Computer Communications 153 (2020) 217–228

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Efficient resource management and workload allocation in fog–cloud computing paradigm in IoT using learning classifier systems

Mahdi Abbasi [a,*], Mina Yaghoobikia [a], Milad Rafiee [a], Alireza Jolfaei [b], Mohammad R. Khosravi [c,d]

[a] Department of Computer Engineering, Engineering Faculty, Bu-Ali Sina University, Hamedan, Iran
[b] Department of Computing, Macquarie University, Sydney, NSW, Australia
[c] Department of Computer Engineering, Persian Gulf University, Bushehr, Iran
[d] Telecommunications Group, Department of Electrical and Electronic Engineering, Shiraz University of Technology, Shiraz, Iran

ARTICLE INFO

Keywords:
Fog computing
Internet of Things
Machine learning
Load distribution
Renewable power source
Cost

ABSTRACT

With the rapid growth in network-connected computing devices, the Internet of Things (IoT) has progressed in terms of size and speed. Subsequently, the amount of produced data and computation loads has increased dramatically. A solution to handle this huge volume of workloads is cloud computing in which a considerable delay exists in the processing load and this has remained a concern in the field of distributed computing networks. Processing workloads at the edge of the network can reduce the response time while at the same time imposing energy constraints by bringing the task of load processing from data centers, which are supplied by electrical energy sources, to the network edges which are only supported by limited energies of batteries. Therefore, workloads need to be distributed evenly between the clouds and the edges of the network. In this paper, two methods based on XCS learning classifier systems (LCS), namely, XCS and BCM-XCS, are proposed to balance the power consumption at the edge of the network and to reduce delays in the processing of workloads. The results of our experiments are indicative of the superiority of BCM-XCS over the basic XCS-based method. The proposed methods distribute the workloads in a way that the delay in their processing and the communication delay between the cloud and fog nodes are both minimized. In addition to considerable advantages in controlling the fluctuations of the processing delay, the proposed methods can simultaneously reduce the processing delay by 42% by using a moderate power consumption at the edge of the network. The proposed methods can also recharge the renewable batteries used at the edge of the network about 18 percent more than the best state-of-the-art method.

1. Introduction

The Internet of Things (IoT) technology has enabled billions of devices including, sensors, actuators, and other objects to get connected to the Internet [1]. These devices generate massive amounts of processable data that may limit the computational and storage resources [2]. With the advent of cloud computing, the storage and processing restrictions have been eliminated. Over the past decade, computation loads and data volumes in clouds have been on the rise. The results of these computations and control data are routinely moved to centralized data centers and network cores [3]. As a result, cloud computing faces new challenges such as increasing workload latency. To this end, a new technology named edge computing has been introduced to allow data processing with low latency that is closer to users on the network edge. Edge-based devices and network cores, such as base stations, modems, and routers, welcome computations and storage spaces requested by users, thereby acting as a replacement to clouds. As a direct consequence of bringing the processing resources and storage spaces closer to the end-users, the transmission delay is reduced. Fig. 1 shows the data processing levels of IoT devices on the network.

As shown in Fig. 1, at the bottom layer, objects and sensors produce processable and storable data. Network edge devices are at the top of these objects with less processing power than clouds. Due to the proximity of these devices to users, requests see minimal emission delay and are responded more quickly. At the top layer, there are potent routers that are located at the core of the network and can process requests without sending them to the cloud. As we move from things to data centers, the delay increases. At the top layer, massive data centers are distributed across the globe, providing high processing and storage capability. These data centers consume great electrical energy due to the high processing volume and power. However, most devices at the edge of the network can work with very little power or battery.

The processing of IoT data in the cloud faces several challenges, including [4,5]:

* Corresponding author.
E-mail address: abbasi@basu.ac.ir (M. Abbasi).

https://doi.org/10.1016/j.comcom.2020.02.017
Received 27 November 2019; Received in revised form 31 January 2020; Accepted 4 February 2020
Available online 6 February 2020

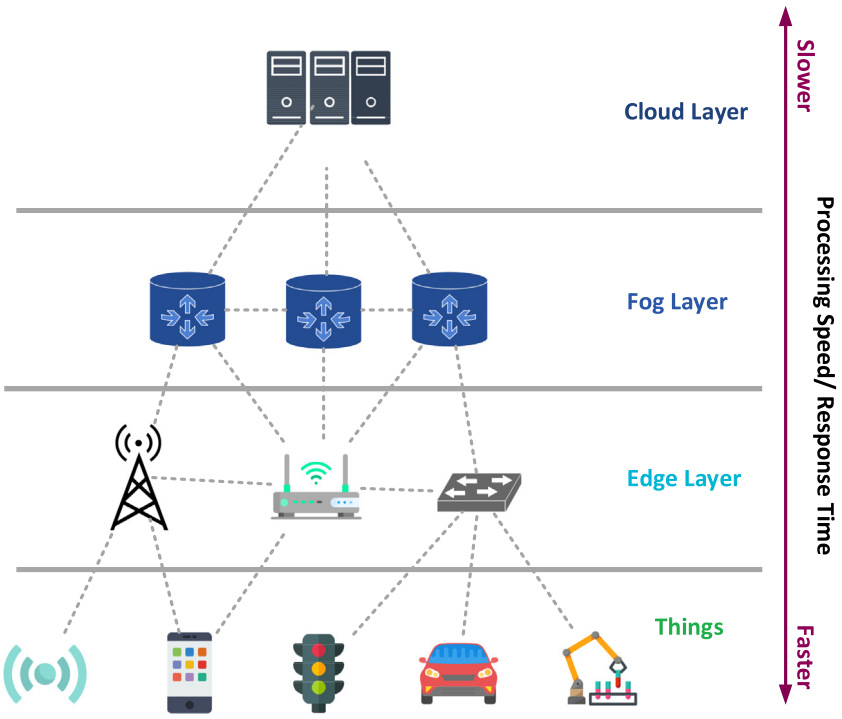

Fig. 1. IoT Data processing layer stack.

- Delay: Delivering data from IoT devices to data centers, due to the long distance, imposes considerable propagation delays.
- Bandwidth limitation: With the rapid increase in the number of Internet-connected objects, the speed of data generation increases at an exponential rate. For example, a machine can generate a few megabytes of data per second. As a result, the increase in the transmission rate and the volume of data produced, places bandwidth limitations on communication links.
- Device resource constraints: Most IoT devices have very limited resources. For example, sensors, actuators, controllers, and CCTVs are unable to rely on themselves to meet all their computational needs due to resource constraints. On the other hand, their permanent dependence on the cloud is very costly as most of their interactions require complex processing resources and protocols.
- Security: Security is a challenging issue. Devices need to be identified and authenticated to access the stored information or processing resources in the cloud.

Fog computing is a type of distributed processing that is a good alternative to cloud computing. Fog computing has more processing power than edge computing, and it has less computation capability compared to cloud computing. However, the required electrical energy for performing calculations is very expensive. Fog computing nodes often supply their energy through rechargeable batteries. This type of energy source is very limited and needs to be recharged after a while [6]. Therefore, renewable energies are used as the principal power supply or the only energy source at the edge of the network.

Given the importance of latency and energy consumption, the loads must be continuously distributed between the cloud and the fog so that the optimal values could be achieved on these two measures. To this end, research has been conducted to reduce energy consumption and delay. In this research, methods such as clustering algorithms, reinforcement learning, and heuristic algorithms have been used [7–9]. Most existing methods for distributing workload have not created a balance between cost and energy consumption, only addressing one of the two criteria.

Recently, learning classifier systems (LCS) have attracted the attention of researchers in different fields as a smart solution for finding the optimal state [10–12]. In the present study, for the first time, an intelligent LCS, naming extended classifier system (XCS) has been used to reach an optimal state in balancing workloads in fog computing. To direct the search for a better state, the LCS uses an evolutionary algorithm to search in the space of existing states and uses another learning process for selecting the best state. By adapting the environmental conditions (such as input workload and battery level) with the input parameters of XCS, the algorithm can optimize and balance energy and delay.

In the XCS algorithm, if the collection of learned actions is not complete, other actions are randomly created by the covering process in the form of new classifiers. These new classifiers can affect the result of subsequent entries. Although these random actions prevent optimal local trapping, increasing the number of these random actions may slow down the learning process. To avoid the generation of the additional random classifiers, a cache has been added to the XCS system. Keeping the faced conditions together with the best corresponding actions in this cache memory can prevent excessive random actions in the system. In addition, the performance of the basic LCS is improved by adding a memory that keeps the optimal states.

In the proposed method, workloads are distributed in a way that their processing delay and the communication delay between the cloud and fog nodes are both minimized. Also, in the proposed method, the energy consumption is optimized. Our experiments demonstrate that the proposed method is able to meet both criteria.

The paper is organized as follows. Section 2 reviews the literature and discusses some of the related issues. Section 3 describes the system model and formulates the problem. It also introduces a modified LCS to balance the load distribution at the edge of the network. In Section 4, the proposed methods are implemented and evaluated compared to the state-of-the-art methods. Finally, Section 5 concludes the paper and highlights the future research directions.

2. Related works

In recent years, particularly since 2016, a large number of studies have been conducted on the distribution of computation loads over the network edge. This section briefly reviews some of the works that

address the optimization of power consumption and computation cost in fog computing.

The earliest studies, which date back to 2016, investigated the possibility of combining renewable energy sources into the power supply of computations at the edges of cellphone networks [7]. These studies resulted in several learning algorithms for resource management and allocation of computation load. One of the algorithms can learn the policy of discharging dynamic workloads into the cloud and creating an edge server. The algorithm is aimed at minimizing the long-term costs of the system, which include the costs of service delay and operating. Its advantage over other algorithms is optimized learning and performance. Simulation results suggest that the power consumption of this algorithm is reduced by 25% in comparison with other algorithms while its learning rate is still low. At the time of the development of this algorithm, several methods were proposed for the optimization of power consumption, but the speed of the proposed methods was still relatively low. Therefore, there was a need to focus on the time cost in any method to be developed.

As a result, Xu et al. modified and improved their previous learning algorithm by changing its learning model [13]. To accelerate the learning process, they developed a reinforcement learning algorithm based on Power Spectral Density (PDS) that would optimize the discharging of loads and the automatization of resource allocation. The algorithm worked efficiently, even with unknown system parameters. Xu et al. evaluated their algorithm both in a simulated and a real environment. Their results showed that their proposed design could significantly optimize the performance of edge computations. Allocation of workloads proportional to unpredictable, renewable power sources was among the advantages of this algorithm. In spite of its success in optimizing the power consumption of computation at the network edge, the algorithm was unable to distribute workloads evenly among the computing nodes.

In [6], Wu et al. proposed an algorithm called GLOBE. To optimize the performance of network-edge stations, they combined two mechanisms, namely, geographic load balancing and input load control. GLOBE functions online without any need for information about the future states of the system and can handle significant challenges concerning the battery status and power limitations. The simulation results suggest that in comparison with offline algorithms, GLOBE can reduce the costs by 50% ('offline' algorithms referring to those that have complete information about the future state of the system); and GLOBE can establish a balance between battery capacity and system performance. GLOBE succeeded in slightly improving the distribution of computation loads among the network nodes, but it was still far from an optimum level.

To improve the deficiencies of the GLOBE algorithm, Wan et al. proposed a method for balancing and planning workloads to reduce power consumption in fog computing [8]. After presenting some formulas for a model of power consumption related to the volume of work on each fog node, they developed an optimization function to balance the loads in a clustered model. They improved this function using a batch optimization algorithm and finally proposed a multi-purpose system to resolve the scheduling of clustered distribution. By experimenting with robots and Raspberry Pi, they arrived at two conclusions. Firstly, in the experimental environment, the robots had a better performance, and the workloads were distributed more evenly among them. Therefore, as power consumption was a function of the workload in each node, it could also be more easily balanced among the nodes.

In [9], Zeng et al. investigated the effect of different power types on fog computing to achieve an efficient structure for providing power services. They designed a fog network called Cyber–Physical Fog System (CPFS) based on Cyber–Physical System (CPS) which used green energy and their focus was on resource rate control, load balancer, and services. They presented the question of energy productivity as an equation of mixed-integer linear programming and proved that it is an NP-hard problem. Then they proposed an exploratory algorithm to overcome this computation complexity. Their simulation results suggest that the energy productivity of this algorithm is near-optimal.

None of the methods discussed above have comprehensively addressed power consumption for all types of workloads. In addition, they have not created a balance between costs and power consumption in distributing the workload among network nodes. Therefore, many researchers are now attempting to study this issue. The present study follows the same research trend.

3. The proposed method

To overcome the deficiencies of the methods discussed in the previous section, we developed a new method to improve the allocation of workloads and balance the consumption of power on the network edge. In this section, we simulate a fog computing system using the model proposed in [13]. This system is modeled in four dimensions: workload, delay, power consumption, and battery status. For each dimension, several equations are presented according to our proposed work. Then, two LCS models are presented. One of these systems uses the XCS learning classifier architecture, and the other one, named here BCM-XCS, possesses a memory for keeping the best classifiers. Finally, the two models are compared with each other and with the proposed methods in the literature.

3.1. System modeling

For the main scenario, we model an edge system consisting of a base station and a set of edge servers that are physically next to each other. Each processing resource on the network edge (i.e., fog servers) has a battery with limited capacity. Therefore, a shared power supply is usually used in the network so that the workloads are sent to the cloud in cases where the edge servers lack enough battery charge. The workload sent by the users to the edge is first received by the base station. The base station is responsible for deciding on the amount of workload that should be allocated to the edge and amount that should be assigned to the cloud. Table 1 summarizes the main parameters of the system model.

We formulate the edge system model in terms of workload, delay, power consumption, and battery status.

A. Workload

The workload is modeled as a discrete-time signal. That is, different values of the workload are represented at certain time instants of the sequence. The parameter $\lambda(t) \in [0,\ \lambda_{max}]$ denotes the rate at which the workload reaches the edge system at the moment t. Here, λ_{max} represents the maximum input load that can enter the edge system. Accordingly, the edge devices should set their computation capacity in each time instance. That is, the number of active servers which is $m(t) \in [0,\ M]$ should be determined at any time interval. Note that, this parameter is constant during an interval but may vary at different intervals. Here, M represents the maximum number of servers on the network edge system. The edge system decides how much of the workload ($\mu(t)$) is processed locally ($\mu(t) \leq \lambda(t)$). Then, the remaining part of the workload ($\lambda(t) - \mu(t)$) should be transmitted to the cloud and processed there. Fig. 2 illustrates how workloads reach the edge.

B. Delay

The edge system model has the following three types of delay.

B-1. Transmission delay: This type of delay, represented by $c_{wi}(t)$, appears in transmitting workloads on the wireless network. It is directly proportional to the input workload of the network (i.e. , $\lambda(t)$). The value of this delay is set to zero in our proposed scenario due to the physical proximity of the processing nodes.

B-2. Local processing delay: This type of delay, represented by $c_{lo}(t)$, appears in the local processing of workloads on the network edge system. This delay can be calculated based on $m(t)$ and $\mu(t)$, and hence is represented as $c_{lo}(\mu(t), m(t))$. The value of this delay depends on the queue management mechanism in the edge servers. Following [13], the M/G/1 mechanism is used for queue

Table 1
Used parameters.

Symbol	Description	Symbol	Description
$\lambda(t)$	The total rate of the input load	$c_{delay}(t)$	The total cost of delays
$\mu(t)$	The amount of workload processed locally	$c_{back}(t)$	The cost of using the supporting power supply
$m(t)$	The number of active servers on the network edge	$d_{op}(t)$	Power consumption for operational tasks on the network edge
$c_{lo}(t)$	The cost of delay in processing workloads on the network edge	$c_{comp}(t)$	Power consumption for processing loads on the network edge
$c_{off}(t)$	The cost of delay in sending workloads to the cloud	$d(t)$	Total power consumption
$g(t)$	The total renewable power received	$b(t)$	Battery level at the network edge
$h(t)$	The congestion status of the network	$s(t)$	System status

management in the servers of the edge system. As a result, the delay in processing at the network edge is obtained through the following equation:

$$c_{lo}(\mu(t), m(t)) = \frac{\mu(t)}{m(t)\cdot k - \mu(t)} \quad (1)$$

In the above equation, k denotes the processing capacity of each server.

B-3. Offloading delay: This type of delay, represented by $c_{off}(t)$, is the delay in transmitting the remaining load to the cloud. The magnitude of this delay depends on the intensity of the congestion in the communication network and the number of offloaded workloads. Given the intensity of the congestion represented by $h(t)$, the magnitude of offloading delay is calculated through the following equation:

$$c_{off}(h(t), \mu(t), \lambda(t)) = (\lambda(t) - \mu(t))\, h(t) \quad (2)$$

To compute the overall delay cost, the magnitude of all three delays is added together.

$$c_{delay}(h(t), \lambda(t), \mu(t), m(t)) = c_{wi}(\lambda(t)) + c_{lo}(\mu(t), m(t)) + c_{off}(h(t), \mu(t), \lambda(t)) \quad (3)$$

As mentioned earlier in this section, since $c_{wi}(\lambda(t))$ is negligible, it is not taken into account.

C. Power Consumption

The total power consumption during an interval is divided into two parts,

C-1. Operating power: This part of power, represented by $d_{op}(t)$, is consumed in basic operations of the edge system, including receiving the workloads and performing basic processes on them. The operating power is not consumed for transmitting workloads to the cloud or processing them at the edge. It is composed of two parts:

$$d_{op}(\lambda(t)) = d_{sta} + d_{dyn}(\lambda(t)) \quad (4)$$

The first part, (d_{sta}), represents the static power consumption of the network edge and the second part, ($d_{dyn}(\lambda(t))$), is the dynamic power consumption which varies with the input workload of the network. $d_{dyn}(\lambda(t))$ is neglected in the proposed edge system model due to the physical proximity of the computing units.

C-2. Computing power: This part of power, represented by $d_{com}(t)$, is required for the computation of workloads at the edge system at time t. The magnitude of this power depends on the size of the workload allocated to the edge system $\mu(t)$, and the number of edge servers $m(t)$.

The total power consumption of the edge system is obtained by the following equation:

$$d(\lambda(t), \mu(t), m(t)) = d_{op}(\lambda(t)) + d_{com}(\mu(t), m(t)) \quad (5)$$

A large number of devices are being connected to the IoT every day. This sheer number of IoT implementations imposes a significant amount of energy consumption for driving IoT devices. In other words, IoT devices are becoming the leading energy consumers in the digital technology world. With the diminishing of brown energy resources on earth and the potentially harmful environmental impacts of carbon emissions, the use of green (renewable) energy for IoT devices has become very promising, particularly in the sustainable development of modern societies [14,15]. Therefore, renewable energies including wind energy or solar energy, represented by $g(t)$, may be imported in this model and used as green power supply. The magnitude of this power supply varies with the environmental conditions.

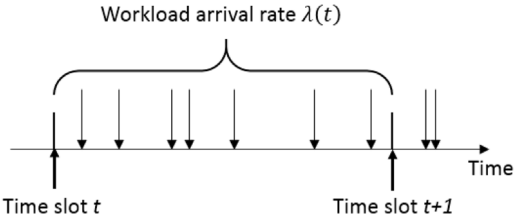

Fig. 2. The arrival of the workloads in a typical time interval.

D. Battery Status

As mentioned earlier, one battery with limited capacity is allocated to each edge server. The number of batteries at the network edge is $b(t) \in [0, B]$, where B denotes the maximum battery capacity at the edge. These batteries are recharged by solar and wind power. The battery energy level is initially zero. To control the battery energy level, the amount of workload at the edge system denoted by $\mu(t)$ is controlled according to the following conditions:

D-1. When $b(t) \leq d_{op}(\lambda(t))$, no processing is allowed at the network edge. Since the battery of the edge system cannot provide the required power, the entire load $\lambda(t)$ is sent to the cloud so that the battery could be recharged. In this case, the supporting power supply is used for sending the workloads to the cloud. The cost of this operation is calculated by the following equation:

$$c_{bak}(\lambda(t)) = \varphi . d_{op}(\lambda(t)) \quad (6)$$

Here, φ represents the cost coefficient of the supporting power supply. In the next interval, the renewable power source will recharge the battery level according to the following equation.

$$b(t+1) = b(t) + g(t) \quad (7)$$

This state is illustrated by Case 2 in Fig. 3.

D-2. When the battery energy level is sufficient, a part of the workload is processed at the edge system and the rest of it is offloaded to the cloud. Thus, the battery energy level in the next interval is calculated by the following equation:

$$b(t+1) = b(t) + g(t) - d(\lambda(t), \mu(t), m(t)) \quad (8)$$

The operational cost of the battery in this state is modeled by the following equation:

$$c_{battery}(t) = \omega . \max\{d(\lambda(t), \mu(t), m(t)) - g(t), 0\} \quad (9)$$

where $\omega > 0$ is the operational cost of one battery unit. This state illustrated by Case 1 in Fig. 3.

The proposed edge system model is shown in Fig. 4. In this model, a number of requests $\lambda(t)$ arrives from users' edge devices. The task of

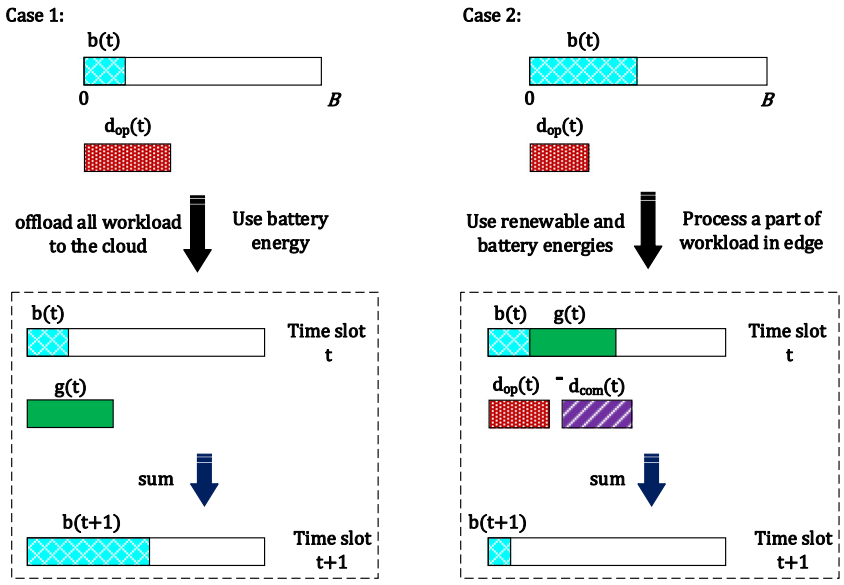

Fig. 3. Two modes of battery status.

the base station is to distribute loads between fog servers and the cloud. The amount of workload that can be processed locally by the fog servers is calculated. Then, the base station sends one part of the workload to fog servers and the residual $\lambda(t) - \mu(t)$ to the cloud. Offloading the remained workload to cloud burdens the network and imposes congestions. The congestion $h(t)$ is measured at each time interval to make more appropriate decisions. On the other hand, the renewable energy source denoted by $g(t)$ provides the energy needed for computing at the edge of the network at any given time. When the renewable energy exceeds the requirements, the remained part charges the batteries $b(t)$ at the edge. In the absence of renewable energy, the energy stored in these batteries is used.

3.2. Reinforcement learning solution

A reinforcement learning system can optimize the allocation of workload to the fog systems and the cloud. In the reinforcement learning system, an agent interacts heuristically with the environment and gradually learns how to solve the problem through the feedbacks of the actions applied to the environment. As a result, rewards and punishments are used as signals to strengthen or weaken specific behaviors.

Fig. 5 illustrates the general schema of a reinforcement learning system which consists of the following parts:

- **Environment:** The physical world in which the agent acts.
- **State:** The current state of the agent.
- **Reward:** The feedback received from the environment.
- **Policy:** A method for mapping the agent's state onto the action.

The aim of reinforcement learning is to find an appropriate policy that maximizes the total cumulative reward of the agent. For this purpose, the agent either repeats decisions that have worked well so far or makes novel decisions, expecting to gain even greater rewards. This is called exploration vs. exploitation trade-off.

In using a reinforcement learning system for the problem of optimal workload allocation in the proposed model, the state of the system at the beginning of each time-interval t is described by $s(t) \triangleq (\lambda(t), g(t), h(t), b(t))$. When the agent applies a new action to the environment, it predicts μ(t) as the future amount of processable workload at the edge of the network. Then, the battery status, $b(t+1)$ is calculated for the next state using following equations:

$$if \left(b(t) \leq d_p(\lambda(t))\right) then$$

$$b(t+1) = b(t) + g(t) \tag{10}$$

$$else$$

$$b(t+1) = b(t) + g(t) - d(\lambda(t), \mu(t), m(t))$$

After processing the allocated loads, the imposed cost is considered as a reward which is fed back from the environment. The cost is calculated by the following equation:

$$if \left(b(t) \leq d_p(\lambda(t))\right) then$$

$$c(t) = c_{delay}(h(t), \lambda(t), 0, 0) + c_{bak}(\lambda(t)) \tag{11}$$

$$else$$

$$c(t) = c_{delay}(h(t), \lambda(t), \mu(t), m(t)) + c_{battery}(t)$$

where $\mu(t) = m(t) = 0$. In the second part where the battery level is sufficient, the operational cost of the battery is replaced by the cost of the supporting power supply.

The Markov decision process (MDP) is a powerful mathematical technique that can describe the processes of the reinforcement learning systems. An MDP consists of a finite set of environment states S, a set of possible actions A in each state, a reward with real values R, and a transition model $P(s', s|a)$ where s is the current state, s' is the next state of the system, and a is the action to be performed on the environment.

To minimize the long-term costs, the costs at the network edge should be reduced at each interval [16]. The following equation estimates the long-term costs of the system.

$$C^*(s) = \min_{a \in A} \left\{ c(s,a) + \sigma \cdot \sum_{s' \epsilon S} P(s', s|a) \cdot C^*(s') \right\} \tag{12}$$

Here, $\delta < 1$ is a reduction factor which indicates that the future costs of the system would be less than the present costs. According to the above equation, the workload allocation policy would minimize the future cost of the system by selecting the appropriate value of $\mu(t)$.

3.3. Learning classifier system

Generally, an LCS is an intelligent agent interacting with an environment. The adaptability of the LCS directs it to choose the best action according to the present state of the environment. This property improves with new experiences of LCS. The learning from reinforcement, i.e. payoff, provided by the environment is the source of the improvement.

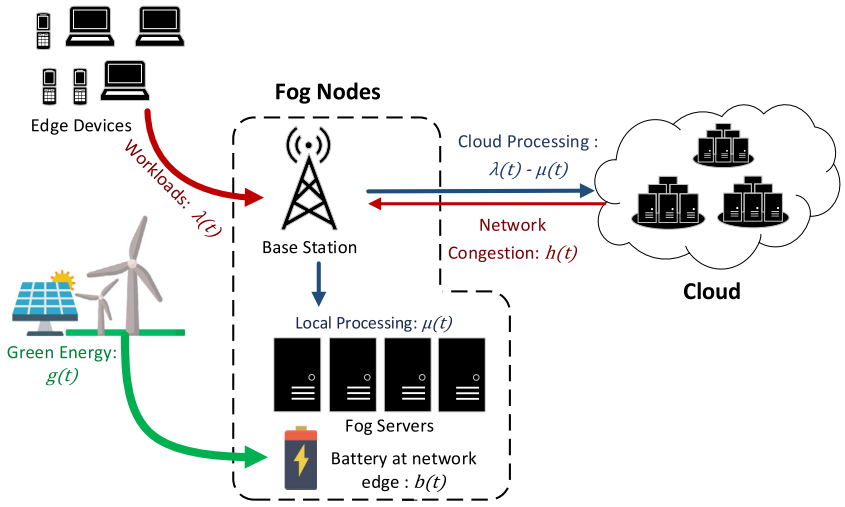

Fig. 4. The proposed model for processing workload in the edge-fog-cloud architecture.

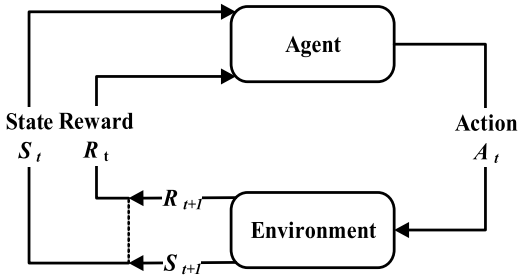

Fig. 5. The general schema of a reinforcement learning system.

An LCS aims to maximize the achieved environmental payoffs. For this purpose, the LCS tries to develop and enhance a set of dense and maximally inclusive "condition-action-payoff" rules, which are called classifiers. Each classifier informs the system in each state (or equivalently the input condition) about the number of payoffs for any available action. Therefore, LCS can be seen as a particular model of reinforcement learning that offers an approach to generalization with a well-defined information representation.

The idea of LCS was initially introduced by Holland to evolve genetic algorithms [17,18]. In 1995, Wilson et al. developed a new type of LCS, called XCS, in which the applicability of classifiers was only determined based on the accuracy of the performed actions [19,20]. In general, the objective of XCS is to capture, in the form of classifiers, the predictive regularities of the environment, so the system can attain payoff at optimal rates. Therefore, XCS has been widely used to solve real-world problems in areas such as control, robotics and data mining [12,21].

XCS evolves a population of classifiers, where each classifier has a rule and a set of associated parameters estimating the quality of the rule. Each rule consists of a condition and an action. At each time step, an input is presented and a match set is built, consisting of those classifiers whose conditions are satisfied by the input. If the match set is empty, or the number of covered actions is less than a preset threshold, covering occurs. The covering operator creates a new classifier with a condition matching the current example and an action that is chosen randomly from those not represented in match set. Once the match set is obtained, a payoff prediction $p(a)$ is computed for each action a available in match set. It is computed as a fitness-weighted average of the predictions of those classifiers advocating a. The winning action can be selected from a variety of regimes, ranging from the pure-explore style (random action) to the pure-exploit style (the action with the highest prediction). In classification problems, pure-explore is used during training while pure-exploit is used when the system is predicting unseen examples. The chosen action specifies the action set, formed by all the classifiers in the match set advocating this action.

Reinforcement learning in XCS includes updating three parameters. These three parameters, are error, predictability, and rule fitness. The criterion of the suitability of each classifier is updated according to the relative validity of the rule in the following five stages.

1– The error rate of each classifier j is first updated.

$$\varepsilon_j = \varepsilon_j + \beta(\left|\boldsymbol{reward} - \rho_j\right| - \varepsilon_j) \tag{13}$$

2– The predictions of each classifier are updated at the learning rate β, where $\mathbf{0} \le \beta \le \mathbf{1}$.

$$p_j = p_j + \beta(reward - p_j) \tag{14}$$

3– Then, the classifier's accuracy is computed as an inverse function of the classifier's error. In the following equation, the values of $\boldsymbol{v}, \boldsymbol{\alpha}$ *and* $\boldsymbol{\varepsilon_0}$ are constants that control the shape of the correctness function. The parameter $\boldsymbol{\varepsilon_0}$ ($\mathbf{0} < \boldsymbol{\varepsilon_0} < \mathbf{1}$) determines the threshold error under which a classifier is considered to be accurate. The parameters $\boldsymbol{v}$ *and* $\boldsymbol{\alpha}$ control the degree of decline in accuracy if the classifier is inaccurate.

$$k_j = \alpha\left(\frac{\varepsilon_0}{\varepsilon}\right)^{v} or, k_j = 1 \;\; if \;\; \varepsilon < \varepsilon_0 \tag{15}$$

4– A relative accuracy is calculated for the classifier system. Then, the relative accuracy $\boldsymbol{K'_j}$ is computed by dividing the accuracy by the total amount of accuracies in the action set.

$$K'_j = \frac{K_j}{\sum_{x\in[A]} K_x} \tag{16}$$

5– Relative accuracy is used to update the fitness value of each classifier. If the fitness is corrected $\frac{1}{\hat{a}}$ times, the following relationship is used:

$$\boldsymbol{F}_j = \boldsymbol{F}_j + \hat{a}(\boldsymbol{k'_j} - \boldsymbol{F}_j) \tag{17}$$

Otherwise, the fitness is set to the average value of the relative correctness until the present time. The amount of fitness in XCS is inversely related to the reward prediction error so that errors smaller than this value would not change the value of fitness. For each action in match set, the classifier predictions p are weighted by fitness F to get the system's net prediction in the prediction array.

In previous LCS systems, the genetic algorithm was initially applied to; but in XCS it is applied on action set to prevent inappropriate mixing of classifiers [21]. First, it selects two parents from the actual action set with a probability proportional to the fitness. Next, the parents are

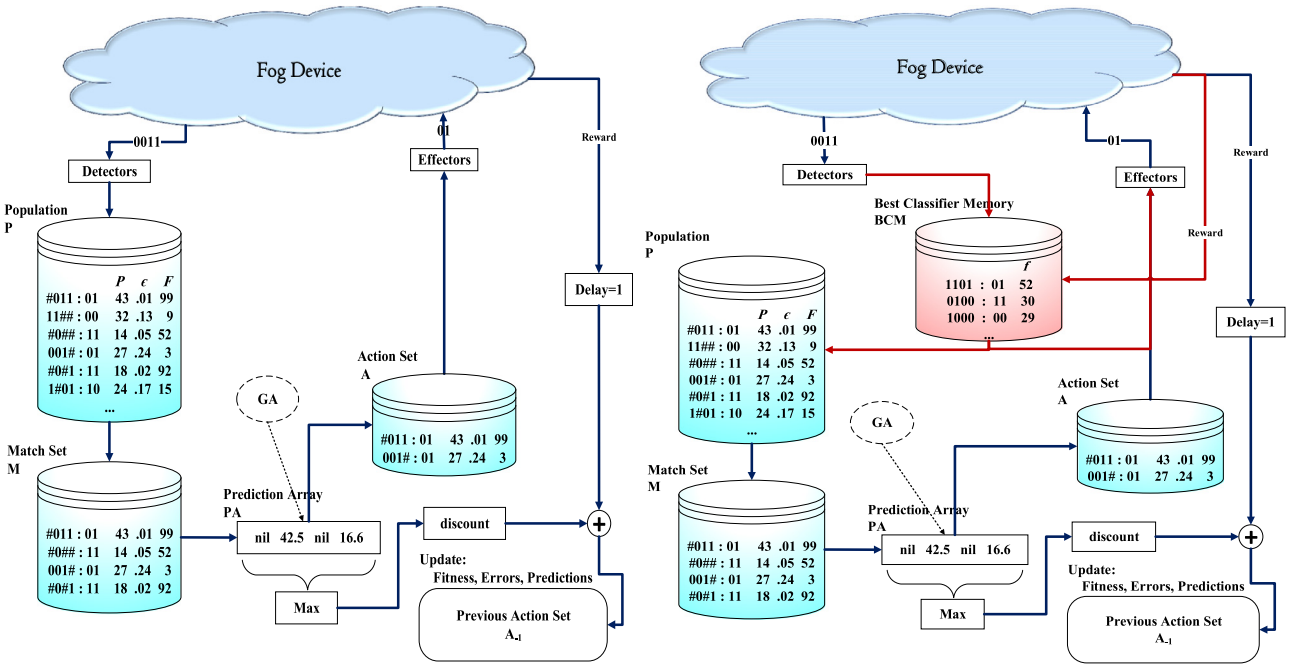

Fig. 6. The scheme of XCS and BCM-XCS.

crossed and mutated. The resulting offspring are introduced into the population.

Fig. 6a depicts a sample structure of XCS. In this structure, the system receives a four-bit input. Each classifier has a two-bit action and can apply one of four possible actions to the environment. In this example, the detector receives input 0011 from the environment. Then, the match set is created by storing the classifiers of population whose conditions match the input. In this example, unique actions in are 01 and 11. The prediction array is constructed using the accuracy and the fitness of classifiers whose actions have been previously seen in the match set.

For example, there are two classifiers with the action 01 in the match set. In each classifier, the values of P and F are multiplied and then added to the corresponding values of other classifiers. For the first classifier with the action 01, $F = 99$ and $P = 43$ are multiplied, which gives 4275. Similarly, for the third classifier with the action 01, the result is 81. Next, the sum of these two values is divided by the sum of the F values of both classifiers (i.e., 102), which gives 42.5. This value is stored in the prediction array entry corresponding the action 01. The following equation represents the mathematical formulation of this calculation. Here, n is the number of the existing classifiers with the action 01 in match set and is computed by Eq. (18).

$$\mathrm{PA[n]} = \frac{\sum_1^n F \times P}{\sum_1^n F} \tag{18}$$

These calculations are done for all the actions in the match set. Finally, if the number of actions in the prediction array is less than the total number of actions, the covering operation is performed.

3.4. XCS with memory

To select actions in the action-set of XCS, this array must be filled in advance. For example, in Fig. 6a, there is no classifier in the population with actions 00 or 10. For this purpose, the XCS fills it using a covering operation. The covering operator creates a new classifier with a condition matching the current input and an action that is chosen randomly from the actions of those classifiers not represented in the match list. Then, the new classifiers are added to the population, and the matching is repeated.

If the input is matched with a classifier of the population while the action-set is not complete, other actions are randomly used by a covering operation, which creates new classifiers for the population. These new classifiers can affect the subsequent entries. For this purpose, as shown in Fig. 6b, a memory unit called BCM (Best Classifier Memory) has been added to XCS. This memory stores the recently faced input condition as well as the best action which maximizes the gained reward.

Algorithm 1 shows the process of selecting the best action corresponding to the input by the BCM learning classifier system. In this algorithm, after receiving the ambient condition from the detector (Line 1), the agent searches for it in the BCM memory using the function $Exist\_Memory$ (line 2). This function searches for a classifier in the memory whose condition matches the input condition. If there exists such a classifier, the value of its index is returned; otherwise, -1 is returned. The main purpose of this search is to find the best matching classifier with the minimum lookup complexity. As a result, after receiving the input condition from the detector, to select the next classifier of XCS with the most appropriate action, a random number is generated with a probability of p. If the condition of the classifier in BCM memory matches the input condition, the agent applies that action to the environment with the probability of p (lines 4–8); otherwise, XCS finds the best matching classifier through its normal process with the probability of $(1 - p)$. The agent updates the BCM classifier after comparing the reward of the classifier in the BCM memory with that of the selected classifier (lines 10–17).

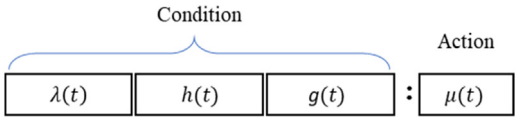

Fig. 7. A typical XCS classifier adapted for workload allocation problem.

Algorithm 1. BCM-XCS

Input: *condition*
Output: *action*

```
current_input ← Environment.state();
index ← Exist_Memory(current_input, Best_Cls);
// classify the input by Best_Cls memory
if random() < n && index > −1 then
    action ← Best_Cls.at(index).action;
    Environment.perform(action);
    flag ← true;
End if
// classify the input by XCS
if flag ≠ true then
    action ← find the best action by the XCS(current_input)
    xcs_Memory newCell;
    newCell.state ← current_input;
    newCell.action ← action;
    newCell.reward ← Environment.reward();
    Add_Update_Memory(newCell, Best_Cls, index);
End if
```

3.5. Application of LCS in the proposed model

The system described in the previous section can contribute to decision-making in the base station concerning the amount of workload at the edge. In modeling the workload of edge systems, the input to the system (conditions) is the network status and the output (actions) is the amount of the workload to be processed at the edge. Fig. 7 represents the input conditions of the system and the action which the classifier generates.

As can be seen in Fig. 7, the environment status is presented by three parameters, i.e., $\lambda(t)$, $h(t)$, and $g(t)$. Also, the battery energy status at the edge system, denoted by b(t), is already known because it was calculated through Eq. (10). Based on these conditions and using Algorithm 1, the optimum workload to be processed at the edge is predicted and learned. This amount of workload ($\mu(t)$) is performed as an action on the environment. The reward of this action is returned as a feedback by the environment. The feedback is the cost that the agent intends to reduce.

By matching the condition against the status of the system $s(t)$ and using the LCS as an agent, the best action is produced and then applied to the environment. After applying the action, the imposed cost $c(t)$ is fed back to the agent. Fig. 8 illustrates this process.

As shown in Fig. 8, this system consists of two parts, i.e. environment and agent. The environment is composed of fog servers, cloud services, devices at the edge of the network, renewable energy sources, and the battery. In the other part, the XCS agent runs on the base station and distributes workloads between the cloud and the fog. The state of the environment at each time is obtained as $s(t) \triangleq (\lambda(t), h(t), g(t), b(t))$, where $\lambda(t)$ represents the amount of workload imported from the network edge devices, $h(t)$ is the amount of energy obtained from the renewable energy sources, and $b(t)$ the status of the battery energy at the edge system. On the other hand, the cost resulted according to Eq. (11) during the action of the agent at the time (t) is fed back to the agent as a reward at the time (t+1). This way, the agent calculates the best input-related action for optimal allocation of the workload.

4. Implementation and evaluation

The proposed methods were implemented in MATLAB R2014 on a system with an 8-core 1.8 GHz CPU and 12 GB RAM. In the following, we first present the initialized values of the variables and then discuss the results. In this scenario, each time interval represents 15 s. The workload in each interval is between 10 and 100 requests per second, which randomly enter the network. The network congestion varies from 10 to 20 ms. The renewable energy input varies according to a normal distribution of N(520 W, 150) [13]. The maximum battery capacity is set to $B = 2$ kWh, with the assumption that $b(0) = 0$. The static power consumption of the base station is $d_{sta} = 300$ W. The maximum number of edge servers is $M = 10$. When active, each of these servers consumes 150 W of electricity. The maximum processing rate of each server is 20 requests per second. The operating cost of each battery unit is $\omega = 0.01$ and the cost coefficient of each unit of the supporting power supply is $\varphi = 0.15$.

This simulation was executed with the basic XCS and the BCM-XCS. Initially, the memory of BCM-XCS was empty. In the following section, the results of the basic XCS are compared with those of the BCM-XCS. We show that, by using a BCM-XCS agent, the workload allocation to cloud and fog is done more intelligently so that the average energy and delay are both reduced considerably. Finally, these two methods are compared with other recently published methods.

Fig. 9 shows the power consumption of the system in processing workloads as distributed by two competitor LCS models (XCS and BCM-XCS). As the figure shows, for XCS, the average power required for processing workloads at the edge is greater than the power provided through renewable sources. Thus, it can be concluded that a combination of the energy of batteries and renewable power sources is required for processing workloads. On the other hand, the energy consumption of the BCM-XCS controlled system is lower than that of the XCS controlled system and is approximately equal to the amount of renewable energy at the edge of the network. We can conclude that, by continuously learning the best classifier, the BCM-XCS distributes the workloads more optimally so that the required power could be supplied by more by green sources than by batteries.

Reducing the energy consumption of the overall system decreases consumption of renewable energies at the network edge. Therefore, the difference between the energy required at the edge of the network and the energy generated by renewable sources can be used to increase the battery charge. Fig. 10 compares the performance of the two proposed methods in charging the battery of the edge system.

The simulation results confirm that, in both proposed methods, the battery begins to be charged after a short time. The highlight of the BCM-XCS is the increase in battery level and its upward slope after the time 1531 which approaches 1400 watts while the battery level in the XCS approaches 800 watts. The reason for the initial fluctuations in the battery level for XCS and BCM-XCS is the emptiness of population and the emptiness of BCM memory in the initial state of XCS and BCM-XCS, respectively. That is, both agents apply random actions without any prediction. As time passes and the population is formed, the BCM memory is filled and the BCM-XCS agent begins to apply the best learned actions. As a result, each step increases the likelihood of more anticipated and appropriate actions, which consequently charges the battery more and more. As a result, this memory improves the performance of the LCS in selecting the optimal load distributer action.

Another criterion in evaluating the efficiency of the method is the average delay required for processing workloads. Fig. 11 shows the average delay of processing workloads (in milliseconds) being distributed by XCS and BCM-XCS agents. Due to the initial emptiness of the population in both methods, the average battery level in the first 1000 intervals has fluctuations that decrease with time. Over the time, the population of the proposed LCS models evolves based on the received reward and the calculated parameters of the classifiers; therefore, the systems make better decisions and perform better actions on the environment.

When the population of BCM-XCS is filled with randomly generated classifiers, the learning process slows down. The processing delay is approximately a fixed value before the time interval 5000. After this time, the average processing delay decreases with a mild slope and

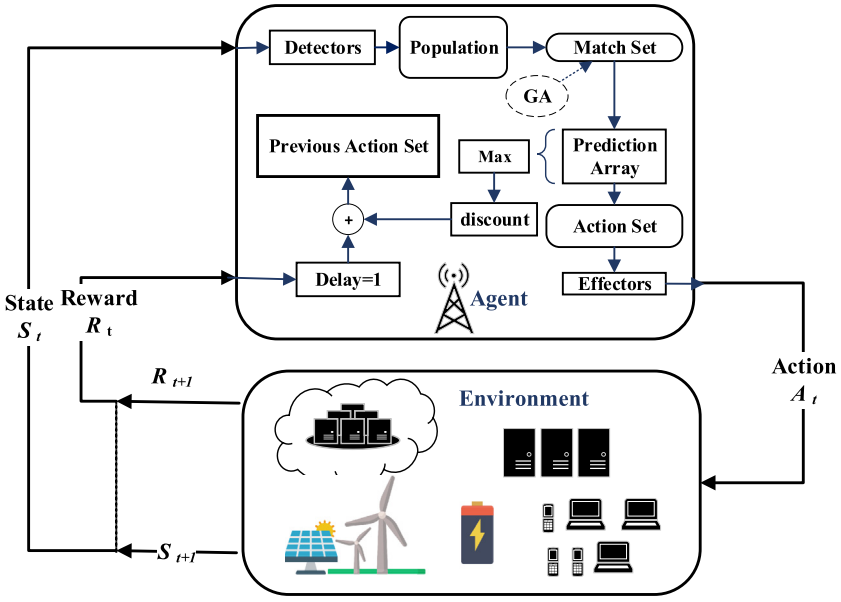

Fig. 8. Adapting XCS as a control agent for optimum workload allocation in the Fog–Cloud environment.

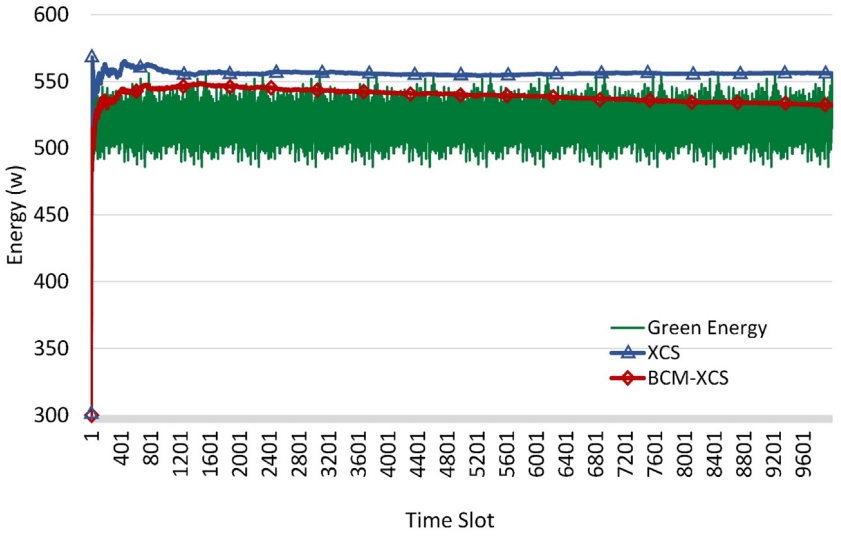

Fig. 9. Average power consumption.

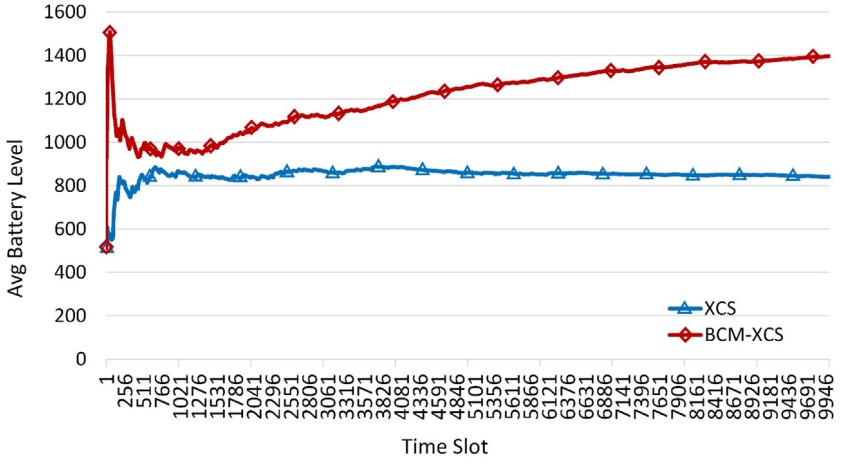

Fig. 10. Average battery level in network edge.

approaches 4.5 s. The BCM-XCS *agent uses the best classifiers stored in* BCM *memory and simultaneously evolves the classifiers to accelerate the prediction process. This accelerated evolution of the BCM-XCS is evident in the corresponding plot*. As the learning continues, the mean processing delay decreases and reaches 3.2 s at the end of the simulation interval.

In the following, we compare BCM-XCS with four recently published methods which are briefly described below.

Fixed Power: In this method, a fixed amount of power is allocated to edge computations at each time interval.

Post Decision State (PDS) algorithm [13]: The PDS algorithm captures the state of the system immediately after making a decision but before the arrival of any new input information. The post-decision state of a decision-making system which is considered as the end-of-period state is called the after-state variable in the reinforcement learning community. The PDS is used extensively in the literature on decision trees since it breaks down the lookup problems into decision nodes and outcome nodes, which respectively represent pre-decision and post-decision states. When dealing with vector-valued decision of workload allocation, the PDS seeks to minimize the long-term costs of the system.

Q-learning [22]: Q-learning is a model-free reinforcement learning algorithm. The algorithm aims to learn a strategy, which tells the agent what is the best action under what circumstances. It does not need a model of the environment and can solve the problems with stochastic transitions and payoffs, without needing adjustments.

Myopic optimization []: This algorithm ignores the relationship between system states and decisions; instead, it minimizes the cost function in the current state using the only current input information of the system. That is, in the Myopic optimization model, the foreknowledge of the workload allocation problem is condensed to a limited number of time instances in a so-called Myopic window, which is shorter than the full timeframe of the problem. Therefore, decisions are repeated during the modeling period.

Fig. 12 compares the battery energy level in the proposed method during 10 000 time intervals with that of the four rival methods. In all methods, the renewable power sources are used in the edge system. The battery energy in the Myopic method is usually insufficient; hence, inevitable use of the supporting power supply imposes high costs on the system. When a low power (e.g., 0.4 kW) is allocated in the Fixed Power method, the battery energy level is usually significantly high (70%–100%). This means that a large part of the renewable energy cannot be used even for charging the batteries. Although with an appropriate Fixed Power policy the system may reach a balance, it cannot conform to the dynamic nature of the problem [13]. An in-depth inspection reveals that the PDS-based algorithm keeps the battery energy level on average around 30%, which is above the power required for processing the workloads. The XCS-based method keeps the battery level around 40% in most of the time, which is higher than the baseline method of PDS.

The battery energy level in BCM-XCS is often more than 60% while in the other methods except Fixed Power (Q-Learning, PDS and Myopic*)* the probability that the battery level exceeds 60% is approximately zero. The higher battery levels achieved by the proposed method shows the system's efforts to distribute workloads optimally and in a way that as much of the required energy as possible could be supplied by renewable energy sources. As a result, the cost of battery depreciation and the total long-term costs of the system are reduced.

Another important achievement of XCS and BCM-XCS is their efficiency in fully charging the battery of the edge system. The XCS-based methods sometimes accidentally decide to send all the workload to the cloud regardless of the learned actions and the received rewards. Especially in BCM-XCS, the balance between applying random actions and using the action of the best classifier lets the renewable energy sources to fully charge the battery. From among the four rival methods, only the Fixed Power method with a power of 0.4 KW can fully charge the battery. But, as compared to the XCS and BCM-XCS, which respectively fully charge the battery in 4% and 25% of the timeline, that competitor in only 3% of the timeline can fully charge the battery.

Another important measure for evaluating the efficiency of the workload allocation methods is the delay in processing the workloads. Fig. 13 compares the mean delay cost of processing workloads in edge-fog-cloud architecture in 10 000 time-intervals. The Fixed Power method and the Myopic method suffer from numerous delays in processing the workloads because of the unpredictability of their allocation policy at the beginning of the simulation. However, in PDS, Q-Learning and the proposed LCS-based methods, the resulting delays are relatively shorter. The mean delay cost of XCS and BCM-XCS is 4.5 and 3, respectively, which is considerably lower than that of any other method. Also, the mean delay of BCM-XCS and XCS converges to the above-mentioned values of mean delay with the minimum fluctuations. The reason behind the negligible fluctuations in the delay in processing workloads by the proposed XCS-based methods is the effect of environment rewards on the fitness of the matched classifiers, which enables the agent to control any sudden increase in each of the three components of the mean delay.

As explained in Section 3.1, the mean delay is composed of three components, namely, delay in wireless communication, delay in the local processing of workloads at network edge, and delay caused by offloading the workloads to the cloud. The wireless communication delay is the least important one. The delay caused by the base station of the network edge for deciding on the amount of workload allocated to each edge device and transmitting the rest to the cloud is the second important component of the mean delay cost. The amount of this delay can be predicted by estimating the computational burden of the workload allocator algorithm. The offloading delay is the most significant delay component that can considerably affect the mean delay.

According to Fig. 13, BCM-XCS has the lowest mean delay cost. The mean delay of this method does not exceed 6 ms. The reason is that, as the number of time intervals increases, not only does the LCS update the population of classifiers according to the reward, it also stores the best classifier at each time interval in the BCM cache. As a result, the system can apply optimal actions in the next time intervals.

5. Conclusion

Transmission of large volumes of data to the cloud causes long delays in the processing of workloads. Alternatively, parts of the data can be processed at the network edge to reduce the delays. However, this leads to power deficit at the edge. This paper proposed a method to optimize the partial distribution of workloads and create balance between delays and power consumption at the edge.

In the proposed method, an LCS called XCS was used. XCS tries to find optimal classifiers by interacting with the environment and receiving rewards as well as using evolutionary algorithms such as the genetic algorithm. In this paper, the optimal form of workload distribution was computed for the first time by adapting the parameters including total workload and the amount of green energy as the input of the system. Also, in spite of the increasing number of additional random classifier rules in the covering process, an additional memory was utilized in the XCS system to accelerate the learning process. The most important innovation of this paper is presenting a new LCS which was called BCM-XCS. In this new XCS-based classifier, the application of obscure random actions is reduced by storing the sequence of the input conditions of the system along with the best action for each state in the secondary memory. One of the major results of this new operation is considerable increase in the accuracy of the LCS.

The results show that delay in processing workloads and battery levels are considerably better in XCS and BCM-XCS than in any recent method. In contrast to the existing methods that cannot charge the battery to more than 40%, the proposed methods can fully charge the battery.

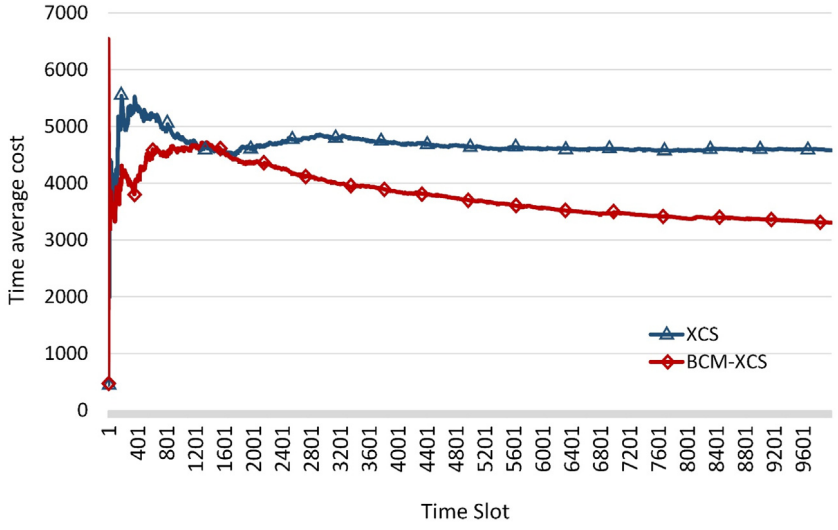

Fig. 11. Mean delay of processing workloads distributed by different methods.

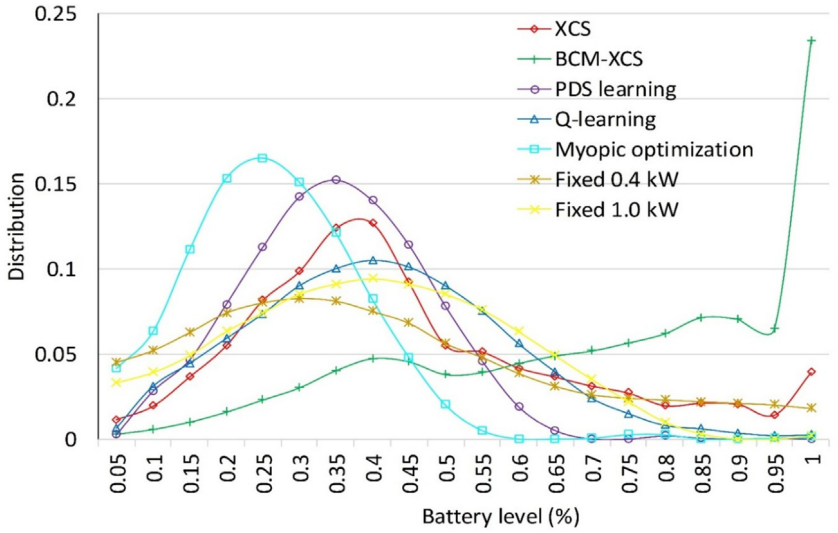

Fig. 12. The distribution of battery energy level over time.

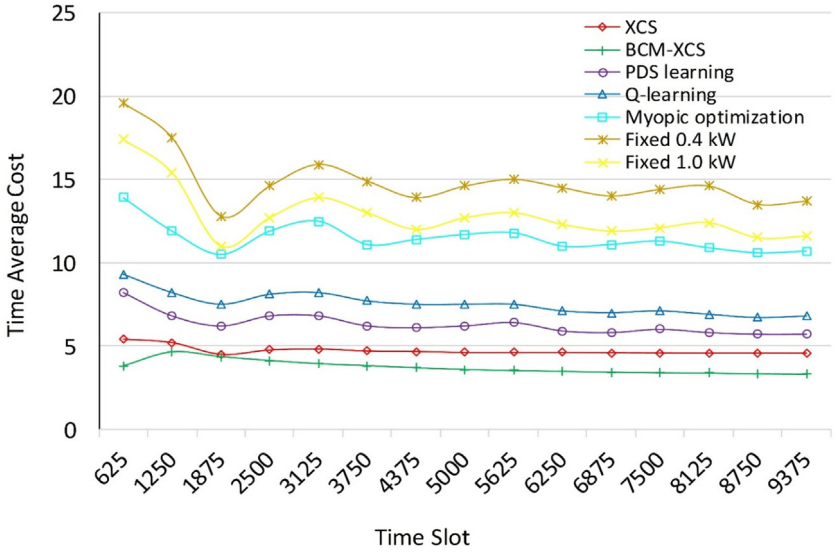

Fig. 13. The mean cost of time (delay).

The mean delay cost of the XCS and BCM-XCS begins to decrease after 1000 time slots and approaches 5 ms and 3 ms, respectively, which is lower than that of any rival method. The proposed XCS-based methods can be used for workload allocation in networks that have limited access or no access to the electrical grids. In these systems, solar or wind energies or any other renewable energy sources can be utilized as the main, or even the sole, power supply to reduce the operating costs of the batteries. In further studies, we may make use of machine learning methods such as neural networks.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Mahdi Abbasi: Supervision, Conceptualization, Methodology, Formal analysis, Writing - review & editing. **Mina Yaghoobikia:** Data curation, Software, Writing - original draft. **Milad Rafiee:** Visualization, Investigation. **Alireza Jolfaei:** Methodology, Validation, Writing - review & editing. **Mohammad R. Khosravi:** Conceptualization, Validation, Writing - review & editing.

References

[1] A.H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, Q.Z. Sheng, IoT middleware: A survey on issues and enabling technologies, IEEE Internet Things J. 4 (2016) 1–20.
[2] E. Niewiadomska-Szynkiewicz, A. Sikora, J. Kołodziej, P. Szynkiewicz, Modelling and simulation of secure energy aware fog sensing systems, Simul. Model. Pract. Theory (2019) 102011.
[3] M. Chiang, T. Zhang, Fog and IoT: An overview of research opportunities, IEEE Internet Things J. 3 (2016) 854–864.
[4] F. Firouzi, B. Farahani, Architecting IoT cloud, in: Intelligent Internet of Things, Springer, 2020, pp. 173–241.
[5] A. Pravin, P. Jacob, G. Nagarajan, A comprehensive survey on edge computing for the IoT, in: Edge Computing and Computational Intelligence Paradigms for the IoT, IGI Global, 2019, pp. 37–45.
[6] H. Wu, L. Chen, C. Shen, W. Wen, J. Xu, Online geographical load balancing for energy-harvesting mobile edge computing, in: 2018 IEEE International Conference on Communications, ICC, 2018, pp. 1–6.
[7] J. Xu, S. Ren, Online learning for offloading and autoscaling in renewable-powered mobile edge computing, in: 2016 IEEE Global Communications Conference, GLOBECOM, 2016, pp. 1–6.
[8] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, C. Liu, Fog computing for energy-aware load balancing and scheduling in smart factory, IEEE Trans. Ind. Inf. 14 (2018) 4548–4556.
[9] D. Zeng, L. Gu, H. Yao, Towards energy efficient service composition in green energy powered Cyber–Physical Fog Systems, Future Gener. Comput. Syst. (2018).
[10] B. Bartin, Use of learning classifier systems in microscopic toll plaza simulation models, IET Intell. Transp. Syst. 13 (2019) 860–869.
[11] A. Kumar, A. Bansal, Software fault proneness prediction using genetic based machine learning techniques, in: 2019 4th International Conference on Internet of Things: Smart Innovation and Usages, IoT-SIU, 2019, pp. 1–5.
[12] A. Stein, Learning classifier systems: from principles to modern systems, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 747–769.
[13] J. Xu, L. Chen, S. Ren, Online learning for offloading and autoscaling in energy harvesting mobile edge computing, IEEE Trans. Cogn. Commun. Netw. 3 (2017) 361–373.
[14] Y. Liu, C. Yang, L. Jiang, S. Xie, Y. Zhang, Intelligent edge computing for IoT-based energy management in smart cities, IEEE Netw. 33 (2019) 111–117.
[15] K. Muhammad, J. Lloret, S.W. Baik, Intelligent and energy-efficient data prioritization in green smart cities: Current challenges and future directions, IEEE Commun. Mag. 57 (2019) 60–65.
[16] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[17] P.L. Lanzi, Learning Classifier Systems: From Foundations To Applications, Springer Science & Business Media, 2000.
[18] O. Sigaud, S.W. Wilson, Learning classifier systems: a survey, Soft Comput. 11 (2007) 1065–1078.
[19] S.W. Wilson, G. Xcs, Generalization in XCS, 1996.
[20] M.V. Butz, S.W. Wilson, An algorithmic description of XCS, in: International Workshop on Learning Classifier Systems, 2000, pp. 253–272.
[21] D. Pätzel, A. Stein, J. Hähner, A survey of formal theoretical advances regarding XCS, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 1295–1302.
[22] R.S. Sutton, A.G. Barto, Introduction to Reinforcement Learning, Vol. 2, MIT Press Cambridge, 1998.