag{3}$$



(a) Total execution cost of DCNS



(b) Total execution cost of VRGame

Figure 11. Comparison of total execution cost by FCFS, concurrent, delay-priority, and GKS based on the number of users.

The tuple's end time as shown in Eq. (3) is when the tuple is completed. Here, $T1$ is the tuple start time, $T2$ is the tuple type to average CPU time, CC is the Cloudsim clock, $(CC - T1)$ is the execution time, and $C1$ is the number of executed tuple type. A is the condition, so if $T2$ is calculated then A is true else A is false.

$$Delay = CC - ET \tag{4}$$

The application loop delay is calculated by Eq. (4), where CC is the Cloudsim clock and ET is the emitting time of a tuple. ET is calculated by sending time of a module to another module.

$$TupleReceiptTime = \frac{T1 * C2 + Delay}{C1 + 1} \tag{5}$$

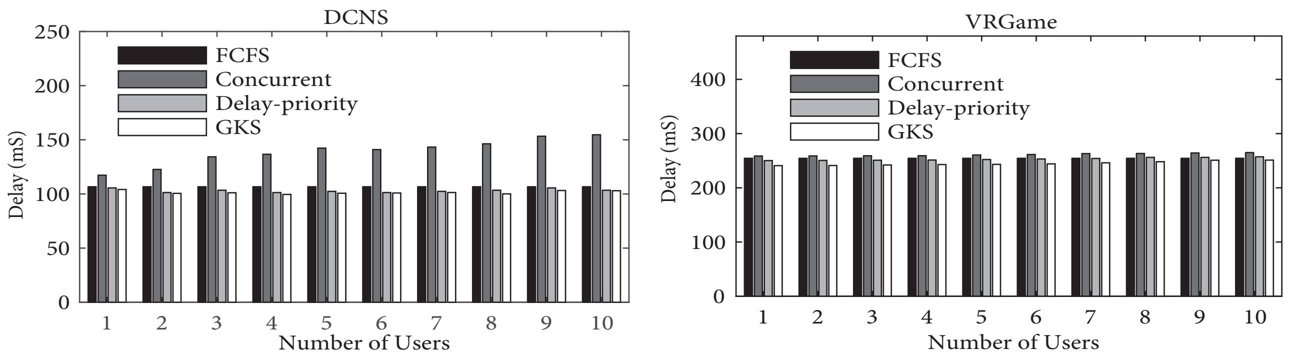
The tuple receipt time is calculated by Eq. (5), where $T1$ and $Delay$ are defined in Eq. (3). $C2$ is the number of receipt tuple types.

Delay is one of the best important parameters in IoT applications. We present this parameter for two case studies as DCNS and VRGame. This value shows the speed of each scheduling method in resource allocation. The results indicate that the best delay of application loop in DCNS is 101.44 mS that is obtained by GKS algorithm. As shown in Figures 12a and 12b, the GKS reduces the delay of DCNS compared to FCFS by 4.87%, concurrent by 27.11%, and delay-priority by 1.53%. Also, this parameter is optimized by GKS compared to FCFS by 3.68%, concurrent by 6.23%, and delay-priority by 3.20%. Table 9 shows the simulation results based on the number of users. To obtain these results, the number of areas or departments is equal to 3 and the number of camera or mobile is equal to 6.

As a result, the data transmission from WSNs to Cloud by FNs is a high-performance method. Our approach is suitable for local storage and process in the edge devices with high performance. We show the low energy consumption of the GKS than the FCFS, concurrent, delay-priority algorithms in resource scheduling by iFogsim. Fog computing by our scheduling method is suitable for IoT applications.

5. Conclusion

Since cloud is located far away from the sensors, data transmission has a lot of overhead. FC has an acceptable architecture for decreasing delay of the sensor applications. We optimized the iFogsim packages by a new



(a) Delay of application loop for DCNS

(b) Delay of application loop for VRGame

Figure 12. Comparison of delay of application loop by FCFS, concurrent, delay-priority, and GKS based on the number of users

Table 9. Simulation results based on the number of users. (Area/department=3 and camera/mobile=6).

Method	Type	Energy consumption		Total execution cost		Delay	
		DCNS	VRGame	DCNS	VRGame	DCNS	VRGame
FCFS	Min	$1.54 * 10^7$	$1.63 * 10^7$	$2.89 * 10^6$	$5.87 * 10^4$	106.64	254.39
	Max	$1.54 * 10^7$	$1.63 * 10^7$	$2.90 * 10^6$	$5.92 * 10^4$	106.64	254.47
	Avg	$1.54 * 10^7$	$1.63 * 10^7$	$2.89 * 10^6$	$5.90 * 10^4$	106.64	254.43
	SD	$1.97 * 10^3$	$7.01 * 10^3$	$2.79 * 10^3$	$1.39 * 10^2$	$2 * 10^{-5}$	$2.10 * 10^{-2}$
Concurrent	Min	$1.69 * 10^7$	$1.64 * 10^7$	$3.18 * 10^6$	$5.91 * 10^4$	117.31	258.64
	Max	$2.15 * 10^7$	$1.67 * 10^7$	$4.53 * 10^6$	$7.11 * 10^4$	154.63	264.99
	Avg	$2.0 * 10^7$	$1.65 * 10^7$	$3.87 * 10^6$	$6.55 * 10^4$	139.17	261.35
	SD	$1.63 * 10^6$	$1.34 * 10^5$	$4.63 * 10^5$	$3.90 * 10^3$	11.41	2.30
Delay-priority	Min	$1.46 * 10^7$	$1.37 * 10^7$	$2.74 * 10^6$	$5.09 * 10^4$	101.31	250.21
	Max	$1.52 * 10^7$	$1.37 * 10^7$	$2.87 * 10^6$	$5.93 * 10^4$	105.58	257.14
	Avg	$1.49 * 10^7$	$1.37 * 10^7$	$2.81 * 10^6$	$5.60 * 10^4$	103.02	253.17
	SD	$2.16 * 10^5$	$2.25 * 10^3$	$4.17 * 10^4$	$2.97 * 10^3$	1.52	2.44
GKS	Min	$1.33 * 10^7$	$1.31 * 10^7$	$1.95 * 10^6$	$3.83 * 10^4$	99.55	240.90
	Max	$1.33 * 10^7$	$1.31 * 10^7$	$1.96 * 10^6$	$5.23 * 10^4$	104.11	251.12
	Avg	$1.33 * 10^7$	$1.31 * 10^7$	$1.95 * 10^6$	$4.57 * 10^4$	101.44	245.07
	SD	$4.49 * 10^{-5}$	$1.84 * 10^3$	$5.48 * 10^3$	$4.52 * 10^3$	1.41	3.63

scheduling algorithm in FDs. Our proposed method is based on GKS algorithm that was simulated by two tracker case studies based on EEG signal and camera sensors with actuators.

The total execution cost, energy consumption, and delay of application loop in DCNS and VRGame are improved by GKS than FCFS, concurrent, and delay-priority algorithms. In total execution cost, our proposed algorithm is better than the concurrent algorithm by 45.3% in DCNS. Also, this parameter by GKS has a better result than delay-priority by 39.3%. In the energy consumption, GKS is better than concurrent algorithm by 17.5% in DCNS and 29.1% in VRGame. In delay of application loop, GKS is better than concurrent algorithm by 27.1% in DCNS and 6.23% in VRGame.

Considering FC as a new paradigm and its high performance as a suitable platform for the IoT, further research in this field will be very helpful. Among the open issues in scheduling and managing resources in FC, we can point to fault-tolerant, security, and trust. As future work, we will research the metaheuristic and statistic methods for scheduling with security and fault-tolerant considerations in other applications.

References

[1] Gupta H, Dastjerdi AV, Ghosh SK, Buyya R. ifogsim: toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. Software: Practice and Experience 2017; 47(9): 1275-1296. doi: 10.1002/spe.2509

[2] Hu P, Dhelim S, Ning H, Qiu T. Survey on fog computing: architecture, key technologies, applications and open issues. Journal of Network and Computer Applications 2017; 98: 27-42. doi: 10.1016/j.jnca.2017.09.002

[3] Yu W, Liang F, He X, Hatcher WG, Lu C et al. A survey on the edge computing for the internet of things. IEEE access 2018; 6: 6900-6919. doi: 10.1109/ACCESS.2017.2778504

[4] Rahmani AM, Gia TN, Negash B, Anzanpour A, Azimi I et al. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: a fog computing approach. Future Generation Computer Systems 2018; 78(2): 641-658. doi: 10.1016/j.future.2017.02.014

[5] Aazam M, St-Hilaire M, Lung CH, Lambadaris I. Pre-fog: Iot trace based probabilistic resource estimation at fog. In: 13th IEEE Annual, Consumer Communications & Networking Conference (CCNC); Las Vegas, NV, USA; 2016. pp. 12-17.

[6] Mahmud R, Kotagiri R, Buyya R. Fog computing: a taxonomy, survey and future directions. Internet of everything. Springer 2018; 103-130. doi: 10.1007/978-981-10-5861-5_5

[7] Singh SP, Nayyar A, Kumar R, Sharma A. Fog computing: from architecture to edge computing and big data processing. The Journal of Supercomputing 2018; 1-36. doi: 10.1007/s11227-018-2701-2

[8] Jiang Y, Huang A, Tsang DH. Challenges and solutions in fog computing orchestration. IEEE Network 2018; 32(3): 122-129. doi: 10.1109/MNET.2017.1700271

[9] Satyanarayanan M, Bahl V, Caceres R, Davies N. The case for vm-based cloudlets in mobile computing. IEEE pervasive Computing 2009; 8(4): 14-23. doi: 10.1109/MPRV.2009.82

[10] Gupta P, Ghrera SP. Trust and deadline aware scheduling algorithm for cloud infrastructure using ant colony optimization. In: International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH); Noida, India; 2016. pp. 187-191.

[11] Rodriguez MA, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Transactions on Cloud Computing 2014; 2(2): 222-235. doi: 10.1109/TCC.2014.2314655

[12] Lv J, Wang X, Huang M, Cheng H, Li F. Solving 0-1 knapsack problem by greedy degree and expectation efficiency. Applied Soft Computing 2016; 41: 94-103. doi: 10.1016/j.asoc.2015.11.045

[13] Bittencourt LF, Diaz-Montes J, Buyya R, Rana OF, Parashar M. Mobility-aware application scheduling in fog computing. IEEE Cloud Computing 2017; 4(2): 26-35. doi: 10.1109/MCC.2017.27

[14] Mathew T, Sekaran KC, Jose J. Study and analysis of various task scheduling algorithms in the cloud computing environment. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI); New Delhi, India: IEEE; 2014. pp. 658-664.

[15] Tsai CW, Huang WC, Chiang MH, Chiang MC, Yang CS. A hyper-heuristic scheduling algorithm for cloud. IEEE Transactions on Cloud Computing 2014; 2(2): 236-250. doi: 10.1109/TCC.2014.2315797

[16] Patil N, Aeloor D. A review-different scheduling algorithms in cloud computing environment. In: 11th International Conference on Intelligent Systems and Control (ISCO); Coimbatore, India: IEEE; 2017. pp. 182-185.

[17] Rodriguez MA, Buyya R. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. Concurrency and Computation: Practice and Experience 2017; 29(8). doi: 10.1002/cpe.4041

[18] Frincu ME, Genaud S, Gossa J. Comparing provisioning and scheduling strategies for workflows on clouds. In: IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW); Cambridge, MA, USA; 2013. pp. 2101-2110.

[19] Malawski M, Figiela K, Bubak M, Deelman E, Nabrzyski J. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. Scientific Programming 2015; 2015: 1-13. doi: 10.1155/2015/680271

[20] Durillo JJ, Prodan R. Multi-objective workflow scheduling in amazon ec2. Cluster computing 2014; 17(2): 169-189. doi: 10.1007/s10586-013-0325-0

[21] Poola D, Ramamohanarao K, Buyya R. Fault-tolerant workflow scheduling using spot instances on clouds. Procedia Computer Science 2014; 29: 523-533. doi: 0.1016/j.procs.2014.05.047

[22] Zhu X, Wang J, Guo H, Zhu D, Yang LT et al. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE Transactions on Parallel and Distributed Systems 2016; 27(12): 3501-3517. doi: 10.1109/TPDS.2016.2543731

[23] Li C, Bai J, Tang J. Joint optimization of data placement and scheduling for improving user experience in edge computing. Journal of Parallel and Distributed Computing 2019; 125: 93-105. doi: 10.1016/j.jpdc.2018.11.006

[24] Pham XQ, Huh EN. Towards task scheduling in a cloud-fog computing system. In: 18th Asia-Pacific Network Operations and Management Symposium (APNOMS); Kanazawa, Japan: IEEE; 2016. pp. 1-4.

[25] Sun Y, Dang T, Zhou J. User scheduling and cluster formation in fog computing based radio access networks. In: IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB); Nanjing, China; 2016. pp. 1-4.

[26] Chen X, Wang L. Exploring fog computing-based adaptive vehicular data scheduling policies through a compositional formal method—pepa. IEEE Communications Letters 2017; 21(4): 745-748. doi: 10.1109/LCOMM.2016.2647595

[27] Zahaf HE, Benyamina AEH, Olejnik R, Lipari G. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. Journal of Systems Architecture 2017; 74: 46-60. doi: 10.1016/j.sysarc.2017.01.002

[28] Lao F, Zhang X, Guo Z. Parallelizing video transcoding using map-reduce-based cloud computing. In: IEEE International Symposium on Circuits and Systems (ISCAS); Seoul, South Korea; 2012. pp. 2905-2908.

[29] Fang W, Zhou W, Li Y, Yao X, Xue F et al. A distributed admm approach for energy-efficient resource allocation in mobile edge computing. Turkish Journal of Electrical Engineering & Computer Sciences 2018; 26(6): 3335-3344. doi: 10.3906/elk-1806-112

[30] Pham XQ, Man ND, Tri NDT, Thai NQ, Huh EN. A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. International Journal of Distributed Sensor Networks 2017; 13(11): 1-16. doi: 10.1177/1550147717742073

[31] Kamal MB, Javaid N, Naqvi SAA, Butt H, Saif T et al. Heuristic min-conflicts optimizing technique for load balancing on fog computing. In: International Conference on Intelligent Networking and Collaborative Systems, Springer; Bratislava, Slovakia; 2018. pp.207-219.

[32] Xu X, Liu Q, Qi L, Yuan Y, Dou W et al. A heuristic virtual machine scheduling method for load balancing in fog-cloud computing. In: IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS); Omaha, NE, USA; 2018. pp. 83-88.

[33] Chen YA, Walters JP, Crago SP. Load balancing for minimizing deadline misses and total runtime for connected car systems in fog computing. In: IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC); Guangzhou, China; 2017. pp. 683-690.

[34] Yang Y, Wang K, Zhang G, Chen X, Luo X et al. Meets: Maximal energy efficient task scheduling in homogeneous fog networks. IEEE Internet of Things Journal 2018; 5(5): 4076-4087. doi: 10.1109/JIOT.2018.2846644

[35] Venkatraman S, Selvagopal D. An optimized multiobjective cpu job scheduling using evolutionary algorithms. Turkish Journal of Electrical Engineering & Computer Sciences 2018; 26(1): 101-114. doi: 10.3906/elk-1701-22

[36] Singh P, Dutta M, Aggarwal N. A review of task scheduling based on meta-heuristics approach in cloud computing. Knowledge and Information Systems 2017; 52(1): 1-51. doi: 10.1007/s10115-017-1044-2

[37] Sheff I, Magrino T, Liu J, Myers AC, Van Renesse R. Safe serializable secure scheduling: Transactions and the trade-off between security and consistency. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; Vienna, Austria; 2016. pp. 229-241.

[38] Domanal S, Guddeti RM, Buyya R. A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment. IEEE Transactions on Services Computing 2017. doi: 10.1109/TSC.2017.2679738

[39] Wang T, Wei X, Tang C, Fan J. Efficient multi-tasks scheduling algorithm in mobile cloud computing with time constraints. Peer-to-Peer Networking and Applications 2018; 11(4): 793-807. doi: 10.1007/s12083-017-0561-9

[40] Bitam S, Zeadally S, Mellouk A. Fog computing job scheduling optimization based on bees swarm. Enterprise Information Systems 2017; 12(4): 373-397. doi: 10.1080/17517575.2017.1304579

[41] Rodriguez MA, Buyya R. A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. In: IEEE 2015 44th International Conference on Parallel Processing (ICPP); Beijing, China; 2015. pp. 839-848.

[42] Rahim S, Khan SA, Javaid N, Shaheen N, Iqbal Z et al. Towards multiple knapsack problem approach for home energy management in smart grid. In: IEEE 18th International Conference on Network-Based Information Systems (NBiS); Taipei, Taiwan; 2015. pp. 48-52.

[43] Chen S, Wu J, Lu Z. A cloud computing resource scheduling policy based on genetic algorithm with multiple fitness. In: IEEE 2012 12th International Conference on Computer and Information Technology (CIT); Chengdu, China; 2012. pp. 177-184.

[44] Sun Y, Lin F, Xu H. Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii. Wireless Personal Communications 2018; 102(2): 1369-1385. doi: 10.1007/s11277-017-5200-5

[45] Gai K, Qiu M. Optimal resource allocation using reinforcement learning for iot content-centric services. Applied Soft Computing 2018; 70: 12-21. doi: 10.1016/j.asoc.2018.03.056

[46] Zhang Q, Lin M, Yang LT, Chen Z, Khan SU et al. A double deep q-learning model for energy-efficient edge scheduling. Transactions on Services Computing 2018. doi: 10.1109/TSC.2018.2867482

[47] Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience 2011; 41(1): 23-50. doi: 10.1002/spe.995

[48] Marler RT, Arora JS. The weighted sum method for multi-objective optimization: new insights. Structural and multidisciplinary optimization 2010; 41(6): 853-862. doi: 10.1007/s00158-009-0460-7

[49] Zao JK, Gan TT, You CK, Méndez SJR, Chung CE et al. Augmented brain computer interaction based on fog computing and linked data. In: IEEE 2014 International Conference on Intelligent Environments (IE); Shanghai, China; 2014. pp. 374-377.