Journal of Information Systems and Telecommunication
Vol.12, No.1, January-March 2024, 29-40

http://jist.acecr.org
ISSN 2322-1437 / EISSN:2345-2773

TPALA: Two Phase Adaptive Algorithm based on Learning Automata for job scheduling in cloud Environment

Abolfazl Esfandi[1], Javad Akbari Torkestani[1*], Abbas Karimi[1], Faraneh Zarafshan[1]

[1.]Department of Computer Engineering, Arak Branch, Islamic Azad University, Arak, Iran.

Received: 30Aug 2022/ Revised: 04 Sep 2023/ Accepted: 21 Oct 2023

Abstract

Due to the completely random and dynamic nature of the cloud environment, as well as the high volume of jobs, one of the significant challenges in this environment is proper online job scheduling. Most of the algorithms are presented based on heuristic and meta-heuristic approaches, which result in their inability to adapt to the dynamic nature of resources and cloud conditions. In this paper, we present a distributed online algorithm with the use of two different learning automata for each scheduler to schedule the jobs optimally. In this algorithm, the placed workload on every virtual machine is proportional to its computational capacity and changes with time based on the cloud and submitted job conditions. In proposed algorithm, two separate phases and two different LA are used to schedule jobs and allocate each job to the appropriate VM, so that a two phase adaptive algorithm based on LA is presented called TPALA. To demonstrate the effectiveness of our method, several scenarios have been simulated by CloudSim, in which several main metrics such as makespan, success rate, average waiting time, and degree of imbalance will be checked plus their comparison with other existing algorithms. The results show that TPALA performs at least 4.5% better than the closest measured algorithm.

Keywords: Cloud Computing; Job scheduling; Learning Automata; Virtual Machine; CloudSim; Simulation.

1- Introduction

Cloud computing is a computer model that attempts to facilitate user access based on the type of demand they have from information and computing resources. This model tries to respond to the needs of users by reducing the need for human resources and costs as well as enhancing the speed of access to information [1]. By increasing the demand for running applications in the cloud, especially large and shared applications, Scheduling strategies for cloud requests have become very important. The key issue and challenge in cloud computing is ensuring the satisfaction of all users with cloud services. The scheduling plan consists of a scheduling algorithm that should plan the jobs and applications, so that users are satisfied and not harm the cloud manager at the same time[2]. Job scheduling is a key process in the IaaS layer that aims to execute logged requests to the system on the resources, in an efficient way by considering other features of the cloud environment. Job scheduling considers virtual machines as computing units to allocate heterogeneous physical resources for job execution. Each virtual machine is a unit containing computing and storage capabilities provided in the cloud[3]. Job scheduling in the cloud environment is NP-Hard due to its dynamic characteristics, heterogeneity, and varying workloads of users. In such a system, the scheduling algorithm must be done automatic and very quickly [4]. In cloud environments, according to the different needs of users, the workload of each user and as a result the computing resources required by them, which in our discussion are virtual machines, is different. Some users need more computing resources and others need fewer computing resources. In case of improper allocation of resources in any of the situations, the efficiency of the system will decrease significantly. Therefore, it is not possible to assign the same resources to all users. In addition, due to the dynamism of the cloud environment, the workload of users may change over time. Therefore, an efficient scheduling algorithm should dynamically allocate the most appropriate resource (virtual machine) to the jobs, according to the user's workload. Various algorithms have been presented for the job scheduling problem, but according to the review of the previous algorithms, it can be said that those algorithms

✉ **Javad Akbari Torkestani**
j-akbari@iau-arak.ac.ir

have not been able to fully adapt themselves to the dynamics of the cloud conditions and the diversity of resources [5].

The purpose of this paper is to propose an efficient algorithm for finding a near-optimal solution to the job scheduling problem in the cloud environment. Considering factors such as the fully dynamic environment of the cloud, the different capacity of virtual machines (VMs), the complexity and largeness of the requested jobs, the efficiency of cloud computing will be completely dependent on the existence of an effective scheduling algorithm.

The main difference between our proposed approach and previous methods that have used learning automata is in the way learning automata are employed and the type of learning automata used. Our proposed algorithm uses two different learning automata (LAs) for each scheduler to schedule the jobs optimally, because there are two main challenges for scheduling jobs generally. One challenge is choosing the proper job among the submitted jobs from users based on the priority and specific conditions of each job, and the other challenge is assigning that job to the most appropriate VM. Actually, according to these two challenges, our algorithm uses the two mentioned LA in two phases. The important contributions of this paper are as follows:

- Using two different learning automata (LAs) for each scheduler to optimally schedule the jobs
- Creating a list of ready-to-use virtual machines simultaneously with the first phase of the algorithm.
- Applying the variable learning automaton in the second phase increases the speed of convergence.
- Providing an online adaptive job scheduling method for cloud environment by using the two different LAs in two phases.
- Dynamically assigning workload to each virtual machine based on the VM's status and the submitted job's conditions.
- Using a hybrid set of jobs for simulations based on two factors: data volume and computational volume.
- Simulating the proposed algorithm using the CloudSim toolkit under different scenarios and comparing it to other scheduling algorithms.

Regarding the paper structure, Section 2 will first provide a comprehensive overview of the relevant and existing works in job scheduling concept in cloud. Then, Section 3 discusses the learning automata and its features. Section 4 describes our new proposed algorithm, and then, the implementation as well as experimental results of the simulation with the CloudSim toolkit and comparing it with other existing algorithms is shown in Section 5. Finally, Section 6 concludes the paper.

2- Related Works

From the birth of cloud computing to this day, there has always been extensive research into scheduling, and as a result, different methods and algorithms have been introduced. Some methods use heuristic algorithm such as Scheduling based on Min-Min [6] or Min-Max [7] Strategy. Researchers in [8] and [9] have investigated that in methods based on meta-heuristics optimization algorithms such as GA. papers [10-12] was presented its methods based on GA. In paper [13], the ant colony optimization algorithm (ACO) is used, in which several artificial ants generate distinct responses randomly based on the amount of pheromones. In [14], the ACO search method with GA was used for the job scheduling problem. In ref [15] an optimization strategy is proposed by using improved ACO algorithm for task scheduling in cloud. In paper [16] a metaheuristics method named GWOA is described that firstly proposed to overcome the early convergence problem and then create a balance between the local and the global search. A hybrid method combining the bee optimization and whale optimization model is proposed in the paper [17]. A fuzzy clustering method is used in [18] as a pre-processing operation to classify cloud resources; then directed graphs are used to schedule jobs to run on distinct clusters of hardware resources. In paper [19] was used fuzzy control theory for accomplish system accessibility between user requirements and users resources availability. In [20], the particle swarm optimization (PSO) has been used as an optimal answer searching method for the optimization problem and paper [21] was proposed hybrid task scheduling method by PSO and GA. Also paper [22] is presented a survey of scheduling algorithms based on PSO in cloud computing. In paper [23] a Hybrid Particle Swarm Optimization (HPSO) based scheduling was presented to optimal job scheduling in the cloud. HPSO against PSO was improved the performance of job scheduling issue by changing the various factors. paper [24] was proposed a hybrid job scheduling algorithm based on Fuzzy system and Modified PSO technique to improve load balancing and throughput in cloud environment. Also paper [25] by using integrated particle swarm algorithm and ant colony algorithm, proposed a efficiency algorithm for task scheduling. In paper [26] was used A Bidirectional Search Algorithm for Flow Scheduling in Cloud Data Centers. This approach was used for static scheduling and reduced makespan. In paper [27], a hybrid method named FUGE based on the fuzzy and GA was presented that reduce the execution time and costs. A combined method of the cellular automata and the bat algorithm (BatCL) was presented in paper [28] which aimed to reduce the cost and time of completion of tasks. Paper [29] proposes a hybrid approach for job scheduling in cloud computing, utilizing a combination of sparrow search algorithm and differential evolution optimization. A combination of the genetic algorithm and the gravitational emulation local search is presented in [30] for task scheduling in the cloud. In [31], a new method was proposed based on LA for energy-aware task scheduling to minimize energy consumption and task completion time in cloud environment. This paper presented a scheduling architecture by LA for optimal job assignment. The paper [32] described a new LA-based scheme in which load distribution was performed in such a way that the level of efficiency in different nodes was almost the same which also used paradigm of two-time scales to achieve its purpose. A LA Based algorithm for container cloud was presented in [33] which designed a task load monitoring framework for usage in real-time monitoring of resource and scheduling evaluation feedback, develop an intelligent scheduling technique, and enhance the performance.

Articles, [34-38], had a brief overview on existed scheduling algorithms.

In table 1, the related works have been categorized into four groups and their general advantages and disadvantages have been stated for each category.

Table 1: Job scheduling algorithms comparison

References	Category	Advantages	Disadvantages
[6, 7]	Heuristic Algorithms	• simplicity of the algorithm structure	• static scheduling • low compatibility with dynamic environments • single goal
[18, 19, 24, 27]	Fuzzy Theory	• making best decision based on inputs	• mostly using in combination • using for static environments
[13, 15-17, 20, 23, 29, 30]	Meta-heuristic Optimization	• searching optimal solution from all solution space • obtain better optimization effect	• complexity scheduling • long optimization time • getting caught in local optimal solutions
[31-33]	Automata Theory	• proper for dynamic environments • global optimization ability • suitable for large-scale job scheduling	• weak directivity in optimal solution searching • Easy to fall into the local optimum

3- Learning Automata theory

Learning automaton is one of the reinforcement learning techniques in artificial intelligence. Learning automata's learning ability in unknown environments is a useful technique for modeling, controlling, and solving many real problems in the distributed and decentralized environments[39]. The environment responds to the action taken in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement signal back from the environment. The purpose of a LA is finding the optimal action from the action set so it was received minimized average penalty from the environment[40].

Figure 1 illustrates interactive between a learning automaton and its random environment that vector α(n) is an action vector, vector β(n) is a reinforcement signal and vector x(n) is called context vector. In nth iteration, an automaton receives x(n) from the environment. Depending on x(n), LA chooses one of its possible actions[41].

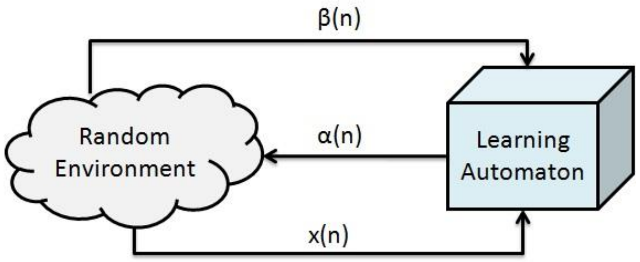


Fig. 1: interactive between a LA and its random environment

The environment can be mathematically modeled by quintuple $< \underline{X}, \underline{\alpha}, \underline{\beta}, \underline{F}, \underline{q} >$ where:

$\underline{X}$: is the set of context vectors

$\underline{\alpha}$: is the set of inputs

$\underline{\beta}$: is the set of values that can be taken by the reinforcement signal $\underline{F}$

$\underline{F}$: is the set of probability distributions

$\underline{q}$: is the probability distributions defined over $\underline{X}$ which is assumed to be unknown.

$\alpha(n)$ and $\beta(n)$ denote the input and output of environment at discrete time n ($n \geq 0$)[42].

The learning algorithm used to update the action probability based on a recurrence relation. Let $\alpha_i(n) \in \underline{\alpha}$ denotes the action selected by LA and P(n) denotes the probability vector defined over $\underline{\alpha}$ at instant n. Let a and b denote the reward and penalty parameters and r be shown the number of actions that can be taken by LA. At each instant n, if the selected action $\alpha_i(n)$ is rewarded by random environment, the action probability vector P(n) is modified by the linear learning algorithm given in Eq.1 and if the taken action is penalized, it is modified as given in Eq.2.

$$P_i(n+1) = P_i(n) + a[1 - P_i(n)]$$
$$P_j(n+1) = (1-a)P_j(n) \qquad \forall j; j \neq i \qquad (1)$$

$$P_i(n+1) = (1-b)P_i(n)$$
$$P_j(n+1) = \frac{b}{r-1} + (1-b)P_j(n) \qquad \forall j; j \neq i \qquad (2)$$

If a and b be equal, Eq.1 and Eq.2 are called a linear reward-penalty (L_{R-P}) algorithm, if a be more greater than b those equations are called linear reward-Ɛ penalty ($L_{R-\varepsilon P}$) and if b=0 they are named linear reward-inaction (L_{R-I})[41]. A variable action-set learning automaton (VLA) is a LA that the number of its actions maybe varies with time. If $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ be action-set of VLA, $A = \{A_1, A_2, \dots, A_m\}$ is the set of action subsets and $A(n) \subseteq \alpha$ denote the subset of all the available actions for choose by the VLA, at instant n.
The special action subsets are chosen randomly by an external agency according to the probability distribution $\psi(n) = \{\psi_1(n), \psi_2(n), \dots, \psi_m(n)\}$ so that $\psi_i(n) = prob[A(n) = A_i | A_i \in \mathrm{A}, 1 \leq \mathrm{i} < 2^{\mathrm{r}}]$.
Let $\hat{p}_i(n) = prob[\alpha(n) = \alpha_i | A(n), \alpha_i \in A(n)]$ be the probability of selecting action α_i if the action subset A(n) has already been selected and $\alpha_i \in A(n)$. That is defined as:

$$\hat{p}_i(n) = p_i(n)/K(n) \qquad (3)$$

Where $p_i(n) = prob[\alpha(n) = \alpha_i]$ and K(n) is the sum of the probabilities of the actions in subset A(k) which defined as:

$$K(n) = \sum_{\alpha_i \in A(n)} p_i(n) \qquad (4)$$

The function of a VLA is as follows: Before selecting an action from the selected subset, the probabilities of all the actions in the A(n) are calculated as defined in Eq.(3). Then VLA randomly selects one of actions of the A(n) based on the scaled action probability vector $\hat{p}_n$. Depending on the response received from the environment, the VLA updates its scaled action probability vector (only the available actions). Finally, the probability vector of the actions of the A(n) is rescaled as:

$$p_i(n+1) = \hat{p}_i(n+1) * K(n), \quad \forall \alpha_i \in A(n) \qquad (5)$$

for more details and Proof refer to [43].

4- Proposed Algorithm

As mentioned, various algorithms have been proposed for job scheduling in the cloud environment so far. However, most of them are not efficient enough due to the dynamic nature of the cloud environment, such as resource dynamics and network conditions. To represent our proposed method, First, we explain the cloud environment, the desired scheduling structure, and the general method of our algorithm, then we describe how each of the LA and the two mentioned phases work.
Let cloud contains several hosts that each of them has multiple virtual machines (VMs), so that the sum of all virtual machines in all Hosts is considered as m virtual machines. The intended resources are in the form of these virtual machines. We also consider k schedulers for scheduling submitted users jobs, each scheduler $s_i \subseteq S$ is associated with one or more VMs and each virtual machine $VM_j \subseteq VM$ can also be connected to one or more schedulers. Also, each virtual machine has set of processing element PE_j so that each processing element $PE_j(k)$ has different processing powers. In this algorithm, two separate phases and two different LA are used to schedule jobs and allocate each job to the appropriate VM, so that a two phase adaptive algorithm based on LA is presented, which we call TPALA. In our algorithm, each scheduler $s_i \subseteq S$ has two LA. The first LA (LA_i^{job}) has the task of selecting and receiving the submitted jobs by users with different workloads, and the second LA (LA_i^{VM}) has the task of optimal allocating the jobs to proper VMs based on their computing capacity. To better understand the proposed algorithm, in the Table 2, the list of most symbols used and their meaning is brought.

Table 2: Symbols specifications

Symbols	**Description**
LA_i^{job}	First LA on scheduler S_i for selecting the jobs
LA_i^{VM}	Second LA on scheduler Si for optimal allocating the job to proper VM
RU_i	List of ready-to-use virtual machines on scheduler S_i
J_i^{sel}	Selected job by LA_i^{job} on scheduler S_i
VM_i^{opt}	Selected VM by LA_i^{VM} on scheduler S_i
α_i^{job}	The action-set of learning automaton LA_i^{job}
$\alpha_i^{job}(j)$	Corresponding action of LA_i^{job} to user jth
P_i^{job}	The action probability vector of LA_i^{job}
PE_j	Set of processing elements of VM_j
$PE_j(k)$	Kth processing elements of VM_j

$T_i^{sel}(l)$	l th task of job J_i^{sel}
α_i^{VM}	The action-set of learning automaton LA_i^{VM}
$\alpha_i^{VM}(j,k,l)$	Corresponding action of LA_i^{VM} to $PE_j(k)$ for allocating to task $T_i^{sel}(l)$.
$RC_i^{sel}(l)$	The required capacity for the task $T_i^{sel}(l)$
$AC_j(k)$	The available capacity of the processing unit $PE_j(k)$
$\overline{X}_i(AC)$	The average available capacities of all VM which have related to scheduler S_i

Now, according to what that has mentioned, we are going to describe TPALA algorithm step by step. As shown in the flowchart in Figure 3, in each stage, first, each VMs corresponding to the scheduler S_i, which has completed its previous work or is idle (has ready PE to work), generates a "request" signal for a new job, including its own characteristic, and send it for Si. In fact, in this way, a list of ready-to-use virtual machines is created, which we call RUi. Also, in RUi, the usable computing capacity of the VMs corresponding to the scheduler S_i is specified.
As it will be said in the explanations of the second phase, the existence of this list (RU_i) reduces the number of action-set used in the second phase and increases the rate and speed of convergence. After send request signal, VM waits for receive proper job or "retry" signal.

```
Input:
Output: list of ready-to-use VMs (RU_i)
01:    For each $VM_j \subseteq$ VMs corresponding to $S_i$ do
02:        If $VM_j$.IsReady(PE) == True then
03:            $VM_j$.Send("request", $S_i$)
04:            $VM_j$.Wait(response, $S_i$)
05:        End If
06:    End For
07:    $S_i$.Collects("request")
08:    $RU_i \leftarrow S_i$.Create(list of ready-to-use VMs)
09:    Return $RU_i$
```

Fig. 2: Pseudo code of create RUi for scheduler S_i

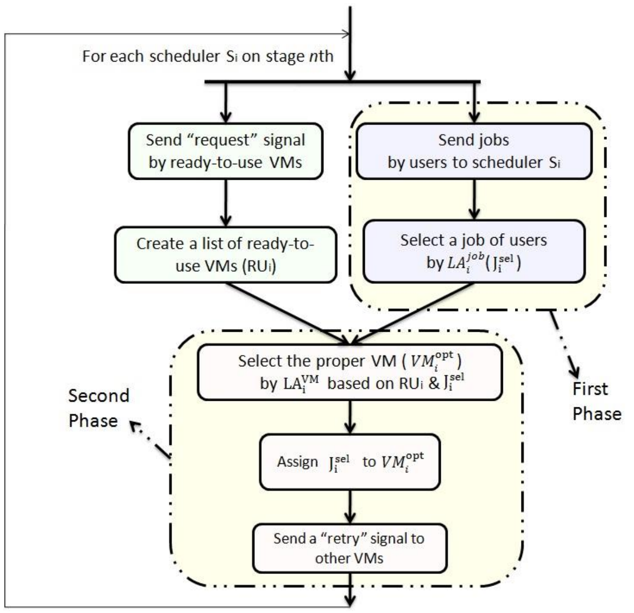

For each scheduler Si on stage nth
Send "request" signal by ready-to-use VMs
Create a list of ready-to-use VMs (RUi)
Send jobs by users to scheduler Si
Select a job of users by $LA_i^{job}(J_i^{sel})$
Select the proper VM (VM_i^{opt}) by LA_i^{VM} based on RUi & J_i^{sel}
Assign J_i^{sel} to VM_i^{opt}
Send a "retry" signal to other VMs
Second Phase
First Phase

Fig. 3: General Flowchart of TPALA algorithm

Simultaneously with this event, in the first phase, for each scheduler $s_i \subseteq S$ by using LA_i^{job} that its operation will be explained later (see pseudo code of Figure 4), one of the jobs sent by users to scheduler S_i is selected (J_i^{sel}). In the second phase of TPALA algorithm, for scheduler S_i, According to the selected job (LA_i^{job}) in the first phase and the list RU_i, by using LA_i^{VM} that its operation will be explained later (see pseudo code of Figure 5), based on the characteristics of the VMs and their computational capacity against the required computational capacity of selected job, J_i^{sel} is assigned to the most suitable VM. After selecting the proper VM (VM_i^{opt}), J_i^{sel} is assigned to VM_i^{opt} and a retry signal is sent to other VMs. These steps are repeated until all the users' jobs are serviced.
Now that the general structure of the TPALA algorithm has been explained, we will describe the details of the operation of each of the phases, that is, the operation of the two LAs related to each of schedulers.
As it was said, in the first phase for schedule S_i, learning automaton LA_i^{job} is used to select J_i^{sel}. Scheduler Si may receive jobs from several users. Let $U = \{U_1, U_2, \dots, U_R\}$ be the set of users that corresponded to scheduler S_i.
The action-set of learning automaton LA_i^{job} has R actions (one action for any user). That is defined as:

$$\alpha_i^{job} = \left\{\alpha_i^{job}(j) \middle| \forall j : j = 1,2,3,\dots,R\right\} \quad (6)$$

If action $\alpha_i^{job}(j)$ be selected, it is means that scheduler S_i allows user U_j to submits its job. Also the action probability vector of LA_i^{job} is defined as:

$$\mathrm{P}_i^{job} = \left\{\mathrm{p}_i^{job}(j) \middle| \forall j : j = 1,2,3,\ldots,R\right\} \quad (7)$$

Since we have R actions, all Actions will have same probability $1/_{R}$. Therefore all users in starting point have the same chance to send their jobs to cloud. If priority is considered for user jobs, instead of considering the equal probability for all actions, different values can be considered according to the priority of jobs and the initial probability of jobs with higher priority can be considered slightly higher. But in any case, the sum of the possibilities must be equal to one, that is:

$$\forall i \in \{1,2,\ldots,k\} \;:\; \sum_{j=1}^{R} \mathrm{P}_i^{job}(j) = 1 \quad (8)$$

For scheduler S_i at any stage, LA_i^{job} as per its action probability vector select one of its actions randomly. Let action $\alpha_i^{job}(j)$ be selected, then scheduler Si checks user Uj that it has a ready job to send. If user Uj could be submit a job (J_i^{sel}), selected action $\alpha_i^{job}(j)$ gets rewards and increases action $p_i^{job}(j)$ by Eq.(1). Otherwise selected action $\alpha_i^{job}(j)$ gets penalty and decreases action $p_i^{job}(j)$ by Eq.(2). Pseudo code of first phase is shown in Figure 4. In any case (reward/penalty) after updating the internal state of LA_i^{job}, this stage will be end and start next stage and repeat same method. By applying of this method, effectively workload on users can be balanced.

```
Input: jobs queue
Output: a job form jobs queue of $S_i$ ($J_i^{sel}$)
01:  $LA_i^{job}$.Initial()
02:  $\alpha_i^{job}(j) \leftarrow LA_i^{job}$. Select-Action($P_i^{job}$)
03:  If $U_j$.HasReady-job == True then
04:      update $LA_i^{job}$ by Eq.1 //reward
05:      $J_i^{sel} \leftarrow S_i$.Selected-job()
06:      Return $J_i^{sel}$
07:  Else
08:      update $LA_i^{job}$ by Eq.2 //penalty
09:  End If
```

Fig. 4: Pseudo code of one round of first phase of TPALA algorithm

After J_i^{sel} determined, second phase by learning automaton LA_i^{VM} can be run. In this phase LA_i^{VM} find a near optimal solution to allocate J_i^{sel} to proper VM based on their computational capacities.

Scheduler S_i has related to several VMs on cloud environment. Let $\{VM_1, VM_2, \ldots, VM_N\}$ be the set of VMs that corresponded to scheduler S_i.

Every virtual machine VM_j have set of processing elements PE_j so that each processing element $PE_j(k)$ has different processing powers. This set defined as:

$$PE_j = \left\{PE_j(k) \middle| \forall k : k = 1,2,3,\ldots,m\right\} \quad (9)$$

Other hand, each job J_i^{sel} divided to several tasks $T_i^{sel}(l)$ $(\forall l \in \{1,2,..,H\})$. In this phase, purpose of learning automaton LA_i^{VM} is to find optimal scheduling method to allocate a processing element $PE_j(k)$ from RU_i to any task $T_i^{sel}(l)$. The used learning automaton LA_i^{VM} is a VLA and its action-set is defined as:

$$\alpha_i^{VM} = \left\{\alpha_i^{VM}(j,k,l) \middle| \forall\, PE_j(k) \in PE_j\right\} \quad (10)$$

If action $\alpha_i^{VM}(j,k,l)$ be selected, it is means that scheduler S_i chooses processing element $PE_j(k)$ to allocated to any task $T_i^{sel}(l)$. As mentioned, we use a VLA in the second phase. The reason is that due to the non-constant number of actions in this type of automata, the rate and speed of convergence increases and it has a more dynamic structure.

To bound the action-set of LA_i^{VM}, let $RC_i^{sel}(l)$ be the required capacity for the task $T_i^{sel}(l)$ and $AC_j(k)$ is the available capacity of the processing unit $PE_j(k)$, if $RC_i^{sel}(l)$ is less than or equal to $AC_j(k)$, then $PE_j(k)$ can be allocated to the $T_i^{sel}(l)$. Otherwise, there is no possibility of allocation. Therefore, the existence of the action $\alpha_i^{VM}(j,k,l)$ is not useful and it can be deleted from the action-set of LA_i^{VM} as explained in section 3.

Therefore for each task $T_i^{sel}(l)$, action-set Learning automaton LA_i^{VM} will be updated as:

$$\alpha_i^{VM} \leftarrow \alpha_i^{VM} - \left\{\alpha_i^{VM}(j,k,l) \middle| RC_i^{sel}(l) \le AC_j(k)\right\} \quad (11)$$

Let AC_j be the sum of available capacities of virtual machine VM_j that is defined as:

$$AC_j = \sum_{k=1}^{N_j} AC_j(k) \quad (12)$$

Where N_j is the number of process elements in VM_j. Also, let $\bar{X}_i(AC)$ be the average available capacities of all VM which have related to scheduler S_i. It's defined as:

$$\bar{X}_i(AC) = \sum_{j=1}^{N} AC_j \Big/ N \quad (13)$$

Learning automaton LA_i^{job} chooses one of actions as per its updated probability vector. Let action $\alpha_i^{VM}(j,k,l)$ be selected, then scheduler S_i checks to see if AC_j is greater than $\bar{X}_i(AC)$. If so, selected action $\alpha_i^{VM}(j,k,l)$ gets rewards and increases its choice probability by Eq.1. Otherwise selected action $\alpha_i^{VM}(j,k,l)$ gets penalty and decreases its choice probability by Eq.2. Pseudo code of second phase is shown in Figure 5.

```
Input: $J_i^{sel}$ And $RU_i$
Output: Allocate $J_i^{sel}$
01:  $LA_i^{VM}$.Initial(RUi)
02:  For each $\alpha_i^{VM}(j,k,l) \in$ action-set($LA_i^{VM}$) do
03:      If $RC_i^{sel}(l) \le AC_j(k)$ then
04:          $LA_i^{VM}$.Remove($\alpha_i^{VM}(j,k,l)$)
05:      End If
06:  End For
07:  $\alpha_i^{VM}(j,k,l) \leftarrow LA_i^{VM}$. Select-Action($P_i^{VM}$)
08:  If $AC_j > \overline{X}_i(AC)$ then
09:      update $LA_i^{VM}$ by Eq.1 //reward
10:  Else
11:      update $LA_i^{VM}$ by Eq.2 //penalty
12:  End If
13:  $T_i^{sel}(l)$.Allocate( $PE_j(k)$ )
14:  For each $VM_i \subseteq VM$ do
15:      $VM_i$.Send("retry")
16:  End For
17:  $LA_i^{VM}$.Restore-Actions()
```

Fig. 5: Pseudo code of one round of second phase of TPALA algorithm

In any case, scheduler S_i allocate process element $PE_j(k)$ to task $T_i^{sel}(l)$ and send a "retry" signal for any other VMs. Finally, at the finish of each allocation, all deleted actions must be added as explained in sec 3. By using this method, the submitted workloads will be distributed on different VMS based on its computational capacity and it is minimized the average running time of user's jobs.

5- Simulation and Experimental Results

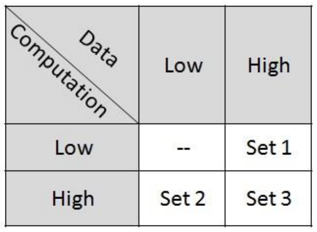
In this section, we describe CloudSim Toolkit, our simulation parameters and experimental results of comparing our proposed algorithm with three algorithms BatCL, FUGE, and HPSO on several evaluation metrics. The reason for choosing these algorithms to compare with our algorithm is that FUGE is a hybrid method based on fuzzy logic and genetic algorithms, HPSO is a metaheuristic algorithm, and BatCL is a cellular automata scheduling method. Therefore, these algorithms were selected from various algorithm categories.

5-1- CloudSim Toolkit

In this paper, the CloudSim Simulator was used to modeling and evaluation our proposed algorithm, because CloudSim simulator is top of simulator tools for cloud computing, that was developed in the Department of Software Engineering and Computer Science at the University of Melbourne [44]. This simulator is used in many industries and famous universities in the world to simulate cloud-based algorithms. Main limited of CloudSim is the lack of proper graphical user interface (GUI). CloudSim architecture has four layers which at now SimJava and GridSim layers are combined and it has changed to three layers architecture. In this version of CloudSim, uses SimJava as distinct event simulation engine that provides several services and process like: event and queuing processing, progression of cloud system elements e.g. hosts, datacenters, brokers and virtual machines[45].

5-2- Simulation Parameters

To simulate our proposed method and compare it with several others scheduling algorithms, we have used different structures and resources distributions in our cloud model which is implemented in the CloudSim simulator environment. We use five different scenarios with different numbers of jobs that in the smallest case we will have 100 input jobs and in the largest case we will have 500 input jobs. In addition, in each of these cases, for a more accurate comparison, jobs are based on two factors: data volume and computational volume. According to Figure 6, generated jobs are divided into three different categories: Set1: jobs with high data volume and low computational volume. Set2: jobs with low data volume and high computational volume. Set3: jobs which have a high data and computational volume.



Data / Computation	Low	High
Low	--	Set 1
High	Set 2	Set 3

Fig. 6: Three different categories for generated jobs

It should be noted that there is a fourth case in which works have a low data volume and low computational volume, which we have not considered in our simulations because in general in small cases (whether in terms of number The jobs and in terms of data volume and computational volume) performance of most scheduling algorithms are the same and are not worth examining and cannot be used as an acceptable criterion. An example of a combination of these three sets of jobs used with their specifications is given in Table 3. For our simulation, we randomly generated the required number of cases in each case from this set of jobs.

Table 3: Typical jobs characteristics

Type	**Job ID**	**Length (MI)**	**Num CPUs**	**File Size**	**Output Size**
Set 1	0	5000	1	6000	500
Set 1	1	10000	1	8000	550
Set 1	2	20000	1	12000	650
Set 1	3	25000	1	14000	700
Set 2	5	60000	2	600	100
Set 2	5	65000	2	700	120
Set 2	6	75000	3	900	160
Set 2	7	80000	3	1000	180
Set 3	8	25000	1	4000	150
Set 3	9	30000	1	4500	160
Set 3	10	35000	2	5000	170
Set 3	11	45000	2	6000	190

In the simulation, three datacenters each with several hosts were used. The characteristics of datacenters and hosts are listed in Table 4 and Table 5 respectively. VMM and OS of all datacenters are Xen and Linux respectively. It is important to note that another very important class used in the CloudSim simulator is the Processing Element (PE), which is related to the hosts and this class represents the processing units or CPUs. This feature is expressed by the millions instructions per second (MIPS) factor and it's listed in the hosts characteristics table.

Table 4: Datacenters characteristics

DC_ID	DC_Name	Architect	Cost per Memory	Cost per Storage
0	DataCenter_0	x64	0.05	0.001
1	DataCenter_1	x64	0.06	0.0015
2	DataCenter_2	x64	0.04	0.002

Where DS is datacenter and BW is bandwidth. Each host has 10 to 20 virtual machines that typical VMs characteristics were shown in table 6.

Table 5: Hosts characteristics

Host ID	DC_Name	MIPS	RAM (MB)	Storage (GB)	BW (Mbps)
0	DataCenter_0	6200	2048	500	500
1	DataCenter_0	7500	1024	1000	500
2	DataCenter_0	8000	4096	1000	500
3	DataCenter_1	4200	4096	500	500
4	DataCenter_1	5000	8192	1500	500
5	DataCenter_1	12100	8192	1500	500
6	DataCenter_2	7100	2048	1500	500
7	DataCenter_2	9495	2048	1500	500
8	DataCenter_2	8500	8192	1500	500
9	DataCenter_2	11900	2048	1500	500
10	DataCenter_2	12100	4096	1000	500

Table 6: Typical VMs characteristics

VM_ID	MIPS	Num CPUs	RAM (MB)	BW (Mbps)	Size
0	600	1	256	50	8000
1	1200	1	1024	50	15000
2	1000	2	512	50	10000
3	800	2	1024	50	20000
4	1200	2	512	50	12000

5-3- Experimental Results

To evaluate and compare the performance of scheduling algorithms, there are various metric that we have used here some of the main and important metric in this field to show the advantage of our proposed algorithm compared to the three algorithms: FUGE [27], HPSO [23] and BatCL[28] that we mentioned in Section 2. In the following, we first describe each of the metrics which used and then express the results of our simulation in the cloud environment using the CloudSim simulator. The four metrics used in this article are: makespan, success rate, average waiting time and degree of imbalance.

5-3-1- Makespan

Makespan is the most common measurement parameter of the optimization methods. This metric is defined as the maximum running time between submitted jobs. In other words, it indicates when the last job was completed. Minimizing this parameter indicates that things are not done in a long time. In our work, makespan is measured in milliseconds. The lower the value of this metric, it means that it has a better scheduling algorithm and the algorithm was able to process the jobs and deliver them to the users sooner. The value of this metric can be calculated based on the Eq.(14) [46]:

$$\text{Makespan} = \max\{ C_i \ , i = 1,2, \dots , n\} \qquad (14)$$

Where, n is number of jobs and C_i is completion time of job *i*th. In Figure 7 shows the average makespan of our algorithm and three other algorithms under different number of jobs. As the number of jobs increases, so does the makespan. When the number of jobs is low, the makespan of all scheduling algorithms is almost close to each other, and as the number of input jobs increases, the difference between the results increases and the TPALA algorithm will have more improvements, in which case the advantage of our proposed method becomes more apparent due to the use of LA. The results show that the TPALA improved the makespan by an average of 4.53% when compared with BatCA and respectively by an average of 10.88% and 19.62% when compared with FUGE and HPSO.

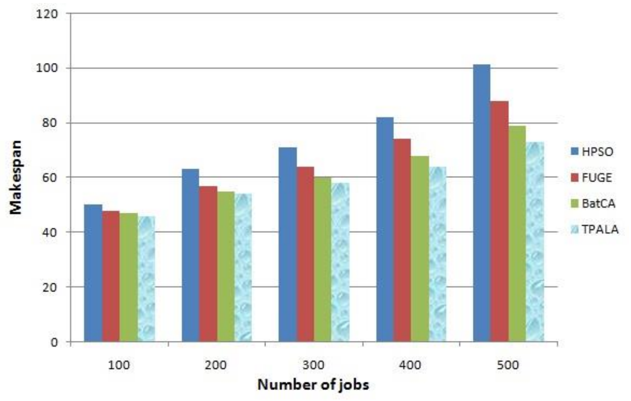


Fig. 7: makespan under number of jobs

5-3-2- Success Rate

Success rate is the fraction or percentage of success among a number of attempts therefore success rate in job scheduling is the ratio of number of successfully completed job to the all number of submitted jobs. So success rate is calculated based on the Eq.(15) [8]:

$$\text{success rate} = \frac{\text{successfully completed job}}{\text{All submitted job}} \qquad (15)$$

The higher value of this metric shows the more successful the scheduling.
In Figure 8 shows the success rate under different numbers of jobs. As can be seen, with increasing the number of jobs, the success rate decreases. Especially in the case of 500 input jobs, there is a more significant reduction of this metric in all scheduling algorithms. The results show that the TPALA has been better performance in large number of jobs than other algorithms and HPSO had the worst performance, So that the success rate of TPALA was on average 20.59% better than HPSO.

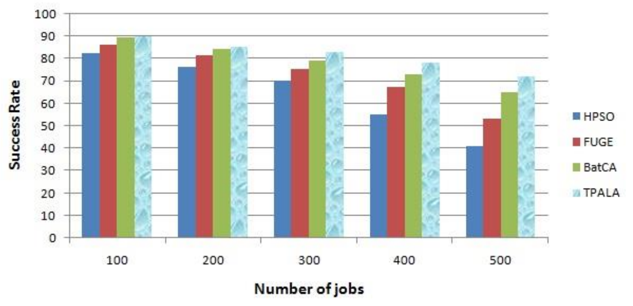


Fig. 8: success rate under number of jobs

5-3-3- Average Waiting time

The waiting time (WT_i) is the amount of time that a process waits in i^{th} run of scheduling algorithm for completion from its submission to completion. The average waiting time is the mean of waiting times in several run of scheduling. One of the goals of the scheduling algorithms is to reduce the waiting time. This metric is calculated based on the Eq.(16) [47]:

$$AWT = \frac{\sum_{i=1}^{n} WT_i}{n} \qquad (16)$$

Where, n is number of jobs and WT_i is waiting time for process i^{th}. In Figure 9 shows the average waiting time under different numbers of jobs. As expected, it shows that in all scheduling algorithms, the average waiting time increases as the number of jobs increases. When the number of jobs is low, the average waiting time of all scheduling algorithms is acceptable and almost close to each other, But as the number of jobs increases, time differences in different algorithms become apparent. The results show that the TPALA has least average waiting time compare to other algorithms and BatCA performance is better than FUGE. The average waiting time of TPALA was on average 5.22% better than BatCA and BatCA was on average 7.25% better than FUGE.

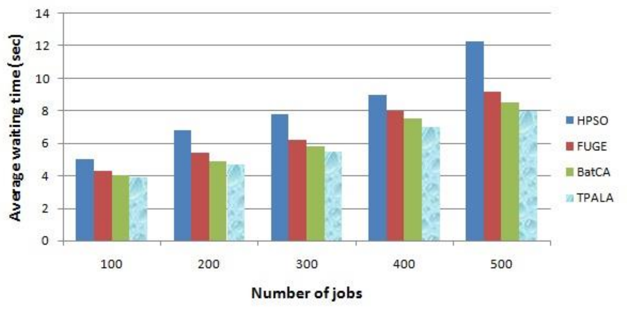


Fig. 9: average waiting time under number of jobs

5-3-4- Degree of Imbalance

Degree of imbalance is the metric for measuring the imbalance among virtual machines. It is a measure that is inversely related to the load balance of the system. If value of this metric was been lower, it shows that the distributed job among the virtual machines is more balanced. Degree of imbalance is computed based on the Eq.(17) [5]:

$$Degree\_Imbalance = \frac{J_{max} - J_{min}}{J_{avg}} \qquad (17)$$

Where, J_{max}, J_{min} and J_{avg} respectively show the maximum, minimum and average J_i between all virtual machines. Also for calculating of J_i, the Eq.(18) is used.

$$J_i = \frac{Length\_jobs}{Num\_PE \times PE\_MIPS} \qquad (18)$$

Where, *Length_jobs* is the total length of jobs which sent to the VM_i, *Num_PE* shows the number of PE and PE_MIPS is the capability of corresponding PE. In Figure 10 shows the average waiting time under different number of jobs. As can be seen, with increasing the number of jobs, the degree of imbalance increases. From the results shown in this figure, it can be seen that the proposed algorithm performed better than other algorithms. The degree of imbalance of TPALA and BatCA was somewhat closer to each other and is clearly better than the HPSO and FUGE, so that the degree of imbalance of TPALA was on average 4.13% better than BatCA algorithm but was on average 22.29% better than HPSO.

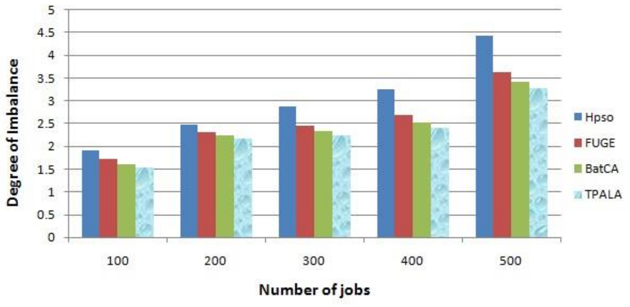


Fig. 10: degree of imbalance under number of jobs

6- Conclusions

In this paper, a new online algorithm based on LA for job scheduling in cloud environment, called TPALA was presented. Our proposed algorithm uses two different LAs for each scheduler to schedule jobs, as there are generally two main challenges in job scheduling. The first challenge is selecting the appropriate job from the submitted jobs based on their priority and specific conditions, while the second challenge is assigning the selected job to the most suitable virtual machine. To address these challenges, our algorithm employs two LAs in separate phases. In first phase a fixed action-set learning automaton was used and in second phase a variable action-set learning automaton was used. To prove the performance of the proposed method, several simulation case based on different scenarios have been simulated by CloudSim toolkit, in which several metric in job scheduling such as: makespan, success rate, average waiting time and degree of imbalance and compared to the three algorithms FUGE, HPSO and BatCL. In contrast to most job scheduling algorithms that use a single job type, we considered a combination of jobs based on two factors in our simulations: data volume and computational volume. In future work, we plan to explore learning automata-based methods for multi-objective job scheduling in cloud computing.

References

[1] B. Varghese, and R. Buyya, “Next generation cloud computing: New trends and research directions,” Future Generation Computer Systems, vol. 79, pp. 849-861, 2018.
[2] N. Moganarangan, R. Babukarthik, S. Bhuvaneswari et al., “A novel algorithm for reducing energy-consumption in cloud computing environment: Web service computing approach,” Journal of King Saud University-Computer and Information Sciences, vol. 28, no. 1, pp. 55-67, 2016.
[3] A. Ghaffari, and A. Mahdavi, “Embedding Virtual Machines in Cloud Computing Based on Big Bang–Big Crunch Algorithm,” Journal of Information Systems and Telecommunication (JIST), vol. 28, no. 7, pp. 305-315, 2020.
[4] L. F. Bittencourt, A. Goldman, E. R. Madeira et al., “Scheduling in distributed systems: A cloud computing perspective,” Computer science review, vol. 30, pp. 31-54, 2018.
[5] N. Mansouri, and M. M. Javidi, “Cost-based job scheduling strategy in cloud computing environments,” Distributed and Parallel Databases, pp. 1-36, 2019.
[6] U. Bhoi, and P. N. Ramanuj, “Enhanced max-min task scheduling algorithm in cloud computing,” International Journal of Application or Innovation in Engineering and Management (IJAIEM), vol. 2, no. 4, pp. 259-264, 2013.
[7] Y. Mao, X. Chen, and X. Li, "Max–min task scheduling algorithm for load balance in cloud computing." pp. 457-465, 2014.
[8] S. A. Hamad, and F. A. Omara, “Genetic-based task scheduling algorithm in cloud computing environment,” International Journal of Advanced Computer Science and Applications, vol. 7, no. 4, pp. 550-556, 2016.
[9] A. Kaleeswaran, V. Ramasamy, and P. Vivekanandan, “Dynamic scheduling of data using genetic algorithm in cloud computing,” International Journal of Advances in Engineering & Technology, vol. 5, no. 2, pp. 327, 2013.
[10] H. Aziza, and S. Krichen, “Bi-objective decision support system for task-scheduling based on genetic algorithm in cloud computing,” Computing, vol. 100, no. 2, pp. 65-91, 2018.
[11] B. Keshanchi, A. Souri, and N. J. Navimipour, “An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification,

simulation, and statistical testing," Journal of Systems and Software, vol. 124, pp. 1-21, 2017.
[12] H. Y. Shishido, J. C. Estrella, C. F. M. Toledo et al., "Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds," Computers & Electrical Engineering, vol. 69, pp. 378-394, 2018.
[13] M. A. Tawfeek, A. El-Sisi, A. E. Keshk et al., "Cloud task scheduling based on ant colony optimization." pp. 64-69, 2013.
[14] C. Z. a. P. W. C. Liu, "A Task Scheduling Algorithm Based on Genetic Algorithm and Ant Colony Optimization in Cloud Computing," in 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, Xi'an, China, 2014, pp. 68-72.
[15] X. Wei, "Task scheduling optimization strategy using improved ant colony optimization algorithm in cloud computing," Journal of Ambient Intelligence and Humanized Computing, 2020/10/21, 2020.
[16] F. Hemasian-Etefagh, and F. Safi-Esfahani, "Dynamic scheduling applying new population grouping of whales meta-heuristic in cloud computing," The Journal of Supercomputing, vol. 75, no. 10, pp. 6386-6450, 2019.
[17] N. Manikandan, N. Gobalakrishnan, and K. Pradeep, "Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment," Computer Communications, vol. 187, pp. 35-44, 2022/04/01/, 2022.
[18] Z. Liu, W. Qu, W. Liu et al., "Resource preprocessing and optimal task scheduling in cloud computing environments," Concurrency and Computation: Practice and Experience, vol. 27, no. 13, pp. 3461-3482, 2015.
[19] V. Priya, and C. N. K. Babu, "Moving average fuzzy resource scheduling for virtualized cloud data services," Computer Standards & Interfaces, vol. 50, pp. 251-257, 2017.
[20] S. Zhan, and H. Huo, "Improved PSO-based task scheduling algorithm in cloud computing," Journal of Information & Computational Science, 2012.
[21] A. Kamalinia, and A. Ghaffari, "Hybrid Task Scheduling Method for Cloud Computing by Genetic and PSO Algorithms," Journal of Information Systems and Telecommunication (JIST), vol. 16, no. 4, pp. 1-10, 2016.
[22] M. Masdari, F. Salehi, M. Jalali et al., "A survey of PSO-based scheduling algorithms in cloud computing," Journal of Network and Systems Management, vol. 25, no. 1, pp. 122-158, 2017.
[23] G. Babu, and K. Krishnasamy, "Task scheduling algorithm based on Hybrid Particle Swarm Optimization in cloud computing environment," Journal of Theoretical and Applied Information Technology, vol. 55, pp. 33-38, 2013.
[24] N. Mansouri, B. M. H. Zade, and M. M. Javidi, "Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory," Computers & Industrial Engineering, vol. 130, pp. 597-633, 2019.
[25] X. Chen, and D. Long, "Task scheduling of cloud computing using integrated particle swarm algorithm and ant colony algorithm," Cluster Computing, vol. 22, no. 2, pp. 2761-2769, 2019/03/01, 2019.
[26] H. Naseri, S. Azizi, and A. Abdollahpouri, "BSFS: A Bidirectional Search Algorithm for Flow Scheduling in Cloud Data Centers," Journal of Information Systems and Telecommunication (JIST), vol. 27, no. 7, pp. 175-183, 2020.
[27] M. Shojafar, S. Javanmardi, S. Abolfazli et al., "FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method," Cluster Computing, vol. 18, no. 2, pp. 829-844, 2015.
[28] Y. Shi, L. Luo, and H. Guang, "Research on Scheduling of Cloud Manufacturing Resources Based on Bat Algorithm and Cellular Automata." pp. 174-177, 2019.
[29] M. I. Khaleel, "Efficient job scheduling paradigm based on hybrid sparrow search algorithm and differential evolution optimization for heterogeneous cloud computing platforms," Internet of Things, vol. 22, pp. 100697, 2023/07/01/, 2023.
[30] H. G. S. Phani Praveen, Negar Shahabi, Fatemeh Izanloo, "A Hybrid Gravitational Emulation Local Search-Based Algorithm for Task Scheduling in Cloud Computing," Mathematical Problems in Engineering, 2023.
[31] S. Sahoo, B. Sahoo, and A. K. Turuk, "An Energy-Efficient Scheduling Framework for Cloud Using Learning Automata." pp. 1-5, 2018.
[32] A. Yazidi, I. Hassan, H. L. Hammer et al., "Achieving Fair Load Balancing by Invoking a Learning Automata-Based Two-Time-Scale Separation Paradigm," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 8, pp. 3444-3457, 2021.
[33] L. Zhu, K. Huang, Y. Hu et al., "A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata," IEEE Access, vol. 9, pp. 81236-81252, 2021.
[34] S. A. Murad, A. J. M. Muzahid, Z. R. M. Azmi et al., "A review on job scheduling technique in cloud computing and priority rule based intelligent framework," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 6, Part A, pp. 2309-2331, 2022/06/01/, 2022.
[35] M. Masdari, and M. Zangakani, "Efficient task and workflow scheduling in inter-cloud environments: challenges and opportunities," The Journal of Supercomputing, vol. 76, no. 1, pp. 499-535, 2020.
[36] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," Future Generation Computer Systems, vol. 91, pp. 407-415, 2019.
[37] M. A. Rodriguez, and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments," Concurrency and Computation: Practice and Experience, vol. 29, no. 8, pp. e4041, 2017.
[38] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," The Journal of Supercomputing, vol. 71, no. 9, pp. 3373-3418, 2015.
[39] J. Kazemi Kordestani, M. Razapoor Mirsaleh, A. Rezvanian et al., "An Introduction to Learning Automata and Optimization," Advances in Learning Automata and Intelligent Optimization, J. Kazemi Kordestani, M. R. Mirsaleh, A. Rezvanian et al., eds., pp. 1-50, Cham: Springer International Publishing, 2021.
[40] K. S. Narendra, and M. A. Thathachar, "Learning automata-a survey," IEEE Transactions on systems, man, and cybernetics, no. 4, pp. 323-334, 1974.
[41] K. S. Narendra, and M. A. Thathachar, Learning automata: an introduction: Courier corporation, 2012.
[42] A. Rezvanian, A. M. Saghiri, S. M. Vahidipour et al., Recent advances in learning automata: Springer, 2018.

[43] M. A. L. Thathachar, and B. R. Harita, “Learning automata with changing number of actions,” IEEE Transactions on Systems, Man, and Cybernetics, vol. 17, no. 6, pp. 1095-1100, 1987.
[44] R. N. Calheiros, R. Ranjan, C. A. De Rose et al., “Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services,” arXiv preprint arXiv:0903.2525, 2009.
[45] R. N. Calheiros, R. Ranjan, A. Beloglazov et al., “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software: Practice and experience, vol. 41, no. 1, pp. 23-50, 2011.
[46] R. D. Lakshmi, and N. Srinivasu, “A dynamic approach to task scheduling in cloud computing using genetic algorithm,” Journal of Theoretical & Applied Information Technology, vol. 85, no. 2, 2016.
[47] J. Blazewicz, K. H. Ecker, E. Pesch et al., Scheduling Computer and Manufacturing Processes: Springer Science & Business Media, 2013.