JACET

J. ADV COMP ENG TECHNOL, 7(1) Winter 2021

Intelligent Resource Allocation in Fog Computing: A Learning Automata Approach

Alireza Enami[1] , Javad Akbari Torkestani[2]
1- Department of Computer Engineering, Islamic Azad University Arak Branch, Arak, Iran.
2- Department of Computer Engineering, Islamic Azad University Arak Branch, Arak, Iran. (j-akbari@iau-arak.ac.ir)

Received (2020-09-28) Accepted (2021-03-23)

Abstract: *Fog computing is being seen as a bridge between smart IoT devices and large scale cloud computing. It is possible to develop cloud computing services to network edge devices using Fog computing. As one of the most important services of the system, the resource allocation should always be available to achieve the goals of Fog computing. Resource allocation is the process of distributing limited available resources among applications based on predefined rules. Because the problems raised in the resource management system are NP-hard, and due to the complexity of resource allocation, heuristic algorithms are promising methods for solving the resource allocation problem. In this paper, an algorithm is proposed based on learning automata to solve this problem, which uses two learning automata: a learning automata is related to applications (LAAPP) and the other is related to Fog nodes (LAN). In this method, an application is selected from the action set of LAAPP and then, a Fog node is selected from the action set of LAN. If the requirements of deadline, response time and resources are met, then the resource will be allocated to the application. The efficiency of the proposed algorithm is evaluated through conducting several simulation experiments under different Fog configurations. The obtained results are compared with several existing methods in terms of the makespan, average response time, load balancing and throughput.*

Keywords: *Fog Computing, Heuristic Algorithms, Learning Automata, Resource Allocation*

How to cite this article:
Alireza Enami , and Javad Akbari Torkestani. Intelligent Resource Allocation in Fog Computing: A Learning Automata Approach. J. ADV COMP ENG TECHNOL, 7(1) Winter 2021 : 18-34

I. INTRODUCTION

Fog computing is a distributed computing that acts as an intermediate layer between cloud servers and Internet of Things (IoT) devices/sensors. Similar to cloud servers, Fog computing provides processing, networking, and storage, but closer to IoT devices/sensors to reduce latency, network traffic, power consumption, and operating costs [1]. Fog computing has both edge and network computations. The development of multi-layered applications and the migration of services from a number of IoT devices/ sensors can be easily done through the Fog. Also, the edge network components can be closer to IoT devices/sensors than cloud and edge servers, thus reducing service delays for real-time applications [2]. Fig. 1 shows a 3-layer architecture for Fog computing.

The features of Fog computing include [4],[5]: Awareness of location, mobility support, real-time interactions, scalability, interoperability, reduce service latency, reduce energy consumption, reduce network traffic, reduce capital and operational expenses, reduce content distribution, reduce network latency, wide geographical distribution, huge number of nodes, wireless access, real-time analysis and heterogeneity of software and hardware resources. Due to the above features, some applications are assumed for Fog computing in the fields of health, medicine, agriculture,

This work is licensed under the Creative Commons Attribution 4.0 International Licence.
To view a copy o this licence, visit https://creativecommons.org/licenses/by/4.0/

data centers, energy, industry, military, smart homes, smart cities, transportation, network of cars, online games and video transmission [4],[5].

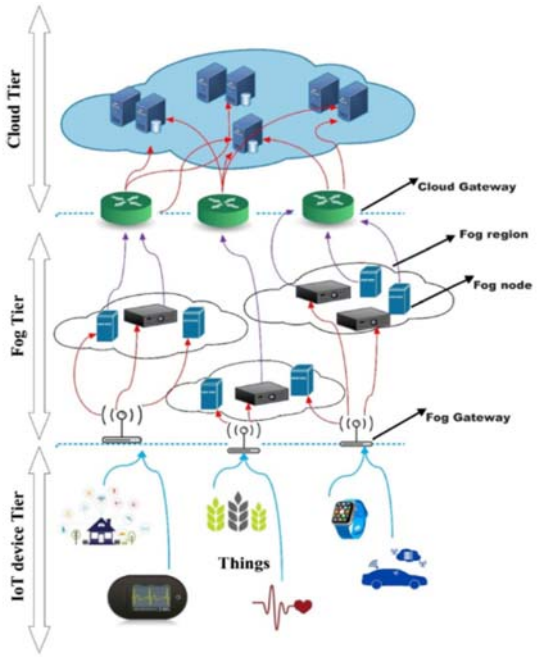

Fig.1. Three-layer architecture of Fog computing [3].

Resource allocation [6]-[8] is one of the most important services of the system, which should always be available to achieve the goals of Fog computing. A common problem with Fog computing is choosing the best resource for running specific applications. Resource allocation is the process of distributing limited available resources among applications based on predefined rules. Resource allocation mechanisms play an essential role in the process of scheduling of Fog computing, and the efficiency of these mechanisms determines the quality of service [3]. There are several interesting features that make resource allocation much more challenging. Some of these features include [9]: scalability, adaptability, error tolerance and reliability, load balancing, dynamic structure, high heterogeneity of resources and applications.

Considering the challenging features of Fog computing, the heuristic solutions are undoubtedly the best way to solve the resource allocation problem in the Fog computing. The features of these solutions include [10],[11].

- Heuristic solutions are well-understood.
- There is no need for optimal solutions.
- Effective heuristic in a short time.
- Dealing with multi-objective nature.
- Appropriateness for decentralized solutions
- Ability to combine with other practices.
- Designing robust schedulers.
- Libraries and frameworks for meta-heuristic.
- The learning automata and its hybrid models can be considered as a suitable model for solving the above problem because of the following features [12]-[24]:
- The learning automata are able to perfectly adapt themselves to environmental changes. This feature is very suitable for use in Fog environments with a high degree of dynamism.
- In addition to very low computational requirements, the learning automata impose a small amount of communication costs in interacting with the environment. This feature distinguishes learning automata as a suitable alternative for use in environments with energy constraints and bandwidth than the other models.
- Interacting with each other, the learning automata are able to perfectly model the distribution of Fog environments and in addition, simulate the changing behavioral patterns of the nodes in relation to each other and with the environment considering their learning ability and their adaptability to the environment.
- Interacting with each other, the learning automata are able to converge to the global optimal answer only based on the local decisions when solving optimization problems. Therefore, learning automata-based algorithms can be considered as an appropriate choice for the Fog as they can resolve the slag resulted from aggregation or dissemination of information in centralized algorithms.
- The learning automata complete their information required for decision-making in an iterable process and over time, from the environment in which they are located. Accordingly, in case of the occurrence of possible errors, the tolerance of learning automata-based algorithms will not affect

the algorithm's performance compared to the other algorithms.

In this paper, a learning automata-based algorithm is proposed to solve resource allocation problem in Fog computing (RALA). This method uses two learning automata: a learning automata is related to applications (LAAPP) and the other is related to Fog nodes (LAN). In the proposed algorithm, an application is selected from the action set of LAAPP and then a Fog node is selected from the action set of LAN. If the following 3 conditions are met, then the resource will be allocated to the application:

- The selected application has the shortest deadline.
- The response time of the selected application is less than its deadline.
- The selected Fog node has the ability to provide the resources of the selected application.

Several simulation experiments are conducted under several Fog configurations to show the performance of the proposed resource allocation algorithm. The results of the proposed algorithm are compared with those of BLA [25], ACO [26], GABVMP [27], and Random. Simulation results show that the proposed algorithm outperforms the other methods in terms of makespan, average response time, load balancing and throughput.

The rest of the paper is organized as follows. Literature is reviewed in the section II. In Section III, the definition of learning automata and learning automata with variable action set is described. In Section IV, the resource allocation algorithm based on learning automata is proposed. In Section V, the results of the proposed algorithm are presented, and finally, conclusion is provided in section VI.

II. RELATED WORKS

As a modern and comprehensive computing model, Fog computing is expanding due to computations at the edge level. Resource allocation is the process of distributing limited available resources among applications based on predefined rules. In this section, a set of related mechanisms and algorithms are introduced.

A method based on the life of bees for the job scheduling problem in the Fog computing environment was presented in the paper [25]. The proposed method was based on marriage (reproduction) and search for a food source. Two evaluation criteria were considered: the execution time of the processor and the total amount of memory required for all jobs accepted for execution (dedicated memory). The results of the proposed algorithm had been compared with genetic algorithm and particle swarm optimization, and the simulation results showed an improvement in the proposed algorithm over other algorithms. Though proving a great performance in solving such large scale problem, it did not consider some special characteristics of Fog computing paradigm, e.g., the tradeoff problem whether send the tasks to the cloud or not.

Ghaffari [26] first studied Fog computing and scheduling, and then proposed an algorithm based on the ant colony to assign tasks to virtual machines with minimal time and cost. The proposed algorithm consists of 3 steps. In the first stage, input tasks were categorized based on end time and cost. In the second step, categorized tasks were prioritized in terms of time and cost. In the third step, the ant colony algorithm was implemented to assign tasks to virtual machines. The proposed algorithm was evaluated in terms of end time, delay, load balancing and energy. The ant colony usually takes longer time to search and not suitable for large-scale problems.

The issue of service quality in cloud/Fog computing environments had been investigated in [27] by providing two models. In this regard, first the issue of assigning tasks to virtual machines was formulated as a linear programming model, and the HABBP algorithm was presented for load balancing policy to assign cloudlets to virtual machines. Next, the problem of virtual machine placement was solved using the genetic algorithm (GABVMP). The simulation results showed, this algorithm outperforms the Random Placement and First Fit algorithms in term of the allocation cost parameter. The genetic algorithm has a slow convergence rate and it may converges to a local optima.

Guangshun Li et al. [28] first standardized and normalized the resource attributes and features. Next, fuzzy clustering and particle swarm optimization methods had been combined in

their paper to categorize resources to reduce the resource search scale. Finally, a resource scheduling algorithm based on fuzzy clustering was presented. The simulation results showed that the proposed algorithm had a higher convergence velocity than the conventional fuzzy algorithm. In this paper, the proposed algorithm is compared with the algorithm based on the Grid structure. Also, user satisfaction is the only parameter examined in this article.

In the paper [29], a heuristic algorithm for task scheduling in Fog computing was presented based on ant colony optimization and particle swarm optimization. This algorithm solved the problem of scheduling end devices with limited computational resources and high energy consumption, so that it was suitable for real-time tasks and efficient processing. The proposed algorithm had been compared with ACO, PSO, Round Robin algorithms and provided good results in terms of reliability, energy consumption and completion time. This algorithm is static and the load balancing parameter is ignored.

A multi-objective simplified swarm optimization (MOSSO) method for solving the scheduling problem in Fog computing was presented in the paper [30]. MOSSO was a multi-objective optimization method based on Simplified Swarm Optimization (SSO). SSO was a population-based stochastic optimization method characterized by its simplicity and efficiency. The objectives of the MOSSO method were to reduce the processing rate as well as the cost, which provided better results compared to the multi-objective particle swarm optimization (MOPSO) algorithm. In this article, no attention has been paid to throughput and load balancing.

Huang et al. [31] provided a blockchain-based model for resource sharing in Fog nodes. The proposed model actively used the blockchain reward and penalty mechanism to share resources. The Fog node behavior in resource sharing and the degree of completion of the task in resource sharing were packaged into blocks and stored in the blockchain system to meet the transparency feature. In the following, a differentiated game method had been used to build a resource sharing model and simulate the optimal resource sharing strategy. One of the advantages of blockchain technology in Fog computing architecture is the creation of an appropriate level of security in an unsafe environment. The results of this algorithm have not been compared with other algorithms.

A new model of task scheduling according to the role of containers was presented in the paper [32]. This model was proposed to minimize task completion time and maximize the number of concurrent tasks for Fog node. The task execution processing was divided into two sub-steps: determining the tasks that were accepted or rejected, and scheduling the tasks accepted in the cloud or in the Fog. The proposed algorithm had been compared with FT-FQ, FT-RE, DT-FQ and DT-RE methods. An important feature of this algorithm was the reduction of tasks delay. In general, this algorithm provides a solution for scheduling workflows, taking into account the QOS factors requested by the user.

An optimization framework for Fog nodes (FNs), data service operators (DSOs), and data service subscribers (DSSs) for IoT Fog computing was presented in the paper [33]. In this framework, first, the Stackelberg game had been proposed to solve the pricing problem of DSOs and FNs, and then a many-to-many matching taked place between FNs and DSSs. The simulation results showed that all FNs, DSOs and DSSs could operate close to their optimal and the proposed framework was highly efficient compared to the FNs-free mode. The results of this algorithm have not been compared with the exact or heuristic algorithms.

Josilo and Dan [34] had proposed a theoretical model for the task allocation problem. In this paper, Variational Inequality Theory was used to compute an equilibrium task allocation in static strategies. Based on this strategy, a decentralized algorithm had been proposed for allocating the computational tasks among nearby devices and the edge cloud. The efficiency of the proposed algorithm had been compared with an optimal algorithm that uses global knowledge of system status. The results showed a good level of efficiency for the proposed algorithm. In this article, no attention has been paid to the completion time.

An application scheduling technique based on virtualization technology to find an effective and efficient algorithm was presented in the paper [35]. The algorithm could reduce energy consumption and the average delay of real-time applications in Fog computing networks. Four task scheduling policies had been reviewed in a

Fog node scheduler to examine their effectiveness. The four algorithms were: First-Come-First-Service (FCFS), Shortest-Job-First (SJF), Round-Robin (RR) and Generalized-Priority (GP). The simulation results showed that the FCFS algorithm has 11%, 7.78%, 4.4%, and 15.1% improvement in terms of energy consumption, average task delay, network usage and execution time, respectively compared to other algorithms. In this paper, heuristic algorithms are not examined.

A method for assigning a dynamic resource (DRAM) was presented in the paper [36] to load balancing in the Fog environment. Initially, a system framework for Fog computing was presented, and the load balancing for computational nodes was analyzed. Then, the DRAM method was implemented based on the allocation of static resource and dynamic scheduling for Fog services. The DRAM algorithm had four steps: Step 1: Fog service partition, Step 2: spare space detection for computing nodes, Step 3: static resource allocation for Fog service subset and Step 4: load-balance driven global resource allocation. The simulation results showed that the proposed algorithm outperforms other algorithms in terms of average resource utilization and average load balancing variance. In this algorithm, the end time of each task is not considered.

III. LEARNING AUTOMATA THEORY

A learning automaton [37],[38] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. The action is chosen at random based on a probability distribution kept over the action set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton (LA) is to find the optimal action from the action set so that the average penalty received from the environment is minimized. LA have been found to be useful in systems where incomplete information about the environment exists. LA are also proved to perform well in complex, dynamic and random environments with a large amount of uncertainties.

The environment can be described by a triple $E=\{\alpha,\beta,c\}$, where $\alpha=\{\alpha_1,\alpha_2,\ldots,\alpha_r\}$ represents the finite set of the inputs, $\beta=\{\beta_1,\beta_2,\ldots,\beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c=\{c_1,c_2,\ldots,c_r\}$ denotes the set of the penalty probabilities, where the element c_i is associated with the given action α_i. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non-stationary environment. The environments depending on the nature of the reinforcement signal β can be classifiedinto P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of the environment allows a finite number of the values in the interval [0, 1] can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval [a,b].

LA can be classified into two main families [37],[38]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $<\beta,\alpha,L>$, where β is the set of inputs, α is the set of actions, and L is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha_i(k) \in \alpha$ and p(k) denote the action selected by learning automaton and the probability vector defined over the action set at instant k, respectively. Let a and b denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. Let r be the number of actions that can be taken by learning automaton. At each instant k, the action probability vector p(k) is updated by the linear learning algorithm given in Eq. 1, if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in Eq. 2 if the taken action is penalized.

$p_j(k+1)=p_j(k)+a[1-p_j(k)]$;for j=i
$p_j(k+1)=(1-a)p_j(k)$; otherwise (1)
$p_j(k+1)=(1-b)\ p\_j\ (k)$;for j=i
$p_j(k+1)=b/(r-1)+(1-b)p_j(k)$; otherwise(2)

If a = b, the recurrence Eq. 1 and Eq. 2 are called linear reward-penalty (L_{R-P}) algorithm, if a>>b the given equations are called linear reward-ϵpenalty ($L_{R-\epsilon P}$), and finally if b = 0 they are called linear reward-Inaction (L_{R-I}). In L_{R-I}, the action probability vectors remain unchanged when the taken action is penalized by the environment.

1. Variable Action-Set Learning Automata

A variable action set learning automaton (VLA) is an automaton in which the number of actions available at each instant changes with time. It has been shown in [37] that a learning automaton with a changing number of actions is absolutely expedient and also ϵ-optimal, when the reinforcement scheme is L_{R-I}. Such an automaton has a finite set of n actions, $\alpha=\{\alpha_1,\alpha_2,\ldots,\alpha_r\}$. $A=\{A_1,A_2,\ldots,A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k. The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $\Psi(k)=\{\Psi_1(k),\Psi_2(k),\ldots,\Psi_m(k)\}$ defined over the possible subsets of the actions, where $\Psi_i(k)=prob[A(k)=A_i|Ai \in A,i=2^n-1]$. Let $\hat{p}_i(k)=prob[\alpha(k)=\alpha_i|A(k),\alpha_i \in A(k)]$ denotes the probability of choosing action α_i, conditionedon the event that the action subset A(k) has already been selected and $\alpha_i \in A(k)$ too. The scaled probability $\hat{p}_i(k)$ isdefined as

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \qquad (3)$$

where $K(k)=\Sigma_{\alpha i \in A(k)} p_i(k)$ is the sum of the probabilities of the actions in subset A(k), and $p_i(k)=prob[\alpha(k)=\alpha_i]$.

The procedure of choosing an action and updating the action probabilities in a VLA can be described as follows. Let A(k) be the action subset selected at instant n. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in Eq. 3. The automaton then randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}_i(k)$. Depending on the response received from the environment, the learning automaton updates its scaled action probability vector. Note that the probability of the available actions is only updated. Finally, the probability vector of the actions of the chosen subset is rescaled as $p_i(k+1) = \hat{p}_i(k+1).K(k)$, for all $\alpha_i \in A(k)$. The absolute expediency and ε-optimality of the method described above have been proved in [37].

IV. RESOURCE ALLOCATION ALGORITHM

Resource allocation is one of the most important services of the system, which should always be available to achieve the goals of Fog computing. Resource allocation is the process of distributing limited available resources among applications based on predefined rules. According to the reasons of resource allocation complexity, as well as the characteristics of the heuristic solutions, we have proposed an algorithm based on variable action set learning automata to solve this problem.

1- Problem Formulation

Suppose $A=\{A_1,A_2,...,A_m\}$ is a set of applications, so that m is the total number of applications. Each application A_i consists of a number of independent tasks that can be run simultaneously: $A_i = \left\{t_i^1,t_i^2,\cdots,t_i^{k_i}\right\}$. Each task in the application is limited to a deadline; in other words, $D_{A_i}^{t^l}$ indicates the deadline for the task t^1 of the application A_i. Also, each of the tasks t^l has a response time feature:

$$RT_{A_i}^{t^l} = Predicted_{execution_{time(t_i^l)}} + Delay\_time(t_i^l) \quad (4)$$

so that $RT_{A_i}^{t^l} \leq D_{A_i}^{t^l}$. Each application also has deadline and response time features that are defined as follows:

$$MAX\left(R_{TA_i}^1, RT_{A_i}^2, \cdots, RT_{A_i}^{k_i}\right) = RT_{A_i} \quad (5)$$

$$MIN\left(D_{A_i}^1, D_{A_i}^2, \cdots, D_{A_i}^{k_i}\right) = D_{A_i} \quad (6)$$

so that $RT_{A_i} \leq D_{A_i}$.

Suppose $N=\{N_1,N_2,...,N_n\}$ is a set of Fog nodes and $R=\{R_1,R_2,...,R_c\}$ is the different types of resources available to the Fog network. Each of the Fog nodes has its own resources: $N_i = \{R_j^1, R_j^2, \cdots, R_j^c\}$. The Eq. 7 determines how much each application needs from each resource, so that $R_{t_i^l}^{r^p}$ indicates how many task t^l from application A_i needs from resource p.

$$R_{A_i}^{r^p} = \sum_{\substack{\forall t^l \in A_i \\ 1 \leq p \leq c}} R_{t_i^l}^{r^p} \quad (7)$$

In this method, two learning automata are used to select applications and Fog nodes: $LA_{APP}=\{A_1,A_2,...,A_m\}$ with the action set of the applications so that m is the total number of applications and $LA_N=\{N_1,N_2,...,N_n\}$ with the action set of the Fog nodes so that n is the total number of Fog nodes.

2. The proposed algorithm

The proposed learning automata-based resource allocation (RALA) uses two learning automata: a learning automata is related to applications (LA_{APP}) and the other is related to Fog nodes (LA_N). The Eq.1 and Eq. 2 are used for reward and penalty, so that a and b are reward and penalty coefficients, respectively, and r is also the number of actions (applications in LA_{APP} and Fog nodes in LA_N) in these equations.

The proposed algorithm has the following steps:

Step1: Assigning probability to the LA_{APP} action set.

Step2: Selecting an application according to the LA_{APP} probability vector.

Step3: Assigning probability to the LA_N action set.

Step4: Selecting a Fog node according to the LA_N probability vector.

Step5: Allocating resources to the application and running it.

Step6: Releasing the allocated resources.

The pseudo-code of the proposed algorithm is shown in Fig. 2. In this algorithm, the probability is firstly assigned to the LA_{APP} action set which is a set of applications provided for execution. At this step, all actions (applications) have an equal probability. An application is then selected based on the probability vector of LA_{APP}. If the selected application has the shortest deadline among the action set, then it will be rewarded according to Eq. 1 and the algorithm will enter the next step; otherwise, the selected application will be penalized according to Eq. 2 and the algorithm will be executed again from the application selection step.

After selecting the application, the probability must be assigned to the LA_N action set. At this step, all actions (Fog nodes) have an equal probability. At this step, a Fog node is selected based on the LA_N probability vector, and the response time of the application must be set. The response time is equal to the sum of predicted execution time and the delay time of the application. If the selected Fog node can meet the requirements of the resource and deadline of the selected application, then it will be rewarded according to Eq. 1 and

the algorithm will enter the next step, otherwise the selected Fog node will be penalized according to Eq. 2 and the algorithm will run again from the selection step on Fog node.

Resource Allocation Based on Learning Automata(RALA)
ALGORITHM RALA
BEGIN
WHILE A set not empty **DO**
SET LA_{APP} action set;
FOR all applications in LA_{APP}
A_i= Choose one of the applications according to probability vector LA_{APP} ;
IF $\left(D_{A_i} \leq D_{A_j}\right)$ **THEN** // $\forall A_j \in A ; j \neq i$
Rewarded the A_i in LA_{APP};
Penalized the A_j in LA_{APP}; // $\forall A_j \in A ; j \neq i$
BREAK;
ELSE
Penalized the A_i in LA_{APP};
Rewarded the A_j in LA_{APP}; // $\forall A_j \in A ; j \neq i$
END IF;

SET LA_N action set;
FOR all Fog nodes in LA_N;
N_i= Choose one of the Fog nodes according to probability vector LA_N;
SET RT_{A_i};
IF $\left(R_{A_i}^{P} \leq N_i\right)$ AND$\left(RT_{A_i} \leq D_{A_i}\right)$ **THEN**
Rewarded the N_i in LA_N;
Penalized the N_j in LA_N; // $\forall N_j \in N ; j \neq i$
BREAK;
ELSE
Penalized the the N_i in LA_N;
Rewarded the the N_j in LA_N; // $\forall N_j \in N ; j \neq i$
END IF;

IF all Fog nodes in LA_N are penalized **THEN**
Transferred A_i to cloud;
Execute algorithm from **BEGIN**;
Assign recourses of N_i to A_i;
IF execution of the A_i is completed by N_i **THEN**
Released recourses of A_i to N_i;
Rewarded the N_i;

END WHILE;
END.

Fig.2. The pseudo-code of the proposed algorithm

If all the Fog nodes are penalized at this step, then the selected application will be transferred to the cloud, and the implementation of the algorithm will be executed from the application selection step. At this step, the resources of selected Fog node should be allocated to the selected application. In other words, resources must be removed from the list of free resources on Fog node and allocated to the application. After the execution of the application is completed by Fog node, then the application resources will be released and will be added to the list of free resources of Fog node, and Fog node will be rewarded according to Eq. 1. The proposed flowchart algorithm is shown in Fig. 3.

V. RESULTS

In order to evaluate the proposed algorithm, we simulated a Fog environment that included 3 configurations (small scale, medium scale, and large scale): A small scale Fog environment includes 16 Fog nodes, 50 Fog devices, 1000 applications, 128 processors, 32 memories and 32 disks, a medium scale Fog environment including 32 Fog nodes, 100 Fog devices, 2000 applications , 256 processors, 64 memories and 64 disks, and finally a large scale Fog environment including 64 Fog nodes, 400 Fog devices, 8000 applications, 1024 processors, 256 memories and 256 disks.

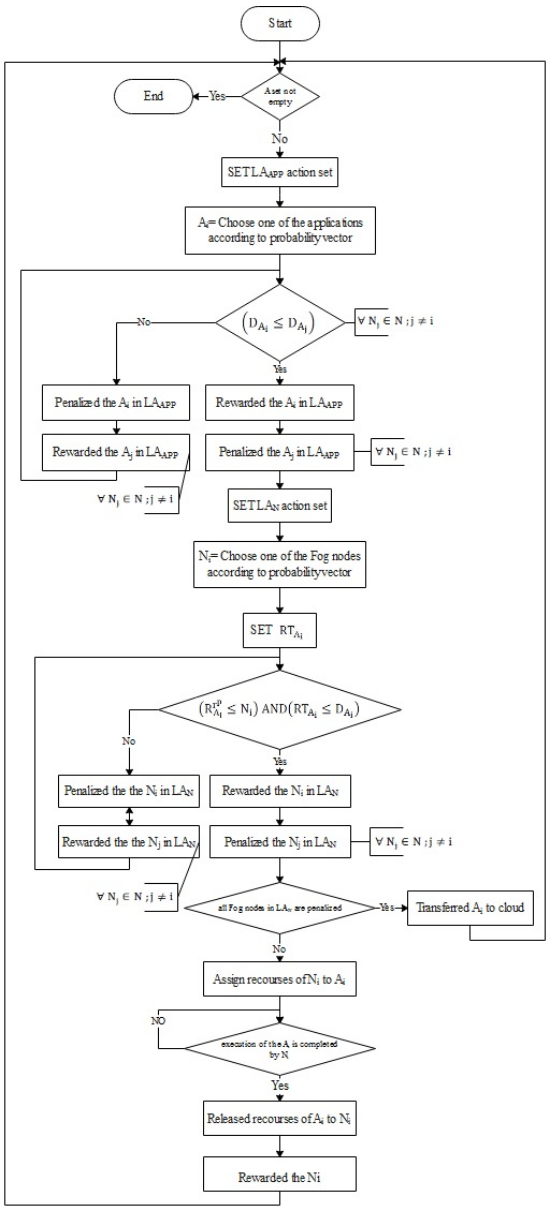

Fig. 3. The flowchart of the proposed algorithm

It is also assumed in all configurations that the system has one data center. Each application consists of a number of tasks. Each application is divided into k tasks so that k is randomly selected from U[1,2,3,4]. In each Fog device, the new application generation rate follows a Poisson distribution with an average rate [5,10,15,20]. Deadline of applications is achieved by a Normal distribution with an average of 500 and a variance of 100. Application execution time is obtained by a Normal distribution with an average of 500 and a variance of 100. Processor computational capacity is obtained by a Normal distribution with an average of 1000 and a variance 150. The memory needed for the application is obtained by a Normal distribution with an average of 256 and a variance of 64. The memory storage capacity by applications is obtained by a Normal distribution with an average of 2000 and a variance of 100. Disk requirement of application is obtained by a Normal distribution with an average of 256 and a variance of 64. The disk storage capacity by applications is obtained by a Normal distribution with an average of 2000 and a variance of 100. The Nominal bandwidth of the network is 100Mbps. To improve the accuracy of the report's results, each test was repeated independently 50 times and the average results were presented. The parameters of this simulation are summarized in Table 1.

We performed all experiments on a desktop PC with an Intel Pentium Core 2 Duo CPU T6600, a clock rate of 2.20 GHz, 4BG of memory and Windows 7 (64-bit).To demonstrate the efficiency of the proposed algorithm (RALA), the obtained results were compared with BLA [25], ACO [26], GABVMP [27] and Random algorithms in terms of makespan, average response time, load balancing, and throughput.

The algorithms are simulated on iFogSim [39]. iFogSim simulation toolkit is developed upon the fundamental framework of CloudSim. CloudSim is one the wildly adopted simulators to model cloud computing environments. Extending the abstraction of basic CloudSim classes, iFogSim offers scopes to simulate customized Fog computing environment with large number of Fog nodes and IoT devices (e.g. sensors, actuators). However, in iFogSim the classes are annotated in such a way that users, having no prior knowledge of CloudSim, can easily define the infrastructure, service placement and resource allocation policies for Fog computing. iFogSim applies Sense-Process-Actuate and distributed dataflow model while simulating any application scenario in Fog computing environment.

Table 1. Simulation Parameters

Parameter	Config #1 Small Scale	Config #2 Medium Scale	Config #3 Large Scale
Number of Fog nodes	16	32	64
Number of Fog Devices	50	100	400
Total number of applications	1000	2000	8000
Number of processors	128	256	1024
Number of memories	32	64	256
Number of disks	32	64	256
Number of data center	1		
Number of tasks per application	Uniform distribution [1, 2, 3, 4]		
Application generation rate	Poisson distribution [5, 10, 15, 20]		
Deadline of application	Normal distribution (500,100) ms		
Execution time of application	Normal distribution (500,100) MI		
Processor computational capacity	Normal distribution (1000, 150) MIPS		
Memory requirement of application	Normal distribution (256,64) MB		
Memory storage capacity	Normal distribution (2000,100) MB		
Disk requirement of application	Normal distribution (256,64) GB		
Disk storage capacity	Normal distribution (2000,100) GB		
Nominal bandwidth	100 Mbps		

1. Makespan

One of the most important and common optimization measures is "makespan reduction". The makespan is a general indicator of efficiency of the Fog system so that small values of makespan indicate that the resource allocation is efficiently. This metric is defined as the maximum execution time of all submitted applications. In other words, makespan is the completion time of the latest application. Here, we measure the makespan in milliseconds.

The simulation results of the proposed algorithm (RALA) are presented in Fig. 4 under different reward/penalty coefficients of 0.005, 0.01 and 0.1 in term of makespan. As can be seen, as the coefficients increase, the makespan decreases in small, medium and large configurations. In small configuration, the makespan for the coefficients of 0.005, 0.01 and 0.1 is 125, 110 and 98, respectively, and therefore the coefficient of 0.1 has a decrease of 27 units compared to the coefficient of 0.005. In the medium configuration, the makespan for the coefficients of 0.005, 0.01 and 0.1 is equal to 415, 380 and 310, respectively, and therefore the coefficient of 0.1 has a decrease of 105 units compared to the coefficient of 0.005. In large configuration, the makespan for the coefficients of 0.005, 0.01 and 0.1 is 1570, 1310 and 960, respectively, and therefore the coefficient of 0.1 has a decrease of 610 units compared to the coefficient of 0.005. Thus, as the configuration size increases, makespan's rate of decline has increased, since the algorithm convergence rapidly. It is contradicts the philosophy of the existence of Fog despite the reduction in makespan, because most applications are transferred to the cloud.

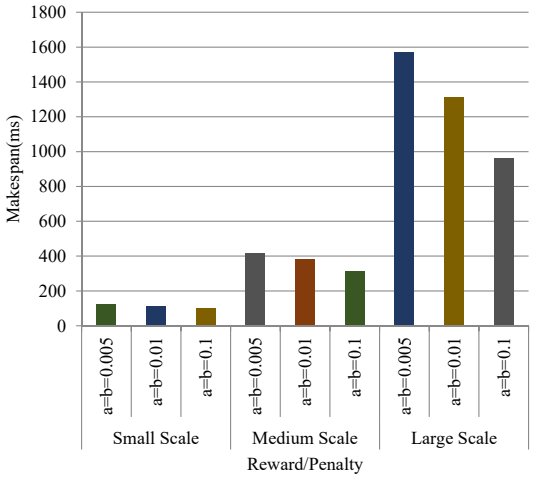

Fig. 4. The average makespan for different reward (a) and penalty (b) parameters in proposed algorithm.

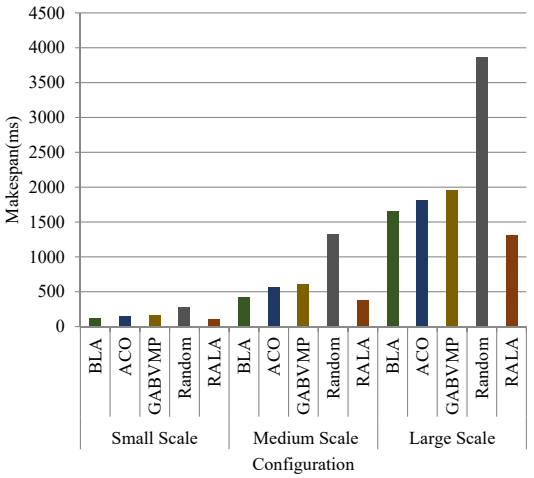

Fig. 5. The average makespan for different algorithms under different configurations.

Fig.5 shows the simulation results of the RALA algorithm, with reward and penalty coefficients of 0.01 with the BLA [25], ACO [26], GABVMP [27] and Random algorithms in term of makespan. The RALA algorithm outperforms other comparable algorithms due to the dynamic

RALA scheduling as well as the use of applications scheduling technique with a smaller response time to meet their deadline. In all configurations, the RALA algorithm differs most from the Random algorithm because it hasn't a specific strategy for assigning applications to Fog nodes so that it is 170, 940, and 2570 in small, medium, and large scales, respectively and therefore, the largest difference occurred in a large scale, which is not unexpected given the size of the configuration. However, the proposed algorithm in medium scale has the most improvement (71%) over the Random algorithm. In all configurations, the proposed algorithm has the least difference with the BLA algorithm, and this is due to the use of CPU and memory parameters in BLA algorithm to assign applications to Fog nodes. The RALA algorithm has the lowest improvement in small scale with a size of 8% and the largest improvement in large scale with a size of 21% compared to the BLA algorithm. ACO and GABVMP algorithms have similar functions due to the use of time parameter in algorithm decisions. The RALA algorithm has the lowest improvement compared to the ACO algorithm in the small scale (27%) and the highest improvement in the medium scale (32%). Similarly, the proposed algorithm has the lowest improvement compared to the GABVMP algorithm in the large scale (33%) and the highest improvement in the medium scale (38%).

2. Average Response Time

The response time is the interval between providing an application and the start of the response reception. A process mostly begins to produce an output when the application processing is continued. Thus, this criterion is better than turn-around. Here, we measure the average response time in milliseconds.

The simulation results of the proposed algorithm (RALA) are presented in Fig. 6 under different reward / penalty coefficients of 0.005, 0.01 and 0.1 in term of average response time. In all three small, medium and large configurations, the average response time at a= b= 0.01 coefficient is 40, 210 and 920, respectively, which is less than the other two coefficients, i.e. 0.005 and 0.1. In the coefficient a=b=0.005 for small, medium and large configurations is 45, 270 and 1150, respectively. At the coefficient a=b=0.1, since the system converges rapidly and also due to the transfer of more applications to the cloud. Therefore, receiving the first output from implementation of the application for Fog devices is delayed, so the average response time is the highest in all three configurations in comparison with other coefficients.

Fig.7 shows the simulation results of the RALA algorithm, with reward and penalty coefficients of 0.01 with the BLA [25], ACO [26], GABVMP [27] and Random algorithms in term of average response time. The proposed algorithm outperforms other algorithms in all 3 configurations. As mentioned in the previous subsection, this good function is due to the use of dynamic scheduling technique and performing applications with a smaller response time in this algorithm. In all configurations, the RALA algorithm has the least improvement over the BLA algorithm compared to other algorithms, so that it is 43%, 38% and 20% in small, medium and large scales, respectively, and this is due to the proximity of the two algorithm decision strategy, because the BLA algorithm also uses CPU and memory parameters to decisions. The difference in average response time for RALA and ACO algorithms is 45, 210 and 650 in small, medium and large scales, respectively, with the largest difference occurring in a large scale; however, the highest improvement in the RALA algorithm compared to the ACO algorithm occurred in the small scale (53%) and the lowest improvement in the large scale (41%). The improvement percentage of the RALA algorithm compared to the GABVMP algorithm is 58%, 59% and 44% in small, medium and large scales, respectively, so that the lowest improvement in a large scale and the highest improvement occurred in a medium scale. However, the biggest difference in the average response time occurred in large scale with 730. The RALA algorithm has the most improvement in all configurations compared to the Random algorithm, with 75%, 78%, and 57% in small, medium, and large scales, respectively.

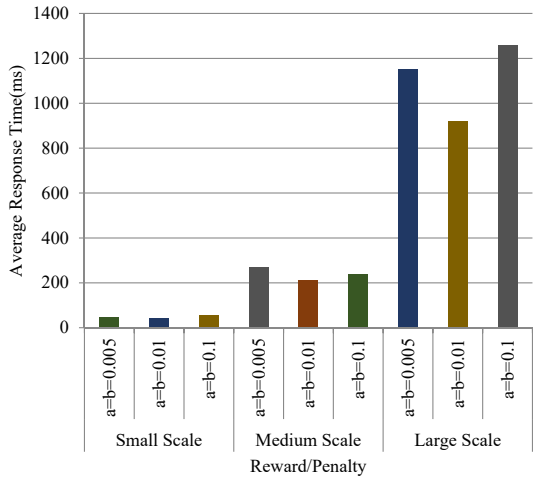

Fig. 6. The average response time for different reward (a) and penalty (b) parameters in proposed algorithm.

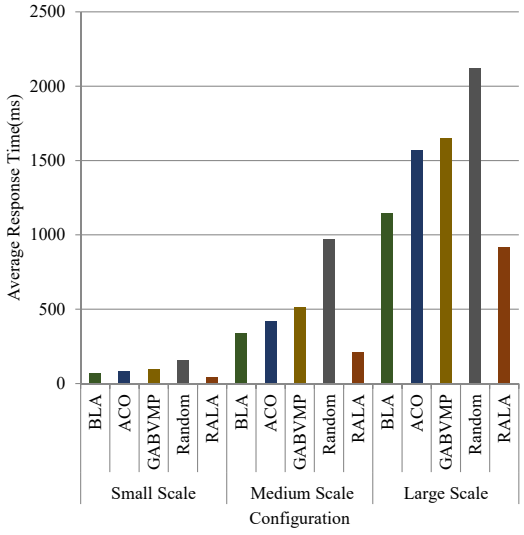

Fig. 7. The average response time for different algorithms under different configurations.

3. Load Balancing

Load balancing represents the distribution of the workload allocated to the Fog nodes. A uniform workload distribution shows the load balancing only when all the Fog nodes have the same computational capacities. However, in case of different computational capacities, the workload submitted to each Fog node must be proportional to its capacity. In this case, the standard deviation of the completion time (i.e., the time at which a Fog node completes the execution of its last application) of the Fog nodes stands for the load balancing. Makespan and response time are minimized, if the workload placed on the Fog nodes is balanced. Load balancing increases as the standard deviation of the completion time decreases. Let T denotes the completion time of all the Fog nodes. Load balancing is computed as

$$\frac{\overline{T} - \sigma_T}{\overline{T}} \times 100. \qquad (8)$$

Where $\overline{T}$ and σ_T denote the mean and standard deviation of completion time T, respectively. Here, we measure the load balancing in percentage.

The simulation results of the proposed algorithm (RALA) are presented in Fig. 8 under different reward / penalty coefficients of 0.005, 0.01 and 0.1 in term of load balancing. As the value of the coefficients increases, the load balancing in small, medium, and large configurations decreases, and this is due to the increased convergence rate of the algorithm, which prevents the load balancing from being performed correctly. As can be seen, with increasing configuration size, the rate of reduction of load balancing has increased, so that in small scale, the difference in load balancing for coefficients of 0.005 and 0.1 is equal to 7 units, while it is equal to 23 units in large scale. In the medium configuration, the load balancing for coefficients of 0.005, 0.01 and 0.1 is equal to 92, 85 and 75, respectively, so that the load balancing for the coefficient of 0.1 compared to the coefficient of 0.005 has a decrease of 17 units.

The simulation results of the RALA algorithm, with reward and penalty coefficients of 0.01 are presented in Fig. 9 with the BLA[25], ACO [26], GABVMP [27] and Random algorithms in term of load balancing. Due to the dynamic scheduling as well as the use of suitable Fog nodes to meet the requirements of application resources, the proposed algorithm outperforms other algorithms. It is obvious that as the configuration size increases, the load balancing tends to decrease. In all configurations, the RALA algorithm has the

most differences with the Random algorithm, so that it is 36, 43 and 44 in small, medium and large scales, respectively, and the largest improvement is in large scale. In all configurations, the proposed algorithm has the least difference with the BLA algorithm. As mentioned earlier, this is due to the proximity of the two algorithm decision strategy. In small, medium and large configurations, the difference between RALA and BLA algorithms is 4, 10 and 16, respectively. The percentage improvement of the RALA algorithm compared to the BLA algorithm is 5%, 13% and 26% in small, medium and large scales, respectively. As it turns out, the biggest difference and the highest percentage of improvement has occurred in large scale. The improvement percentage of the RALA algorithm compared to the GABVMP algorithm is 21%, 63% and 63% in small, medium and large scales, respectively. The biggest difference in load balancing between RALA and GABVMO algorithms has occurred in medium scale of 33. The difference between RALA and ACO algorithms is 11, 21 and 27 in small, medium and large scales, respectively, but the highest improvement in the RALA algorithm compared to the ACO algorithm occurred in the large scale (53%) and the lowest improvement in the small scale (14%).

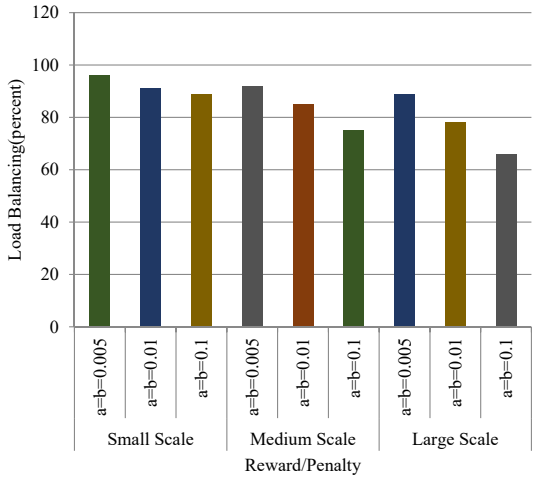

Fig.8. The load balancing for different reward (a) and penalty (b) parameters in proposed algorithm.

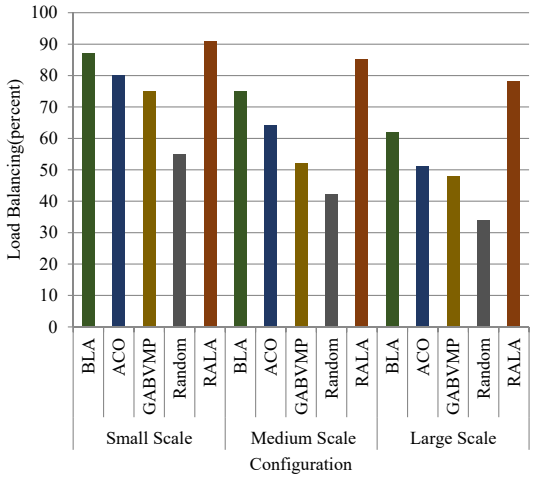

Fig9. The load balancing for different algorithms under different configurations.

4. Throughput

Throughput is another important criterion that indicates the number of applications that are processed per unit time. The simulation results of the proposed algorithm (RALA) are presented in Fig. 10 under different reward/penalty coefficients of 0.005, 0.01 and 0.1, respectively.

As the coefficients increase, the throughput in small, medium, and large configurations decreases, since the algorithm convergence rapidly, but it is not appropriate because it contradicts the existential philosophy of Fog, because most applications are transferred to the cloud. In small configuration, the throughout for the coefficients of 0.005, 0.01 and 0.1 are 850, 910 and 950, respectively. In medium configuration, the throughout for the coefficients of 0.005, 0.01 and 0.1 are 1250, 1810 and 1915, respectively. In the large configuration, the throughout for the coefficients of 0.005, 0.01 and 0.1 are 6230, 7450 and 7820, respectively. The difference in throughout for the coefficients of 0.005 and 0.1 in small, medium and large scales is 100, 665 and 1590, respectively, and therefore it can be concluded that in large scale, most applications transformed to the cloud.

The simulation results of the RALA algorithm are presented In Fig. 11 with reward and penalty coefficients of 0.01, with the BLA [25], ACO [26], GABVMP [27] and Random algorithms in term of throughput. In all configurations, the RALA

algorithm has the least improvement over the BLA algorithm compared to other algorithms, so that it is 5%, 19%, and 14% in small, medium, and large scales, respectively, and the largest improvement occurred in medium scale. In all configurations, the RALA algorithm has the most differences with the Random algorithm, so that it is 250, 890 and 3130 in small, medium and large scales, respectively, and the largest difference occurred in large scale that has not been unexpected due to the configuration size. The difference in throughput for RALA and ACO algorithms is 100, 600, and 940 in small, medium, and large scales, respectively, with the largest difference occurring in large scale, but the highest improvement in the RALA algorithm compared to the ACO algorithm occurred in medium scale (53%) and the lowest improvement in small scale (12%). The percentage improvement of RALA algorithm over GABVMP algorithm is 15%, 56% and 19% in small, medium and large scales, respectively, so that the lowest improvement occurred in small scale and the highest improvement occurred in medium scale. However, the biggest difference in the throughput occurred in large scale with a size of 1240.

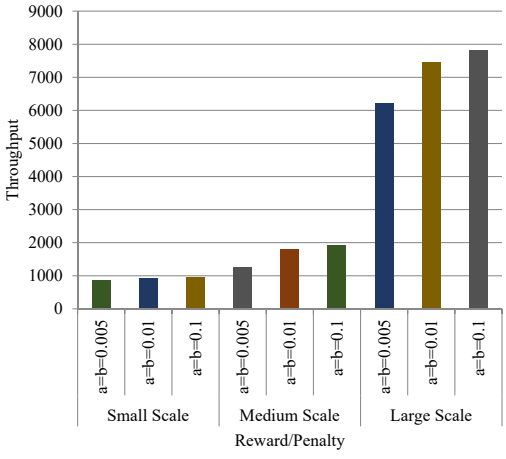

Fig.10. The throughput for different reward (a) and penalty (b) parameters in proposed algorithm.

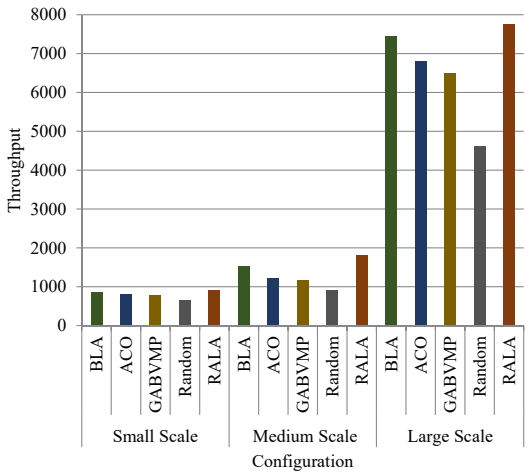

Fig.11. The throughput for different algorithms under different configurations.

VI. CONCLUSION

The present study proposed a learning automata-based algorithm to solve the resource allocation problem in Fog computing. In this method, two learning automata are used to select applications (LA_{APP}) and Fog nodes (LA_N). Initially, the probability values are assigned to the action set of LA_{APP}. Then an application is then selected based on the LA_{APP} probability vector. If the selected application has the shortest deadline among the action set, then the algorithm enters the next step, otherwise the selection of the application takes place again. Next, the probability values are assigned to the action set of LA_N. At this step, a Fog node is selected based on the LA_N probability vector, and the response time of the application must be set. If the selected Fog node is able to meet the requirements of the resource and deadline of the selected application, then the algorithm enters the next step; otherwise the selection of the Fog node takes place again. If all the conditions are met, then the selected Fog node resources should be allocated to the selected application. After the execution of the application is completed by Fog node, then the application resources will be released and will be added to the list of free resources of Fog node.

For the purpose of demonstrate the efficiency of the proposed algorithm, several simulations have been performed on three configurations: small, medium and large scales. Finally, the results of the proposed algorithm were compared with the results of BLA, ACO, GABVMP and Random algorithms. According to the obtained results, the proposed algorithm outperforms the other mentioned algorithms in terms of makespan, average response time, load balancing and throughput.

REFERENCES

1. R. Huang, Y. Sun, C. Huang, G. Zhao, Y. Ma. A survey on fog computing. International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. Springer, 2019. p. 160-9.
2. R. Mahmud, R. Kotagiri, R. Buyya. Fog computing: A taxonomy, survey and future directions. Internet of everything. Springer, 2018. p. 103-30.
3. M. Ghobaei-Arani, A. Souri, A.A. Rahmanian. Resource management approaches in fog computing: a comprehensive review. Journal of Grid Computing. (2019) 1-42.
4. F. Bonomi, R. Milito, J. Zhu, S. Addepalli. Fog computing and its role in the internet of things. Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 2012. p. 13-6.
5. M. Mukherjee, L. Shu, D. Wang. Survey of fog computing: Fundamental, network applications, and research challenges. IEEE Communications Surveys & Tutorials. 20 (2018) 1826-57.
6. H. Ghorashi, and M. Mirabi, An Effective Task Scheduling Framework for Cloud Computing using NSGA-II. Journal of Advances in Computer Engineering and Technology, 2020. 6(3): p. 151-160.
7. M. Bozorgi Elize, and A. KhademZadeh, A Genetic Based Resource Management Algorithm Considering Energy Efficiency in Cloud Computing Systems. Journal of Advances in Computer Engineering and Technology, 2017. 3(4): p. 203-212.
8. M. Davarpanah, , A review of methods for resource allocation and operational framework in cloud computing. Journal of Advances in Computer Engineering and Technology, 2017. 3: p. 51-60.
9. C.-H. Hong, B. Varghese. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. ACM Computing Surveys (CSUR). 52 (2019) 1-37.
10. F. Xhafa, A. Abraham. Computational models and heuristic methods for Grid scheduling problems. Future generation computer systems. 26 (2010) 608-21.
11. K. Singh, A. Chhabra, A. GNDU. A Survey of Evolutionary Heuristic Algorithm for Job Scheduling in Grid Computing. International Journal of Computer Science and Mobile Computing. 4 (2015) 611-6.
12. T. Remani, E. Jasmin, T.I. Ahamed. Residential load scheduling with renewable generation in the smart grid: A reinforcement learning approach. IEEE Systems Journal. (2018).
13. M. Rezapoor Mirsaleh, M.R. Meybodi. Balancing exploration and exploitation in memetic algorithms: a learning automata approach. Computational Intelligence. 34 (2018) 282-309.
14. A. Rezvanian, A.M. Saghiri, S.M. Vahidipour, M. Esnaashari, M.R. Meybodi. Recent advances in learning

automata. Springer, 2018.

15. M.M.D. Khomami, A. Rezvanian, M.R. Meybodi. A new cellular learning automata-based algorithm for community detection in complex social networks. Journal of computational science. 24 (2018) 413-26.

16. H. Morshedlou, M.R. Meybodi. A new learning automata based approach for increasing utility of service providers. International Journal of Communication Systems. 31 (2018) e3459.

17. M. Hasanzadeh-Mofrad, A. Rezvanian. Learning automata clustering. Journal of computational science. 24 (2018) 379-88.

18. M. Ranjbari, J.A. Torkestani. A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers. Journal of Parallel and Distributed Computing. 113 (2018) 55-62.

19. S.M. Vahidipour, M. Esnaashari, A. Rezvanian, M.R. Meybodi. GAPN-LA: A framework for solving graph problems using Petri nets and learning automata. Engineering Applications of Artificial Intelligence. 77 (2019) 255-67.

20. E. Susmitha, B.R. Devi. Pipelined Learning Automation for Energy Distribution in Smart Grid. International Conference on E-Business and Telecommunications. Springer, 2019. p. 732-42.

21. A. Yazidi, X. Zhang, L. Jiao, B.J. Oommen. The hierarchical continuous pursuit learning automation: a novel scheme for environments with large numbers of actions. IEEE transactions on neural networks and learning systems. (2019).

22. M. Jamshidi, M. Esnaashari, A.M. Darwesh, M.R. Meybodi. Detecting Sybil nodes in stationary wireless sensor networks using learning automaton and client puzzles. IET Communications. 13 (2019) 1988-97.

23. A.M. Saghiri, M.D. Khomami, M.R. Meybodi. Random Walk Algorithms: Definitions, Weaknesses, and Learning Automata-Based Approach. Intelligent Random Walk: An Approach Based on Learning Automata. Springer, 2019. p. 1-7.

24. A. Enami, J.A. Torkestani, A. Karimi. Resource selection in computational grids based on learning automata. Expert Systems with Applications. 125 (2019) 369-77.

25. S. Bitam, S. Zeadally, A. Mellouk. Fog computing job scheduling optimization based on bees swarm. Enterprise Information Systems. 12 (2018) 373-97.

26. E. Ghaffari. Providing a new scheduling method in fog network using the ant colony algorithm. Collection of Articles on Computer Science. (2019).

27. S.B. Akintoye, A. Bagula. Improving quality-of-service in cloud/fog computing through efficient resource allocation. Sensors. 19 (2019) 1267.

28. G. Li, Y. Liu, J. Wu, D. Lin, S. Zhao. Methods of resource scheduling based on optimized fuzzy clustering in fog computing. Sensors. 19 (2019) 2122.

29. J. Wang, D. Li. Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing. Sensors. 19 (2019) 1023.

30. W.-C. Yeh, C.-M. Lai, K.-C. Tseng. Fog computing task scheduling optimization based on multi-objective simplified swarm optimization. Journal of Physics: Conference Series. IOP Publishing, 2019. p. 012007.

31. H. Wang, L. Wang, Z. Zhou, X. Tao, G. Pau, F. Arena. Blockchain-Based Resource Allocation Model in Fog Computing. Applied Sciences. 9 (2019) 5538.

32. L. Yin, J. Luo, H. Luo. Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. IEEE Transactions on Industrial Informatics. 14 (2018) 4712-21.

33. H. Zhang, Y. Xiao, S. Bu, D. Niyato, F.R. Yu, Z. Han. Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining Stackelberg game and matching. IEEE Internet of Things Journal. 4 (2017) 1204-15.

34. S. Jošilo, G. Dán. Decentralized algorithm for randomized task allocation in fog computing systems. IEEE/ACM Transactions on Networking. 27 (2018) 85-97.

35. M. Mtshali, H. Kobo, S. Dlamini, M. Adigun, P. Mudali. Multi-Objective Optimization Approach for Task Scheduling in Fog Computing. 2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD). IEEE, 2019. p. 1-6.

36. X. Xu, S. Fu, Q. Cai, W. Tian, W. Liu, W. Dou, et al. Dynamic resource allocation for load balancing in fog environment. Wireless Communications and Mobile Computing. 2018 (2018).

37. M. Thathachar, B.R. Harita. Learning automata with changing number of actions. IEEE Transactions on Systems, Man, and Cybernetics. 17 (1987) 1095-100.

38. M. Thathachar, K. Narendra. Learning Automata: an Introduction, 1989.

39. H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Software: Practice and Experience. 47 (2017) 1275-96.