Computing and Informatics, Vol. 42, 2023, 37–74, doi: 10.31577/cai_2023_1_37

COST-EFFECTIVE SCHEDULING AND LOAD BALANCING ALGORITHMS IN CLOUD COMPUTING USING LEARNING AUTOMATA

Ali Sarhadi, Javad Akbari Torkestani*

Department of Computer Engineering
Arak Branch, Islamic Azad University
Arak, Iran
e-mail: {asarhadi, j-akbari}@iau-arak.ac.ir

Abstract. Cloud computing is a distributed computing model in which access is based on demand. A cloud computing environment includes a wide variety of resource suppliers and consumers. Hence, efficient and effective methods for task scheduling and load balancing are required. This paper presents a new approach to task scheduling and load balancing in the cloud computing environment with an emphasis on the cost-efficiency of task execution through resources. The proposed algorithms are based on the fair distribution of jobs between machines, which will prevent the unconventional increase in the price of a machine and the unemployment of other machines. The two parameters Total Cost and Final Cost are designed to achieve the mentioned goal. Applying these two parameters will create a fair basis for job scheduling and load balancing. To implement the proposed approach, learning automata are used as an effective and efficient technique in reinforcement learning. Finally, to show the effectiveness of the proposed algorithms we conducted simulations using CloudSim toolkit and compared proposed algorithms with other existing algorithms like BCO, PES, CJS, PPO and MCT. The proposed algorithms can balance the Final Cost and Total Cost of machines. Also, the proposed algorithms outperform best existing algorithms in terms of efficiency and imbalance degree.

Keywords: Cloud computing, load balancing, learning automata, efficiency

* Corresponding author

1 INTRODUCTION

Cloud computing, the new form of on demand computing gains its popularity in the last few years [1]. Cloud computing is used to provide the calculation platform for Internet users as a large-scale distributed computing environment. The cloud computing is usually in ultra large scale and high scalability. To be more specific, cloud computing can be linked with a large number of idle resources and constitute a large scale resource pool [2]. Cloud computing offers services with minimum cost compared to the setting up a datacenter. Cloud Service Providers (CSPs) permits the users to select the machine hours based on their requirement regardless of the costs without paying a premium for large scale. As cloud environment is evolving day by day and confronted with various issues, one of them being uncovered is scheduling. Job scheduling methods is one of the most challenging hypothetical problems in the Cloud computing environment. Grid and cloud environment have the same primary goals such as decreasing computing cost, enhancing flexibility and reliability by converting systems from something that we purchase and process our-selves to something that is worked by a third party. The existing scheduling methods are according to the factors like service response time, resource utilization, cost and performance [3]. There are various factors that can be mentioned as the parameter of allocation issue like system load balance, throughput, reliability, cost of service, etc. Today, low cost and appropriate resource utilization of job scheduling problem has persuaded innovators to propose different cost related job scheduling and load balancing strategies [4].

Load balancing is playing a main role in maintaining the organization of Cloud computing. The main goal of load balancing mechanism is to map the jobs which are set forth to the cloud domain to the unoccupied resources so that the overall available response time is improved, and it also provides efficient resource utilization [5, 6].

Therefore, providing the efficient load-balancing algorithms and mechanisms is a key to the success of cloud computing environments. On the other hand, developing the economic-based resource load balancing and scheduling system is a very vital issue that has not been addressed properly. So, this work presents an implementation economic aspect of resource load balancing strategies with simulated result in CloudSim3.0 software by considering the economic parameters. New learning automata based resource load balancing algorithms solves the problem posed by imbalanced loads and the subsequent increase in the price of one resource and the idleness of other resources by defining new economic parameters.

1.1 Proposed Approach

Learning Automata are adaptive decision-making devices operating on unknown random environments. They have proved capable of solving NP-complete problems which need extensive search in solution space. Therefore, using this tool can increase the efficiency of the proposed algorithms. The obtained results indicate the improvement of the proposed algorithms in comparison with the existing ones. In this paper, our main concern is to address the consequences imposed by the unfair distribution of tasks between resources that lead to an excessive increase in the price of a resource in the computational networks such as cloud computing, using an enhanced learning scheme known as Learning Automata.

Therefore, a learning automata based approach is designed to distribute tasks between resources to increase resource efficiency and to prevent the price increase of a resource. By defining two economic parameters Total Cost (TC) and Final Cost (FC), we will try to solve economic-based load balancing problem using a learning automata.

1.2 Contributions

The main contributions of our paper can be expressed as follows:

- We proposed a set of learning automata based algorithms for cloud computing load balancing and scheduling problem.
- We designed a framework for economic task scheduling and resource management by defining Total Cost (TC) and Final Cost (FC) parameters.
- We conducted a series of experiments to evaluate the performance of the proposed algorithms under the defined dataset.

1.3 Organization of the Paper

The paper is organized as follows: In Section 2, the related works are discussed. We introduce research motivation and challenges in Section 3. Techniques, concepts, general proposed approach are introduced in Section 4. The automata model for scheduling and load balancing is provided in Section 5. The proposed algorithms are discussed in Section 6. Simulation and experimental results are introduced in Section 7. Finally, there is our conclusion in Section 8.

2 RELATED WORKS

At present, traditional scheduling algorithms, list scheduling algorithms and metaheuristic scheduling algorithms for cloud workflow task scheduling are available. Traditional scheduling algorithms, such as round robin scheduling algorithm, MIN-MIN algorithm and MIN-MAX algorithm, have the advantages of simple implementation and low algorithm complexity, but they can only be applied to specific scenarios. A literature task scheduling survey based on three different perspectives (methods, applications, and parameter-based measures utilized) is organized in [7]. MCT heuristic technique is used in both the static and dynamic (online mode) load balancing strategy. [8] have used MCT technique where they considered both ready-to-execute time and the expected execution time of the tasks for balancing purposes. They allocate the task to the core that has the least completion time. The MCT will perform after allocation of task to a machine for the selection of the appropriate core.

Meta-heuristic techniques proved to be very capable solving scheduling problems. From a cloud perspective a brief on traditional and heuristic scheduling methods is firstly provided, and only secondly it dives deeply into the most popular meta-heuristics for the cloud task scheduling [9]. [10] introduces an improved particle swarm optimization algorithm (IPSO). [11] presents a novel hybrid antlion optimization algorithm with elite-based differential evolution for solving multi-objective task scheduling problems in cloud computing environments.

The proposed task scheduling algorithm in [12] is based on the gray wolf optimizer – a nature-inspired algorithm. A new method of initial optimization on the crossover mutation probability of adaptive genetic algorithm, is introduced in [13, 14].

Load balancing is another aspect of scheduling services which is the process of distributing workloads and computing resources in a cloud computing environment. So we need to introduce some of the most important study in this area.

[15] proposed a complete survey of cloud computing load balancing algorithms. This paper presents a comprehensive and comparative study of various load balancing algorithms. Load balancing principles are introduced to many forms of constraints and environment settings [16, 17]. Perfect surveys of the published load balancing algorithms achieved by server consolidation via a meta-analysis is introduced in [18]. [19] presents a state-of-the-art (SOTA) review of issues and challenges associated with the existing load balancing techniques for researchers to develop more effective algorithms. [20] proposes hybrid metaheuristics technique which combines the osmotic behavior with bio inspired load balancing algorithms. [21] proposes an efficient binary version of PSO algorithm with low time complexity and low cost for scheduling and balancing tasks in cloud computing. In [22], a massive study on the scheduling and load balancing algorithms of cloud computing was proposed with the objective to minimize the performance and the makespan.

2.1 Economic-Based Scheduling and Load Balancing

On the other hand, economic concepts play a key role in cloud computing environment. In distributed computing with a lot of different participants and contradicting requirements, the well-known efficient approaches are based on economic principles [23]. Here, we will mainly discuss the economic-based cloud resource load balancing and task scheduling strategy. We have focused on economic aspect of cloud computing scheduling and load balancing mechanism, so we need to introduce a brief overview.

To address these issues, the economic-based distributed resource management and scheduling has become a hot point of research for domestic and foreign scholars, and there is a great deal of research results [24]. Buyya proposed distributed computational economy-based framework called the Grid Architecture for Computational Economy (GRACE) [25]. This economic-based framework offers an incentive to resource owners for contributing and sharing resources. [5] addressed the idea of applying economic models to the task scheduling. The efficiency of this approach in terms of response and wait time minimization as well as utilization is evaluated. [25] developed three heuristic scheduling algorithms for cost, time, and time-variant optimization strategies that support deadline and budget constraints.

For cost-critical parallel applications, the cost-aware scheduling algorithms have been proposed for minimizing execution cost or satisfying the budget constraint on heterogeneous systems in [26]. [27] presents a new opportunistic scheduling and resource consolidation system based on an economic model related to different service level agreements (SLAs) classes. [28] proposes a new load balancing and scheduling technique, Hybrid Genetic Gravitational Search Algorithm (HG-GSA) for reducing the total cost of computation in this paper. An adaptive fitness function is used that takes into account both the cost and the makespan. The main objective of [29] is to propose a Completion Time Driven Hyper-Heuristic (CTDHH) approach for cost optimization of SWFS in a cloud environment. fundamental research issue addressed in [30] is the potential trade-off between the makespan and the cost of virtual machine usage. This paper proposes a HEFT-ACO approach, which is based on the heterogeneous earliest end time (HEFT), and the ant colony algorithm (ACO) to minimize them. [31] proposed a critical-greedy (CG) algorithm to minimize the end-to-end delay of budget constrained parallel applications. [32] discussed task scheduling and resource allocation problem for implementing tasks in clouds; a novel provisioning and scheduling algorithm is presented to execute tasks under budget constraint while reducing the slowdown. [33] presented HCOC (the Hybrid Cloud Optimized Cost) scheduling algorithm. HCOC decides which resources should be leased from the public cloud and aggregated to the private cloud to provide sufficient processing power to execute a workflow within a given execution time. One of the most important studies from the economic aspect of scheduling was done by Buyya and it is known as BCO algorithm, which is collected in [25]. In BCO (Buyya Cost Optimization) algorithm a designation queue is allocated to each resource. The user's requests are set in the designation queue in an ascending order of time. [34] suggested a new eonomic based scheduling strategy known as Cost-based Job Scheduling (CJS). The CJS algorithm uses data, processing power and network characteristics in job allocation process. The CJS strategy computes three important costs: the cost of network, cost of computation and data transfer cost.

Preference-based Economic Scheduling (PES) algorithm is proposed for distributed computing with regard to preferences given by various groups of virtual organization (VO) stakeholders (such as users, resource owners and administrators) to improve the overall quality of service and resource load efficiency. In this paper, a problem of finding a balance between VO stakeholders' preferences to provide fair resource sharing and distribution is studied [35]. Our approach is to apply a new method to cloud computing load balancing based on the economic criteria using learning automata.

2.2 Learning Automata Based Scheduling and Load Balancing

Learning Automata are adaptive decision-making devices operating on unknown random environments. Learning Automata has a finite set of actions and each action has a certain probability (unknown to the automata) of getting rewarded by the environment of the automata. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction on the system. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can result in selection of the optimal action. Learning Automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA). 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 [36, 37]. Learning automata have a perfect adaptability to environmental changes. This feature is very suitable for use in distributed computational environments such as cloud environment with a high degree of dynamism [38].

A task scheduling service that adopts cost optimization strategy should map heterogeneous grid resources for heterogeneous user applications so that their execution finishes in the specified deadline with minimum cost proposed in [39]. [40] proposed new algorithms based on learning automata. For this purpose a set of learning automata-based algorithms are proposed to solve resource scheduling and load balancing with economic parameter and fair distribution of tasks over resources in grid computing environment.

Our study typically focuses on resource load balancing and scheduling based on learning automata, especially with economic concepts. However, the literature review that reports the extensive use of learning automata to typically achieve these specific goals has been limited.

[41] proposed a novel learning automata-based scheduling framework for deadline sensitive tasks in the cloud. In [42] authors proposed a self-adapting task scheduling algorithm (ADATSA) using learning automata to solve these problems. [43] designed a model for task offloading using learning automata based decision making algorithm (LADMA). This algorithm considers the completion time and energy consumption of the tasks during the allocation of the tasks to the suitable VMs in the cloud.

A learning automata model for the task scheduling in distributed environments was introduced by [44]. The mechanism introduced by the authors of this paper acts absolutely without a prior information about the task but rather adapts to the changing loads of the servers. To minimize the response time, a heuristic approach based on the model of a learning automata was introduced by [45]. In this paper, depending on the status of the current load distribution, a new task would be scheduled to be executed either locally or on some other machine. [46] introduced a scheme based on learning automata that is capable to solve some of the problems on various cloud applications.

A cost attentive Reward Minimum-Penalty approach was introduced in [47]. In the doctoral thesis of Meybodi [48] a threshold was introduced, which was the average response time taken over both streams, and the response time of a served dispatched request from the chosen stream by the learning automata was compared with that threshold for the inferred learning automata response. But in the field of applying learning automata in economic scheduling and load balancing, scattered and partial work has been done. Authors in this paper try to present a comprehensive study in this field. In addition to the work mentioned above, the learning automata and its hybrid models can be considered as a suitable model for solving the above problem due to the following features:

1. The learning automata are able to perfectly adapt themselves to environmental changes. This feature is very suitable for use in Cloud environments with a high degree of dynamism [49].
2. 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 [36, 50, 51].
3. Interacting with each other, the learning automata are able to perfectly model the distribution of Cloud 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 adaptability to the environment [52, 53, 54, 55].
4. Interacting with each other, the learning automata are able to converge to the global optimal answer based only on the local decisions when solving optimization problems. Therefore, learning automata-based algorithms can be considered as an appropriate choice for the Cloud as they can resolve the slag resulted from aggregation or dissemination of information in centralized algorithms [56, 57, 58].
5. 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, the tolerance of learning automata-based algorithms, in case of the occurrence of possible errors, will not affect the algorithm's performance like the other algorithms [53].

Also, other learning techniques such as reinforcement learning and deep learning have also been used in cloud computing scheduling and load balancing problem.

Reinforcement learning (RL) is broadly used to solve problems in partially visible environments. As the most recent state-of-the-art (SOTA) reinforcement learning (RL) algorithms, they have many advantages in solving problems. Asynchronous Advantage Actor-Critic (A3C) [59], Proximal Policy Optimization (PPO) [60] and Soft Actor-Critic (SAC) [61] are the most important models of SOTA algorithm. Those models are already being used to solve various problems, such as robot navigation, elevator controls, military applications, medical diagnostics, task scheduling, and education [62].

Proximal Policy Optimization (PPO) is one of the SOTA models and a type of reinforcement learning algorithm. The PPO algorithm is gradient policy algorithm that is suitable for continuous control problems. PPO is an offline learning method, and its strategy to interact with the environment and the strategy to be learned differ. The main idea of the PPO algorithm is to transfer online learning to offline learning based on importance sampling and adopt two networks to improve the network convergence rate [63].

2.3 Multi Objective Scheduling Strategy

Simultaneous time and cost optimization is another aspect of the reported studies of resource scheduling and load balancing. [64] presented a hybrid strategy named FUGE on the base of fuzzy approach and a genetic method that wants to reduce the execution time and costs. [65] proposed a job scheduling strategy by heuristic search approaches within cloud computing system where time and cost optimization is the main object of this paper. [66] optimized the job scheduling approach on the base of biogeography-based optimization (BBO). BBO strategy provides the advantageous of adaptive strategy that is proposed to solve the challenge of binary integer job scheduling in cloud environment. In [67, 68] authors proposed an algorithm for task scheduling based on multiple criteria and multiple decision to choose a task to be executed in a particular VM, Multiple criteria include the various QoS parameters. This algorithm helps to reduce the makespan of the system. In [69] authors proposed an algorithm based on NSGA-II for load balancing of CPU, memory and bandwidth in cloud computing and [70] author uses the combination of genetic algorithm along with fuzzy optimization theory. In [71, 72] authors implement the modified ant colony optimization to minimize the execution time and cost by considering the execution time, arrival time and other QoS parameters as a criteria for searching the best VM for the execution of tasks such that the make span of the system is reduced.

3 RESEARCH MOTIVATION AND CHALLENGES

Although, cloud is equipped with high performance elements, it is found that the lack of resource load balancing and efficient job scheduling controlled it from working in its full capacity. Hence, in this study we have given a great emphasis to the job scheduling and load balancing in the cloud right after the identification of high-level problems such as economic aspect of load balancing and fair distribution of task over resources. Obviously, attempting to solve the scheduling and load balancing problem in the cloud and delivering an enhanced load balancing mechanism as a result does not only increase the performance of the cloud but also the pleasure of users who are the main factors in the cloud environment.

Load balancing concerns the distribution of resources among users or it requests, in a uniform manner, so that no node is overloaded or sitting idle. Like in all other internet based distributed computing, load balancing is an important aspect in cloud computing [73]. In the absence of load balancing provision, efficiency of some overloaded nodes can sharply degrade at times, leading to violation of SLA [74].

On the other hand, cloud computing approach, ideas and strategy need a new field for researching in economic aspect of resource load balancing. Following Cost-effective load balancing strategies, this paper proposes a new cloud resource load balancing and scheduling algorithms, which can not only increase the usage of resources or system utilization, but also, by trying to import new parameters, it can satisfy both the resource providers and consumers. This satisfaction will be achieved when both of them receive economic benefits. In fact, in this paper we propose the set of cost-efficient and fair cloud computing load balancing algorithms which try to balance different economic parameters.

4 TECHNIQUES, CONCEPTS, GENERAL PROPOSED APPROACH

LA theory is appropriate for the environment which is dynamic, complex, and there is a large number of uncertainties like cloud environment, computer networks. [51] presented a survey in the area of learning automata. Their study mainly focused on norms and behavior of learning automata, reinforcement learning schemes, the convergence of learning algorithm, the interaction of several automata. Variable Structure Learning Automata (VSLA) is a quintuple $(\alpha, \beta, p, T(\alpha, \beta, p))$, where α is the action set of automata, β is an environment response set, p is the probability vector, each being the probability of performing every action in the current internal automaton state, and the function of T is the reinforcement algorithm. If the response of the environment takes binary values learning automata model is P-model, and if it takes finite output set with more than two elements that take values in the interval $[0, 1]$ such a model is referred to as Q-model, and when the output of the environment is a continuous variable in the interval $[0, 1]$, it is referred to as S-model. The function of T is the reinforcement algorithm, which modifies the action probability vector P with respect to the performed action and received response. Assume $\beta \in [0, 1]$. A general linear schema for updating action probabilities can be represented as Equation (1) and Equation (2). Let action i be performed then:

$$P_j(n+1) = P_j(n) + \beta(n)[b/(r-1) - bp_j(n)] - [1-\beta(n)]\alpha p_j(n), \tag{1}$$

$$P_i(n+1) = P_i(n) - \beta(n)bp_i(n) + [1-\beta(n)]\alpha[1-P_i(n)], \tag{2}$$

where a and b are reward and penalty parameters. When $a = b$, the automaton is called L_{RP}, if $b = 0$ the automaton is called L_{RI} and if $0 < b \lll a < 1$, the automaton is called L_{ReP}. For more information about learning automata the reader may refer to [52, 75, 76, 77].

Load balancing is the process of improving the performance of a parallel and distributed system through a Reassign of load among the VMs. It can also generally be described as anything from distributing computation and communication evenly among processors, or a system that divides many client requests among several servers.

To analyze and implementation the proposed learning automata based load balancing and scheduling approach and before introducing proposed model, the task execution cost must be estimated on each machine. The estimate is stored in an $M*N$ matrix, named ECC (Expected Cost to Compute). This matrix can be changed to obtain different ranges of computing environments varying price or resource value. To generate this matrix, an $M*1$ basic column vector, named B, is first made of floating-point values. The upper bound of possible values in the basic vector is shown by ω_b. The basic column vector is generated through the frequent repetition of a uniform random number when $x_b^i \in [1, \omega_b)$. Then $B(i) = x_b^i$ is defined for $1 \leq i \leq M$. After that, ECC rows are generated. To obtain each element $ECC(t_i, m_j)$ on the i^{th} row of ECC, the basic value $B(i)$ is multiplied by a uniform random number $x_r^{i,j}$ $(j \leq 1 \leq N)$, which is a random number ranging in $[1, \omega_b)$. It is called the row multiplier. Each row requires N row multipliers. Each row of ECC can be described as $ECC(t_j, m_j) = B(i) \times x_r^{i,j}$, the basic column does not appear in the final ECC. This process is repeated for each row until the ECC is filled with $M*N$ elements. Therefore, all of the ECC elements range in $[1, \omega_b * \omega_r)$. Now we will explain how to use the ECC matrix in the proposed model based on the learning automata.

In this paper a general maping function, $Q(i) = j$ is defined from the domain of tasks $i = 1, \dots, M$ to the domain of machines $j = 1, \dots, N$ and its general procedure is shown in Figure 1. In this paper the required cost of executing all the tasks assigned to a machine in the n^{th} iteration is called the Total Cost (TC) of that machine, indicated by $C^{(n)}(j)$. It is determined through the Equation (3):

$$c^{(n)}(j) = \sum ECC(k, j), \quad j = Q(k), 1 \leq k \leq M. \tag{3}$$

The maximum value of $c^{(n)}(j)$ on $1 \leq j \leq N$, in the n^{th} iteration is called the Final Cost (FC) and it is shown by $F^{(n)}$. In the proposed model, a Learning Automata was assigned to each task T_i. The tasks can be assigned to each of N machines, Automata share the same actions. Hence, there can be $\alpha(i) = m_1, m_2, \dots, m_{n-1}$ and $0 \leq \beta(i) \leq 1$ or each task T_i $(1 \leq i \leq M)$. The closer $\beta(i)$ gets to zero, the more satisfactory the action of automata i gets, however the action will be dissatisfactory if $\beta(i)$ gets closer to 1 according to Figure 1.

5 AUTOMATA MODEL FOR SCHEDULING AND LOAD BALANCING

This section presents a model of the computational cloud studied for scheduling and load balancing algorithms. Figure 2 shows the schematic representation of the

```
While termination-conditions are not met do
Begin
        $n = n+1$ //n represents the iteration#
        For each $A(i)$ do
            $F^{(n)}(i)$= A(i).Select-Action()
        For each $A(i)$ do
            Evaluate $\beta^{(n)}(i)$ according to mapping algorithm
          For each $A(i)$ do
            $A(i)$.Update($\beta^{(n)}(i)$)
  End
```

Figure 1. General proposed procedure used by Learning Automata mapping algorithms

environment. The environment consists of the virtual machines (VM) which will be used to execute the application. The scheduling and load balancing system consists of the automata, and the model of the application and the virtual machines.

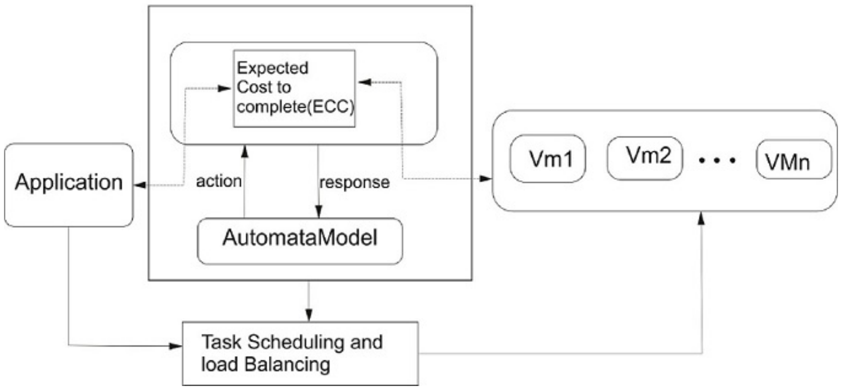
Expected Cost to complete(ECC)
Application
action
response
AutomataModel
Vm1
Vm2
VMn
Task Scheduling and load Balancing

Figure 2. Model of the cloud computing

The learning automata model, as shown in Figure 3, is constructed by associating every task s_i, i with a variable structure learning automaton, which is represented by a 3-tuple. Each action of an automaton is associated with a virtual machine, and since the tasks can be assigned to any of the N virtual machines (N: number of virtual machines), the action set of all learning automata is identical. Therefore, for any task s_i, $1 \leq i \leq M$ (M: number of tasks), $\alpha(i) = m_1, m_2, \ldots, m_N$ (m_i is the i^{th} virtual machine) and $\beta(i) \in [0, 1]$. Environment may be interpreted as a P-model, Q-model, or S-model.

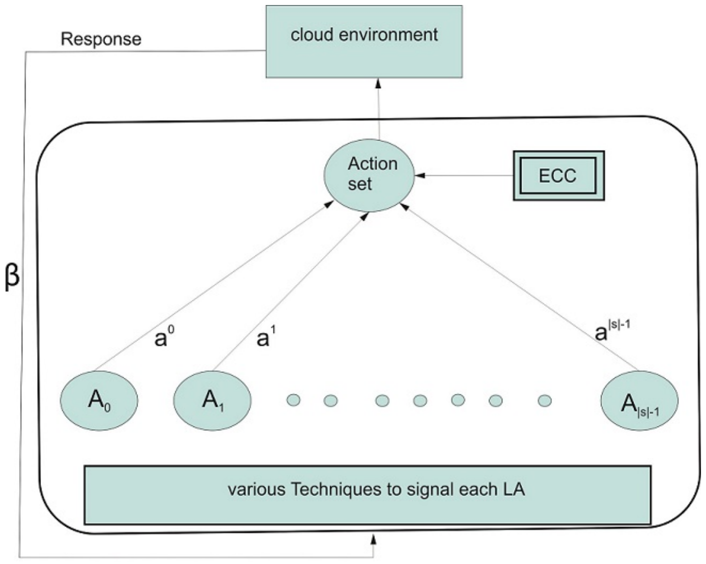

Figure 3. Learning Automata model for scheduling and load balancing algorithm

6 PROPOSED ALGORITHMS

This section introduces three category of learning automata based algorithms, emphasizing FC and TC parameters.

6.1 Cost-Efficient Load-Balancing Profit-Based Algorithms

The innovative feature behind Profit-based algorithms is to simultaneously decrease Final Cost as well as reduce Total Cost of the selected machine in comparison to previous iterations. In other words, the highest reward is given to an automaton when both Final Cost and Total Cost of selected machine are less than their values in previous iteration.

6.1.1 The Profit-Based Algorithm with Specified Rewarding (PS)

Profit-based algorithm with specified rewarding(PS) evaluates goodness of an automaton action considering the Final Cost and Total Cost of machine associated with the action. This algorithm describes the environment as the Q-Model. In the n^{th} iteration, the Final Cost might be greater than, smaller than, or equal to the Final Cost in the $(n-1)^{\text{th}}$ iteration. Likewise, the Total Cost of a selected machine by automata $A(i)$ in the n^{th} iteration might be greater than, smaller than, or equal to the Total Cost of a selected machine in the $(n-1)^{\text{th}}$ iteration. Therefore, given the Final Cost and Total Cost of selected machines in two consecutive iterations, 9 possible cases can occur for the determination of $\beta^{(n)}(i)$. One value is allocated to each of these 9 cases. This value shows the goodness of the action taken by the automata. The environmental response to automata $A(i)$ in the n^{th} iteration is determined in the following way: This value indicates how appropriate is the automaton's action. Equation (4) is employed to determine the environmental response to automata $A(i)$ in the n^{th} iteration:

$$\beta^{(n)}(i) = 1 - (f(F^{(n-1)}, F^n)\beta_F + f(C^{(n-1)}(Q^{(n-1)}(i), C^n(Q^n(i)))\beta_C). \tag{4}$$

In this equation, $\beta_F + \beta_C = 1$, and β_F shows the received amount of reward if the Final Cost is smaller than that of the previous iteration. Moreover, β_C shows the received amount of reward if the Total Cost of the selected machine is smaller than that of the previous iteration. The $f(x, y)$ in Equation (4) can be defined as:

$$f(x,y) = \begin{cases} 0, & x < y, \\ 1/2, & x = y, \\ 1, & x > y. \end{cases} \tag{5}$$

The coefficient $f(F^{(n-1)}, F^n)$ prevents β_F from involving in the determination of environmental response if the Final Cost is greater than that of the previous iteration. If the Final Cost remains constant, β_F is put into action with a coefficient of $\frac{1}{2}$. If the Final Cost is smaller than before, β_F is involved. Furthermore, $f(C^{(n-1)}(Q^{(n-1)}(i), C^n(Q^n(i)))$ involves β_C in determining the environmental response if the Total Cost of the selected machine is smaller than that of the previous iteration. If it is equal to that of the previous iteration, β_C is put into action with a coefficient of $\frac{1}{2}$. If it is greater, β_C is not involved. Therefore, the environmental response is one of the 9 values shown in Table 1, in which D, U, and I indicate a decrease, an unchanged value, and an increase, respectively. If the Total Cost and the Final Cost decrease, the automata will receive the total reward. If they increase, it will receive no reward (and it will be fined).

Final Cost	Total Cost	Penalty
D	D	0
D	U	$1/2\beta_C$
D	I	β_C
U	D	$1/2\beta_F$
U	U	$1/2\beta_F + 1/2\beta_C$
U	I	$1/2\beta_F + \beta_C$
I	D	β_F
I	U	$\beta_F + 1/2\beta_C$
I	I	$\beta_F + \beta_C = 1$

Table 1. Probabilities of rewards allocated to the nine possible cases

Four tests were conducted with different values of β_C and β_F. In the first test (PS-1), all of the automata are rewarded if the total cost decreases; therefore, $\beta_C = 0$ and $\beta_F = 1$. In the second test (PS-2), an automaton is rewarded if the total cost of the selected machine is smaller than that of the previous iteration; therefore, $\beta_C = 1$ and $\beta_F = 0$. In the third test (PS-3), $\beta_C = 0.75$ and $\beta_F = 0.25$. Since $\beta_F < \beta_C$, decreasing the total cost of the selected machine will have a greater effect on reward determination than the previous iteration. In the fourth test (PS-4), $\beta_C = 0.25$ and $\beta_F = 0.75$. Unlike the previous test, increasing the final cost will have a greater effect on reward determination than the previous iteration because ($\beta_F > \beta_C$).

6.1.2 The Profitability-Based Algorithm with Random Rewarding (PR)

In this algorithm, abbreviated as PR, the appropriateness of an action, determines the probability of receiving reward. This algorithm determines the value of $\beta^{(n)}(i)$ for automata $A(i)$ by considering the Final Cost and Total Cost of the selected machine. The PR algorithm describes the environment as the P-Model; therefore, $\beta^{(n)}(i) \in \{0, 1\}$. The Final Cost of the nth iteration might be greater than, smaller than, or equal to the Final Cost of the $(n-1)$th iteration. Likewise, the Total Cost of the selected machine by automata $A(i)$ in the nth iteration might be greater than, smaller than, or equal to that of the selected machine by the automata in the $(n-1)$th iteration. Hence, there will be 9 possible cases with respect to the Final Cost and Total Cost of the selected machine in the two consecutive iterations. A probability value is allocated to each of these 9 cases. These values determine the probability of rewarding the selected action, which is obtained from Equation (7) for automata $A(i)$ in the nth iteration:

$$P^{(n)}(i) = f(F^{(n-1)}, F^n)P_F + f(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))P_C. \tag{6}$$

In Equation (6), $P_F + P_C = 1$, $P_F \neq 0$, $P_C \neq 0$, and P_F is the probability of receiving reward if the Final Cost is smaller than of the previous iteration. Moreover, P_C is the probability of receiving reward if the Total Cost of the selected machine is smaller than the Total Cost of the selected machine in the previous iteration. $f(x, y)$ is defined according to Equation (5). The environmental response to automata $A(i)$, $\beta^{(n)}(i)$ is determined through Equation (7) in the nth iteration:

$$\beta^{(n)}(i) = I(1 - P^{(n)}(i)). \tag{7}$$

In Equation (7), $I(q)$ is an indicator function [78], which returns 1 for q and 0 for $1-q$. Table 2, shows the probability of rewarding in the 9 possible cases. Accordingly, D, U, and I indicate the decreased, unchanged, and increased values, respectively, compared with the previous iteration.

Final Cost	Total Cost	Probability of receiving rewards
D	D	$p_F + P_C = 1$
D	U	$p_F + 1/2P_C$
D	I	p_F
U	D	$1/2p_F + P_C$
U	U	$1/2p_F + 1/2P_C$
U	I	$1/2p_F$
I	D	P_C
I	U	$1/2P_C$
I	I	0

Table 2. Probabilities of Rewards Allocated to the Nine Possible Cases

6.2 Cost-Efficient Load-Balancing Threshold-Based Algorithms

Threshold-based algorithms benefit from a threshold in addition to the Total Cost and Final Cost in order to determine the goodness of an action taken by an automaton. If the Final Cost is lower than the threshold, automata's action is evaluated appropriate, otherwise it is considered inappropriate. A problem of profit-based algorithms is that, an automata fails to receive the full reward when it approaches a good mapping and starts converging because the Total Cost of a selected VM or the Final Cost of consecutive iterations might remain constant. Therefore, profit-based algorithms interpret this case as inappropriate and cease to reward the automaton completely, so they prevent the automata from reaching the appropriate mapping. To avoid this problem, threshold-based algorithms use a threshold to determine reward or penalty in addition to considering the Total Cost and Final Cost. Threshold-based cost-effective load-balancing algorithms are divided in two categories:

- The threshold-based algorithm with specific rewarding (TS),
- The threshold-based algorithm with random rewarding (TR).

6.2.1 Threshold-Based Algorithm with Specific Rewarding (TS)

TS evaluates goodness of an automaton action similarly to PS algorithm. This algorithms interpret the environment as the Q-model. The Final Cost of the n^{th} iteration might be greater than, smaller than, or equal to that of the $(n-1)^{\text{th}}$ iteration. Likewise, the Total Cost of the selected VM by automaton $A(i)$ in the n^{th} iteration might be greater than, smaller than, or equal to that of the selected vm by automata in the $(n-1)^{\text{th}}$ iteration. In addition, the Final Cost might be smaller or greater than a Threshold. Thus, there are 18 possible states based on the Final Cost and Total Cost of the selected VM in two consecutive iterations. In fact, two rewarding policies are applied. One policy pertains to the case when the Final Cost is greater than the threshold, whereas the other policy indicates the case where the Final Cost is smaller than the threshold. To determine $\beta^{(n)}(i)$ a value is allocated to each of these 18 states. This value indicates the inappropriateness of automata's actions. The environmental response to automaton $A(i)$ in the n^{th} iteration is calculated in the function (8):

$$\beta^{(n)}(i) = \begin{cases} 1 - (f_l(F^{(n-1)}, F^n)\beta_f + f_l(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))\beta_c), & F^n < T, \\ 1 - (f_g(F^{(n-1)}, F^n)\beta_f + f_g(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))\beta_c), & F^n \geq T. \end{cases} \tag{8}$$

where $\beta_f + \beta_c = 1$.

In the above function, β_F is the received reward if the Final Cost is smaller than the previous iteration, and β_C is the received reward if the Total Cost of the selected VM is smaller than the Total Cost of the selected VM in the previous iteration. Moreover, T is the predefined threshold. Functions $f_l(x, y) \in \{0, 1\}$ and $f_g(x, y) \in \{0, 1/2, 1\}$, can return 0, 0.5 or 1 if the two inputs are greater, equal, or smaller. They are defined whenever necessary. In fact, f_g determines the reward and penalty policy when the Final Cost is greater than the threshold, and f_l determines the reward and penalty policy when the Final Cost is smaller than the threshold. To determine the threshold, it is possible to use the Final Cost obtained from one of the existing algorithms. For instance, the BCO algorithm is first executed in the tests for each meta task, and then the obtained Final Cost result is used as the threshold. When the Final Cost is smaller than the threshold in the TS algorithm, the equality of smallness (of the Total Cost of the selected VM or the Final Cost in comparison with the previous iteration) is regarded as the goodness solution, whereas the greatness (of the Total Cost of the selected VM and Final Cost in comparison with the previous iteration) is regarded as the inappropriate solution. When the Final Cost is greater than the threshold, then the greatness, equality, and smallness (of the Total Cost or Final Cost in comparison with the previous iteration) are regarded as inappropriate, semi-appropriate, and appropriate solutions, respectively. Accordingly, f_g and f_l are defined as functions (9) and (10):

$$f_g = \begin{cases} 0, & x < y, \\ 1/2, & x = y, \\ 1, & x > y, \end{cases} \tag{9}$$

$$f_l = \begin{cases} 0, & x < y, \\ 1, & x. \geq y \end{cases} \tag{10}$$

6.2.2 Threshold-Based Algorithm with Random Rewarding (TR)

Called TR, this algorithm acts like the PR algorithm, however, it is interpreted as the P-model. In the n^{th} iteration, the Final Cost might be greater than, smaller than, or equal to the Final Cost in the $(n-1)^{\text{th}}$ iteration. Likewise, the Total Cost of the selected VM by automaton $A(i)$ in the n^{th} iteration might be greater than, smaller than, or equal to the Total Cost of the selected VM by automata in the $(n-1)^{\text{th}}$ iteration. In addition, the Final Cost might be smaller than, greater than, or equal to a threshold. Therefore, there might be 18 possible cases based on the Final Cost and Total Cost of the selected VM in two consecutive iterations. In fact, two rewarding policies are applied. One policy is allocated to the cases in which the Final Cost is greater than the threshold, whereas the other one is allocated to the case in which the Final Cost is smaller than the threshold. To determine $\beta^{(n)}(i)$, a value is attributed to each of the 18 states. This value is the probability of rewarding the automaton. It is calculated in function (11) for automaton $A(i)$ in n^{th} iteration:

$$p^{(n)}(i) = \begin{cases} (f_g(F^{(n-1)}, F^n)p_f + f_g(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))p_c), & F^n < T, \\ (f_l(F^{(n-1)}, F^n)p_f + f_l(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))p_c), & F^n \geq T. \end{cases} \tag{11}$$

In this function, p_f is the probability of rewarding if the Final Cost is smaller than the previous iteration, and p_c is the probability of rewarding if the selected VM's Total Cost is smaller than that of the selected VM in the previous iteration $(p_f + p_c = 1)$. Moreover, Ω shows the threshold. Functions $f_l(x, y) \in \{0, 1/2, 1\}$ and $f_g(x, y) \in \{0, 1/2, 1\}$ return 0, 0.5 or 1 with respect to the greatness, equality, or smallness of two inputs, which are defined when necessary. In fact, f_l determines the reward and penalty policy when the Final Cost is greater than the threshold, and f_g determines the reward and penalty policy when the Final Cost is smaller than the threshold. To determine the threshold, it is possible to use the Final Cost obtained from one of the existing algorithms. For instance, the BCO algorithm is executed first in the tests for each meta task, and then the obtained Final Cost result is used as the threshold. The environmental response to automata $A(i)$ in the n^{th} iteration is calculated in the following way:

$$\beta^{(n)}(i) = I(1 - p^{(n)}(i)). \tag{12}$$

In Equation (12), $I(q)$ as described above is an indicator function, which returns 1 and 0 for the probabilities of q and $1-q$, respectively. When the Final Cost is smaller than the threshold in the TR algorithm, the equality or smallness (of the Total Cost of the selected VM or the Final Cost in comparison with the previous iteration) is regarded as the goodness solution, and the greatness (of the Total Cost of the selected machine or the Final Cost in comparison with the previous iteration) is regarded as the inappropriate solution. When the Final Cost is greater than the threshold, the greatness, equality, or smallness (of the Total Cost or Final Cost in comparison with the previous iteration) is considered inappropriate, semi-appropriate, and appropriate, respectively. Functions f_o and f_b are defined as Equations (9) and (10). When the Final Cost is smaller than the threshold in the TR algorithm, only the greatness (of the Total Cost of the selected VM or the Final Cost in comparison with the previous iteration) is considered inappropriate. When the Final Cost is greater than the threshold, the equality and greatness (of the Total Cost of the selected VM or the Final Cost in comparison with the previous iteration) are considered inappropriate.

6.3 Cost Utilization and Financial Efficiency Algorithm

When every job is executed on the machine with the lowest execution cost, it is defined as an ideal resource selection which is represented as $\lambda_{min}(i)$, $1 \leq j \leq N$ and evaluated as bellow relation:

$$\lambda_{min}(i) = jSuchthatECC(i,j) = minECC(i,q), \quad 1 \leq q \leq N.$$

On the other hand, when a resource is selected for a job with the heights of cost, it is defined as the worst matching which is represented as $\lambda_{max}(i)$, $1 \leq j \leq N$ and evaluated as bellow:

$$\lambda_{max}(i) = j \text{ such that } ECC(i,j) = \max ECC(i,q), \quad 1 \leq q \leq N.$$

Selecting a resource for a job can be measured by a parameter called financial efficiency [9]. To better selecting machines, the machine should be selected for a job with smaller ECC. Financial efficiency at iteration n is evaluated in Equation (13):

$$\omega^{(n)} = \frac{\sum_{0<i<M} ECC(i, \lambda_{min}(i))}{\sum_{0<i<M} ECC(i, \lambda^{(n)}(i))}, \tag{13}$$

where $0 < \omega \leq 1$. When $\omega = 1$, we have the ideal selecting. For more efficiency, the load should be balanced to avoid increasing the cost of athe machine. Load balance can be measured by cost utilization [13] as evaluated in Equation (14).

$$\sigma^{(n)} = \frac{\sum_{0<i<M} ECC(i, \lambda^{(n)}(i))}{M \times T_N}. \tag{14}$$

When the system is completely balanced (based on machines cost), then $\sigma = 1$; otherwise $\sigma < 1$. In this section, we propose a new category of load balancing algorithms by means of learning automata where the load balancing problem is reduced to an optimization problem with cost effective and cost utilization as objective functions. From Equations (13) and Equation (14), it can be inferred that Final Cost is dependent on financial efficiency and cost utilization, as shown in Equation (15).

$$F^{(n)} = \frac{\sum_{0<i<M} \frac{\lambda_{min}(i)}{N}}{\omega^{(n)} \times \sigma^{(n)}}. \tag{15}$$

Thus, to minimize the Final Cost, ω and σ must be maximized. However, these design goals are in conflict with each other because mapping jobs to their first choice of machines may cause the load imbalance.

Therefore, selecting is essentially a tradeoff between the two criteria; a good algorithm must balance between financial efficiency and cost utilization. The algorithms proposed in this section exploit learning automata to find mappings by optimizing the matching and the load balancing criteria simultaneously. Two algorithms named as CF1, CF2, are introduced and analyzed.

6.3.1 CF1 Algorithm

This algorithm interprets the environment as an S-model; that is $\beta^{(n)}(i) \in [0,1]$ To evaluate the contribution of each automaton to improve the financial efficiency and cost utilization at each iteration, we define two parameters, financial efficiency_LA (fe_LA) and max_monetization_LA (mm_LA). Input to each automaton is a linear combination of fe_LA (denoted by $\omega^{(n)}(i)$), and mm_LA (denoted by $\sigma^{(n)}(i)$:

$$\beta^{(n)}(i) = \omega^{(n)}(i)\theta_\omega + \sigma^{(n)}(i)\theta_\sigma$$

θ_ω and θ_σ are weights associated with fe_LA and mm_LA, respectively. fe_LA for each automaton $A(i)$ at iteration n is evaluated as Equation (16):

$$\omega^{(n)}(i) = \frac{ECC(i, \lambda^{(n)}(i)) - ECC(i, \lambda_{(min)}(i))}{ECC(i, \lambda_{(max)}(i)) - ECC(i, \lambda_{(min)}(i))}, \tag{16}$$

where $\lambda_{(max)}(i)$ is the worst selecting (which was mentioned above). The closer $\omega^{(n)}(i)$ to 0, the more favorable the response from the environment as far as the selecting is concerned. In the case that the automaton selects the machine with the worst selecting, $\omega^{(n)}(i)$ is evaluated to 1. mm_LA for each automaton $A(i)$ at iteration n is evaluated as Equation (17) and Equation (18):

$$\sigma^{(n)}(i) = \frac{|c^{(n)}(\lambda^{(n)}(i)) - \overline{c}^{(n)}|}{\text{Max}_{1 \le j \le N} |ci^{(n)}(j) - \overline{c}^{(n)}|}, \tag{17}$$

where

$$\overline{c}^{(n)} = \frac{\sum_{j=1}^{N} c^{(n)}(j)}{N}. \tag{18}$$

CF1 evaluates mm_LA by comparing the Total Cost of each machine with the average Total Cost of all machines. Higher reward is granted to those learning automata which select the machines with Total Cost near the average. In this way, learning automata are guided to select machines with the average Total Cost.

6.3.2 CF2 Algorithm

Algorithm CF2 is the same as CF1, but differs in evaluation of mm_LA. The heuristic for evaluation of mm_LA in algorithm CF1 suffers from two defections. Firstly, when the Total Cost is relatively balanced, the maximum distance between the Total Cost of machines and the average Total Cost is a small value. Thus, as the Total Cost gets more balanced, $\sigma(i)$ tends to higher values. As a result, even if an automaton chooses a machine with the Total Cost close to the average Total Cost, it receives a penalty. This makes the learning automata convergence more difficult.

Secondly, Total Cost evaluated in CF1 is in favor of high Total Cost machines. By numerous experiments, we learned that in certain situations, no task is assigned to a machine while others are heavily overloaded. In fact, when the average Total Cost is close to the Total Cost of machines with heavy Total Cost, thus the machines with the Total Cost close to zero have a greater distance from the average Total Cost, mm_LA of automata choosing these machines are evaluated closer to 1 than that of automata choosing machines with the higher Total Cost. Consequently, machines with the lower Total Cost have less chance to be selected. In order to address mentioned defections, CF2 evaluates mm_LA for each automaton $A(i)$ at iteration n, as shown in Equation (19):

$$\sigma^{(n)}(i) = \frac{c^{(n)}(\lambda^{(n)}(i))}{F_N^{(n)}} \left(1 - e^{\frac{-1}{2}\left(\frac{(\sigma(n)-1}{0.1}\right)^2}\right). \tag{19}$$

The former part of the above expression is close to 0 when the chosen machine has the Total Cost less than the maximum Total Cost. Thus, the learning automata are encouraged to choose machines with the low Total Cost, thus, they are guided in a way to decrease the distance between the maximum Total Cost and the minimum Total Cost. The latter part of the expression is a Gaussian function. It gets closer to 0 as the Financial Efficiency increases; therefore, when the Total Cost is relatively balanced, mm_L of each automaton is close to 0.

7 SIMULATION AND EXPERIMENTAL RESULTS

Let us presume that the cloud system composed of number of machines shown in Table 3 (each host has between 10 and 20 virtual machines). And 500 tasks which properties of some of them are included in Table 4. Termination condition occurs when no change in Final Cost is made for 2 000 consecutive repetitions, or the number of repetitions is more than 200 000. The scheduling and load balancing policies can be categories into two methods, immediate and batch mode [79, 80]. In the batch mode, jobs are gathered into a set, called metatask, which is further examined for mapping at prescheduled times. We have proposed a set of scheduling and load balancing algorithms based on Learning Automata for batch mode.

For the simulation studies, characteristics of the ECC matrices where defined above are used. The algorithms are simulated on a discrete-event Cloud simulation toolkit for cloud environment called CloudSim [81]. This toolkit provides facilities for modeling and simulating Cloud resources and Cloud users with different capabilities and configuration CloudSim is a self-configured platform that provides an extensible simulation environment which enables modeling and simulation of cloud computing systems and application provisioning environments [82]. The performance of the proposed algorithm has been analysed based on the results of the simulation. The cloud computing experiment has been carried out through the CloudSim3.0.3 simulator and this simulator runs on the machine with the configuration of Intel core i7 processor, 8 GB RAM, 3.4 GHz CPU and Window 7 platform. Table 2 shows the complete information on cloud resources distributed on different geographic sites. This method is used to show how well the scheduler service exploits resources even if there are different resources that could center run cloudlets. The aforesaid data center methods described in the above section are realized by applying packages in the CloudSim. Furthermore, "Datacenter" is the main class applied to simulation of cloud environment. It is a cloud resource whose host list is virtualized.

The CloudSim was configured with three datacenters that include 3–5 physical resources – or "host" in CloudSim terminology – with the following parameters: Architecture = 'x64', host OS = 'linux', vmware machine = "xeon 5600", Time Zone = 004 (GMT-08:00) Pacific Time (US and Canada). Table 3, presents the specifications of the configured simulation environment and resource features. Each host has between 10 and 20 virtual machines. To demonstrate the efficiency of the proposed algorithm, the obtained results were compared with BCO, PES, CJS, PPO, MCT algorithms in terms of Total Cost (TC), Final Cost (FC), Degree of imbalance, Efficiency, Waiting Time and Error Rate which are described in detail in the next section. These algorithms are mentioned with details in the Related work section.

Data Center	Host	RAM Size (MB)	MIPS	BandWith (MBPS)	Extended Memory (MB)
DC0	H1	4 096	5 100	300	100 000
	H2	1 024	4 600	300	300 000
	H3	2 048	2 500	300	1 000 000
DC1	H1	2 048	8 000	300	750 000
	H2	512	6 140	300	250 000
	H3	4 096	2 570	300	125 000
DC2	H1	1 024	6 500	300	300 000
	H2	4 096	1 250	300	800 000
	H3	512	3 540	300	450 000
	H4	2 048	2 700	300	700 000

Table 3. Cloud computing resource model

The execution unit of the instructions is based on million instruction per second (MIPS), RAM and extended memory units are based on mega byte (MByte), network bandwidth is introduced in Mega Byte Per Second (Mbps). The processing powers of each processor are specified. So, the job scheduling issue must be assigned jobs to hosts and achieve efficient resource utilization. All Cloudlets are sending to the service providers based on the Poisson distribution. This platform helps us to experiment with custom job scheduler policies, like the one presented in this work, which was in fact compared against the other job schedulers.

Job creation component is responsible to implement user actions by generating different size jobs throughout the scheduling period. As it is known, executing such type of operations must have a multi-threaded component in order to execute concurrently.

Table 4, shows the number of jobs and requirements of cloudlets applied in this experiment. We randomly generate various jobs among these three types. The introduced jobs are the combination of data intensive and computation intensive jobs.

Job Name	Length (MI)	File Size	Output Size	Execution Time (Normal Distribution)	Execution Cost
J0	40 000	2 500	100	$N(5,10)$	Based on ECC Matrix
J1	37 000	3 000	120	$N(5,10)$	Based on ECC Matrix
J2	24 000	7 800	230	$N(5,10)$	Based on ECC Matrix
J3	18 000	4 300	180	$N(5,10)$	Based on ECC Matrix
J4	42 000	11 000	400	$N(5,10)$	Based on ECC Matrix
J5	36 000	6 500	230	$N(5,10)$	Based on ECC Matrix
J6	22 000	3 500	320	$N(5,10)$	Based on ECC Matrix
J7	31 000	7 600	160	$N(5,10)$	Based on ECC Matrix
J8	19 000	10 000	220	$N(5,10)$	Based on ECC Matrix
J9	24 000	9 600	290	$N(5,10)$	Based on ECC Matrix
J10	31 000	4 300	310	$N(5,10)$	Based on ECC Matrix

Table 4. Job features and specifications

7.1 Performance Evaluation Parameters

In addition to the proposed framework, the designed evaluation parameters are among the innovations of this paper in comparison with the existing algorithms. In this paper, the proposed and existing algorithms are evaluated in terms of Total Cost (TC), Final Cost (FC), Degree of imbalance, Efficiency, Waiting Time and Error Rate parameters. for the first time two main important parameters TC and FC are defined based on tasks execution cost. in reported economic-based scheduling and load balancing studies so far, evaluation of algorithms are often based on some existing economic parameters such as, the users profit, system utilization or resource owner profits. The definitions of these two parameters are mentioned below:

Total Cost: The required cost of executing all the tasks assigned to a machine in the n^{th} iteration is called the Total Cost (TC) of that machine, indicated by $C^{(n)}(j)$:

$$c^{(n)}(j) = \sum ECC(k,j), \quad j = Q(k), \quad 1 \leq k \leq M$$

Final Cost: The maximum value of $C^{(n)}(j)$, $1 \leq j \leq N$ in above equation is called the Final Cost (FC) in the n^{th} iteration.

Degree of Imbalance: It is one of the most important parameters for evaluating the proposed algorithms and improving this parameter, is one of the main objectives of this paper. It is calculated based on Equation (20). The proposed definitions for the degree of imbalance in existing papers such as [83, 84] are often designed based on the total tasks execution time on a VM and computational power of that VM. In this paper, it is for the first time when the purely economic-based definition of imbalance degree has been proposed. Based on the following equation, the proposed definition is based on the TC parameter mentioned above.

$$\begin{aligned} c^n_{avg} &= \frac{c^n(1) + c^n(2) + \cdots + c^n(m)}{m}, \\ c^n_{min} &= \min\{c^n(1) + c^n(2) + \cdots + c^n(m)\}, \\ c^n_{max} &= \max\{c^n(1) + c^n(2) + \cdots + c^n(m)\}, \\ imbalance\_d &= \frac{c^n_{max} - c^n_{min}}{c^n_{avg}}. \end{aligned} \tag{20}$$

Efficiency: The definitions of Efficiency in most of the reported studies such as [85, 86] are mostly based on speed and completion time. In this paper, for the first time, an innovation has been created in the definition of efficiency. A completely economic-based definition of efficiency is proposed based on Equation (21). The denominator in Equation (21) indicates the total cost obtained for the execution of all the assigned tasks in cloud environment. So, for each of algorithms a larger number represents a better result.

$$Efficiency = \frac{M}{\sum_{j=1}^{N} \sum_{k=1}^{M} ECC(k,j)}. \tag{21}$$

Error rate: The Error rate parameter in load balancing algorithms is regulated with regard to each allocation, for instance in failure cases of allocation, or in cases demanding reallocation.

Waiting time: It is defined as the time when a process waits from its submission to completion in the queues.

7.1.1 Total Cost (TC) and Final Cost (FC)

The Total Cost (TC) is one of the most important parameters, and the proposed algorithms are designed according to it. In fact, this parameter is general optimization criteria which the proposed algorithms are trying to reduce and balance. It can be defined as the total monetization by a machine for executing all allocated jobs. According to Figures 4, 5 all proposed algorithms PS, TS, PR outperforms the existing algorithms in the inconsistent environment, and in the consistent environment TS and PS outperform the existing algorithms but CJS algorithm has a better result from the proposed PR and TR algorithms. The results also show that algorithm PS performs better than others in the inconsistent environment and in the consistent environment TS performs better than others. Figure 6 also shows that with respect to the number of virtual machines, the proposed algorithms PS have better performance than existing algorithms and between the proposed algorithms produces the best result.

Another important optimization criteria is Final Cost (FC) as previously explained, is maximum value of Total Cost, that's mean choosing a machine which has the highest monetization. Based on the results shown in Figure 4, PS, TS and PR have better performance than other algorithms and CJS algorithm outperform the proposed algorithms CF2, TR and CF1 in the inconsistent environment. Figure 5 shows that in the consistent environment TS and PS algorithms have a better performance but CJS algorithm outperforms other proposed algorithms and other existing algorithms. According to Figure 6, the results obtained for comparing the proposed algorithms with the existing algorithms, are precisely similar to the Total Cost (TC) parameter. Figures 7, 8 and 9 show the comparison of the Final Cost parameter of the proposed algorithms with the existing algorithms. As shown in Figure 7 in the inconsistent environment PS and TS have a better performance than other algorithms and CJS. CJS algorithm outperforms the proposed algorithms CF2, TR and CF1. Figure 8 shows that TS has a better performance in consistent environment. Finally, the simulation results shown in Figure 9 show the comparison of the Final Cost according to the number of VMs.

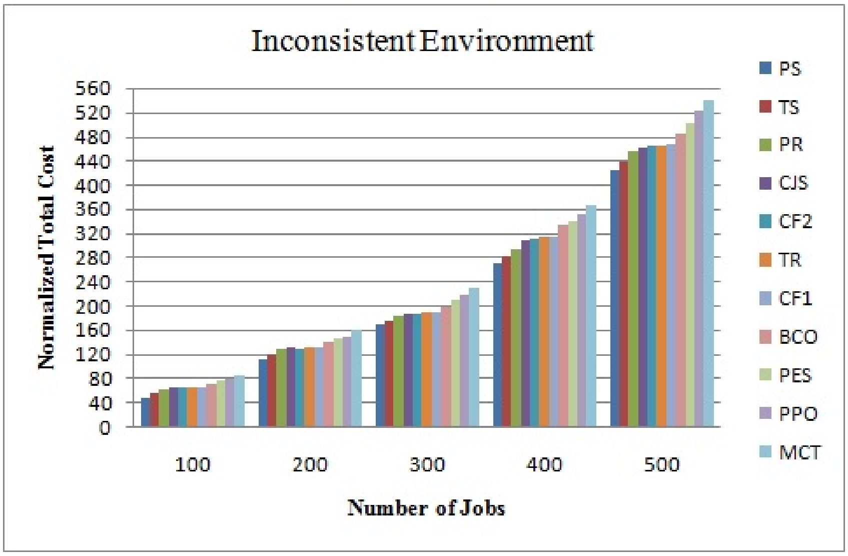

Figure 4. Total Cost versus number of jobs in proposed and existing algorithms in inconsistent environment

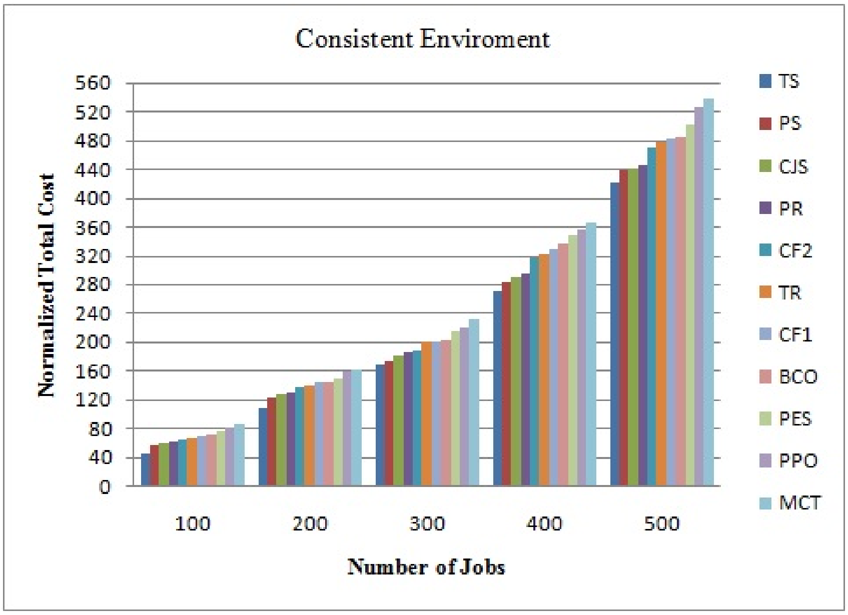

Figure 5. Total Cost versus number of jobs in proposed and existing algorithms in consistent environment

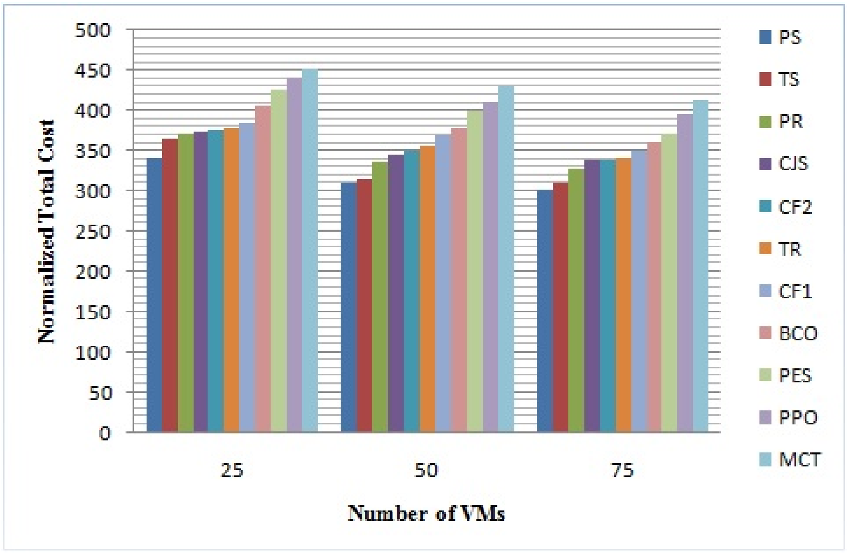

Figure 6. Total Cost versus number of VMs in proposed and existing algorithms in consistent environment

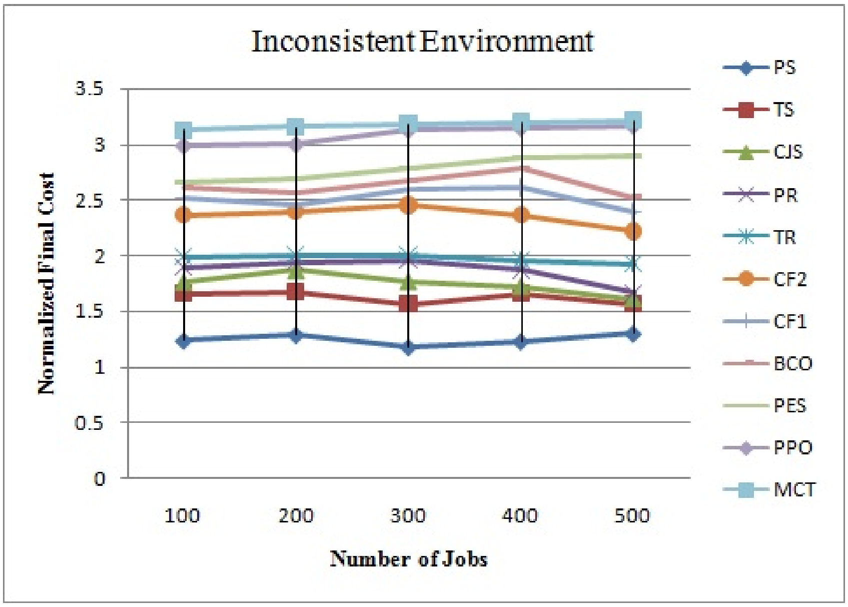

Figure 7. Final Cost versus number of jobs in proposed and existing algorithms in inconsistent environment

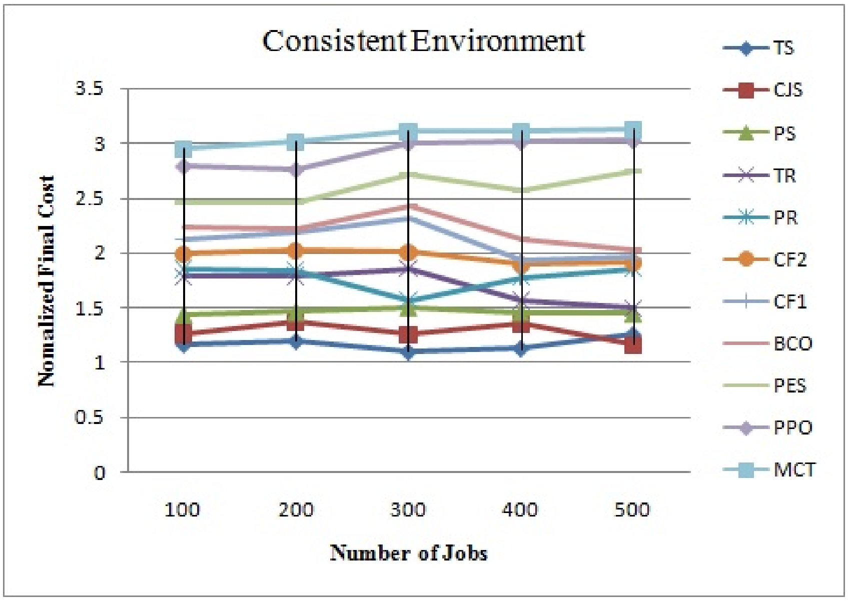

Figure 8. Final Cost versus number of jobs in proposed and existing algorithms in consistent environment

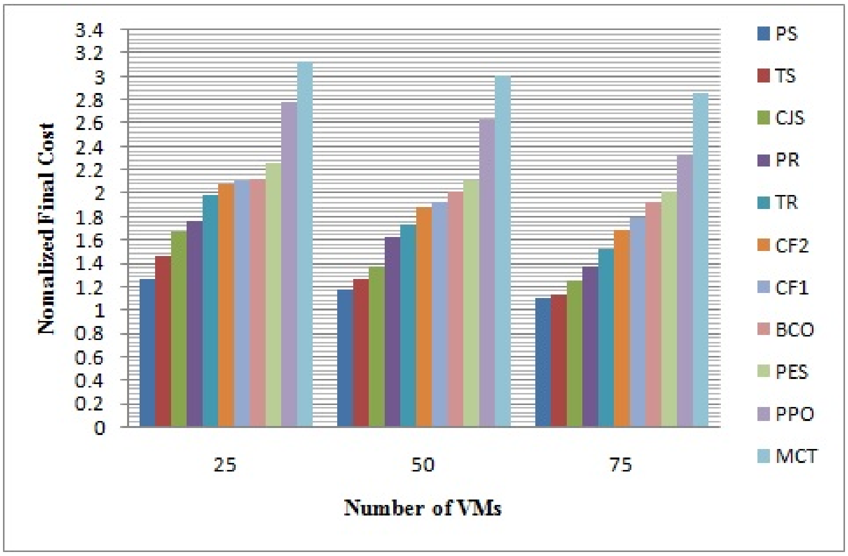

Figure 9. Final Cost versus number of VMs in proposed and existing algorithms in consistent environment

7.1.2 Waiting Time

Waiting time is the total time spent by the job in the ready state for allocate to a machine and execute. Figure 10 presents Average waiting time required by every job in the proposed and existing algorithms. It shows that the average waiting time for each algorithm has rises by increasing the number of jobs. Usually, load balancing policies lead to a reduction in waiting time because one of the important goals of traditional load balancing algorithms is often time criteria, but in the proposed algorithms the criterion of load balancing is the resource's cost and preventing excessive increase in the price of a resource, so unlike traditional approaches of load balancing, the waiting time of the proposed algorithms is longer than traditional models. Also, the centrality of learning automata in the proposed algorithms can increase waiting time. This is also true about PPO algorithm. According to Figure 10 among the proposed algorithms, TR has the shortest waiting time, and PS has the longest waiting time and between all simulated algorithms the MCT algorithm provides the best waiting time.

7.1.3 Degree of Imbalance

Degree of imbalance is one of the most important parameters for the evaluation of the proposed algorithm. Degree of imbalance indicates whether the jobs are distributed monotonously among VMs or not. Traditional load balancing algorithms give a discussion on the Degree of Imbalance (DI) by presenting its relationship with

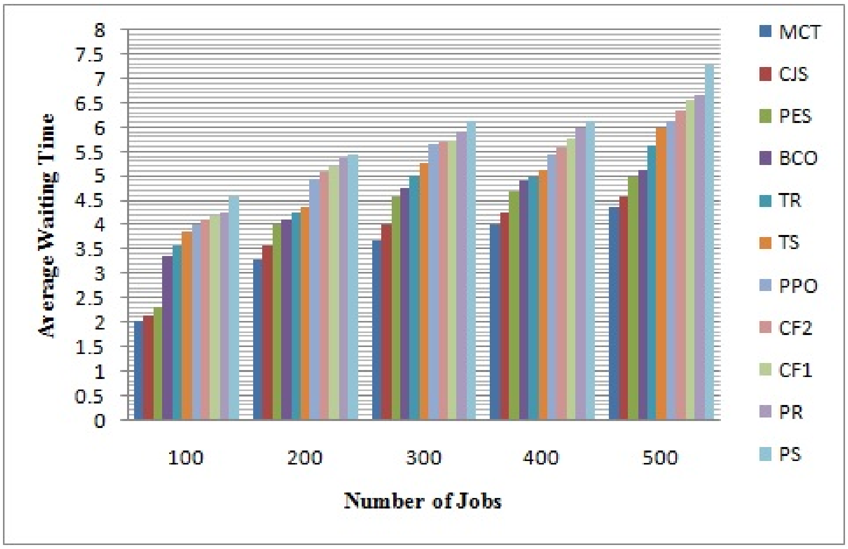

Figure 10. Average waiting time versus number of jobs comparisons

makespan, resource utilization or completion time. Our objective is to reduce the Degree of Imbalance of all VMs in a dynamic cloud computing environment. However, in this paper, due to the design of economic criteria in proposed scheduling and load balancing algorithms, the imbalance factor is also based on economic parameter (TC). The small value for the degree of imbalance (DI) in Equation (20) unveils how tasks on a system are balanced, either the better the performance. Hence, our aim to ensure load balancing across VMs through minimizing Degree of Imbalance is achieved. In denoted Equation, the closer difference between the maximum and minimum Total Cost (TC) of VMs to the average Total Cost (TC) will decrease Degree of Imbalance. According to Figure 11 all proposed algorithms outperform the existing ones. Between the proposed algorithms, PS algorithm has the best results. And between the existing algorithms, CJS has a better performance.

7.1.4 Efficiency

Other most important parameter in the evaluation of the proposed algorithms and existing algorithms is efficiency. Based on the purposes of this paper, the authors presented a new definition of efficiency based on economic criteria illustrated in Equation (21) to recognize the quality of economic distribution of jobs among VMs, so that jobs can be executed at a lower cost. The higher value in Equation (21) represents better results. As shown in Figure 12 the proposed algorithms PS and TS outperform other algorithms and also one of the existing algorithms called CJS has better results than the proposed algorithms PR, TR, CF1 and CF2. Also, with the increase in the number of VMs, efficiency has increased.

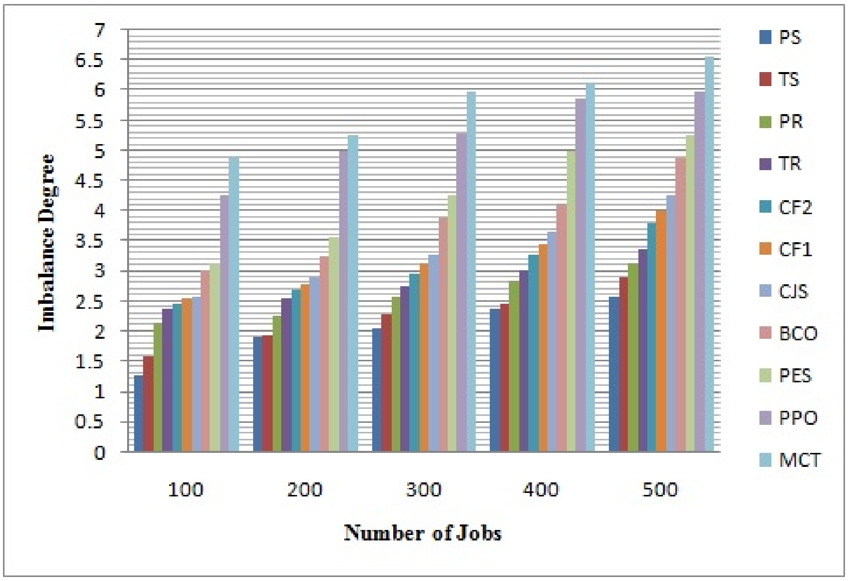

Figure 11. Imbalance Degree versus number of jobs comparisons

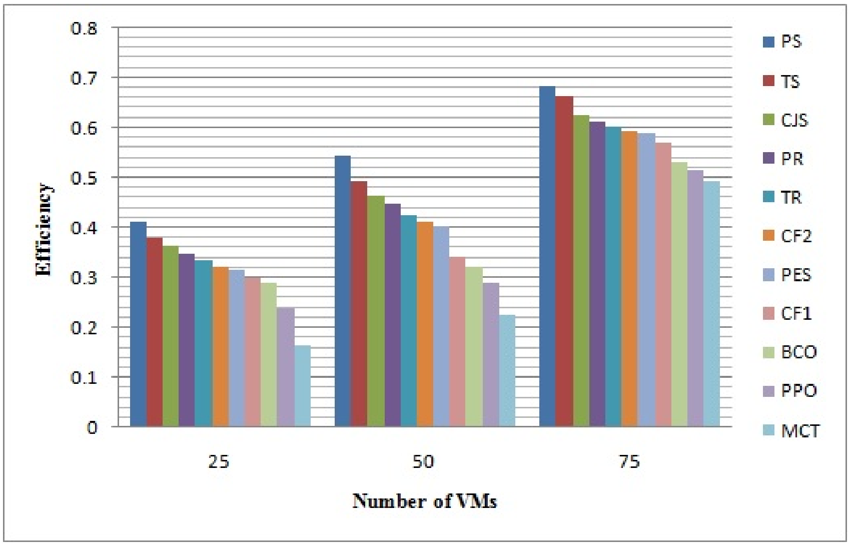

Figure 12. Efficiency versus number of jobs in proposed and existing algorithms

7.1.5 Error Rate

The Error Rate is the number of jobs not completed successfully, due to any reason, in the total number of jobs submitted. As shown in Figure 13 the Error Rate in the proposed algorithms is higher than the existing algorithms due to the use of learning automata in the structure of the proposed algorithms. The learning automata is prone to higher Error Rates as many iterations and reallocation job to VMs converge to desired response. MCT algorithm has the lowest error rate among all the proposed and existing algorithms.

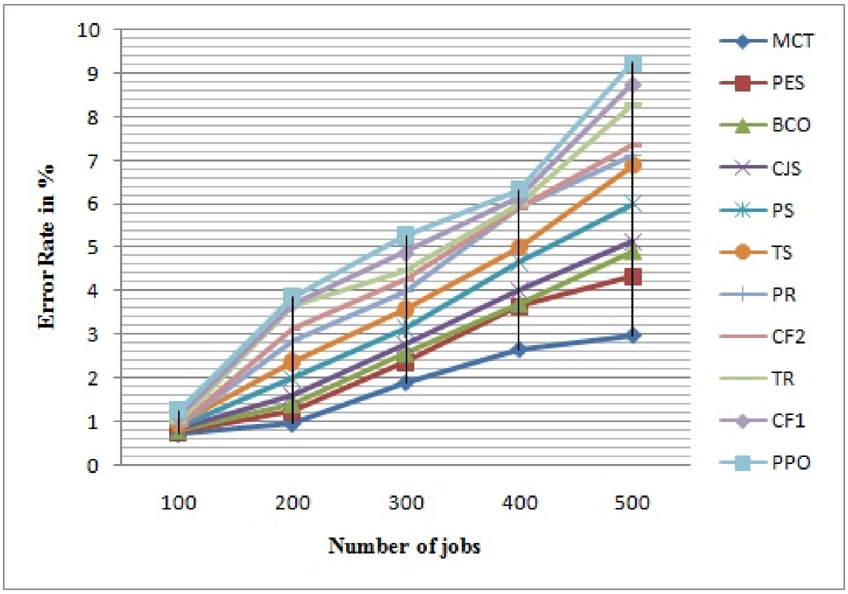

Figure 13. Error Rate versus number of jobs in the presented and existing algorithms

8 CONCLUSIONS

Cloud computing is a collection of distributed and parallel computing to create shared resources, including hardware, software, and information. scheduling and load balancing is an important technique to enhance the overall performance of the distributed system. Usually traditional methods for scheduling and load balancing in cloud environments are often based on concepts such as Makespan, execution time, resource utilization etc. In this paper, for the first time, a complete set of learning automata based algorithms with economic criteria such as Total Cost (TC), Final Cost (FC) for solving load balancing challenge are proposed. The main idea in the proposed algorithms is the fair allocation of tasks to VMs in order to prevent the excessive increase in the price of one VMs and unemployment of other VMs. For performance evaluation of the proposed algorithms and comparison with existing mechanisms, several simulations have been performed on consistent and inconsistent cloud computing environment with cloudsim simulator. Finally, the results of the proposed algorithms PS, PR, TS, TR, CF1 and CF2 were compared with the results of BCO, CJS, PES, PPO and MCT algorithms. Results show that the proposed algorithms outperform the existing algorithms in terms of Total Cost (TC), Final Cost (FC), Degree of Imbalance and efficiency.

Acknowledgment

This article was derived from a Ph.D. degree thesis in the Islamic Azad University-Arak branch.

REFERENCES

[1] Rochwerger, B.—Breitgand, D.—Levy, E.—Galis, A.—Nagin, K.—Llorente, I. M.—Montero, R.—Wolfsthal, Y.—Elmroth, E.—Caceres, J.—Ben-Yehuda, M.—Emmerich, W.—Galan, F.: The Reservoir Model and Architecture for Open Federated Cloud Computing. IBM Journal of Research and Development, Vol. 53, 2009, No. 4, pp. 4:1–4:11, doi: 10.1147/JRD.2009.5429058.

[2] Nurmi, D.—Wolski, R.—Grzegorczyk, C.—Obertelli, G.—Soman, S.—Youseff, L.—Zagorodnov, D.: The Eucalyptus Open-Source Cloud-Computing System. 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 124–131, doi: 10.1109/CCGRID.2009.93.

[3] Foster, I.—Zhao, Y.—Raicu, I.—Lu, S.: Cloud Computing and Grid Computing 360-Degree Compared. 2008 Grid Computing Environments Workshop, IEEE, 2008, pp. 1–10, doi: 10.1109/GCE.2008.4738445.

[4] Fang, Y.—Wang, F.—Ge, J.: A Task Scheduling Algorithm Based on Load Balancing in Cloud Computing. In: Wang, F. L., Gong, Z., Luo, X., Lei, J. (Eds.): Web Information Systems and Mining (WISM 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6318, 2010, pp. 271–277, doi: 10.1007/978-3-642-16515-3_34.

[5] Ernemann, C.—Hamscher, V.—Yahyapour, R.: Economic Scheduling in Grid Computing. In: Feitelson, D. G., Rudolph, L., Schwiegelshohn, U. (Eds.): Job Scheduling Strategies for Parallel Processing (JSSPP 2002). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2537, 2002, pp. 128–152, doi: 10.1007/3-540-36180-4_8.

[6] Garg, S. K.—Yeo, C. S.—Anandasivam, A.—Buyya, R.: Environment-Conscious Scheduling of HPC Applications on Distributed Cloud-Oriented Data Centers. Journal of Parallel and Distributed Computing, Vol. 71, 2011, No. 6, pp. 732–749, doi: 10.1016/j.jpdc.2010.04.004.

[7] Arunarani, A. R.—Manjula, D.—Sugumaran, V.: Task Scheduling Techniques in Cloud Computing: A Literature Survey. Future Generation Computer Systems, Vol. 91, 2019, pp. 407–415, doi: 10.1016/j.future.2018.09.014.

[8] Kim, S. I.—Kim, H. T.—Kang, G. S.—Kim, J. K.: Using DVFS and Task Scheduling Algorithms for a Hard Real-Time Heterogeneous Multicore Processor Environment. Proceedings of the 2013 Workshop on Energy Efficient High Performance Parallel and Distributed Computing (EEHPDC '13), ACM, 2013, pp. 23–30, doi: 10.1145/2480347.2480350.

[9] Houssein, E. H.—Gad, A. G.—Wazery, Y. M.—Suganthan, P. N.: Task Scheduling in Cloud Computing Based on Meta-Heuristics: Review, Taxonomy, Open Challenges, and Future Trends. Swarm and Evolutionary Computation, Vol. 62, 2021, Art. No. 100841, doi: 10.1016/j.swevo.2021.100841.

[10] Wu, D.: Cloud Computing Task Scheduling Policy Based on Improved Particle Swarm Optimization. 2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), IEEE, 2018, pp. 99–101, doi: 10.1109/ICVRIS.2018.00032.

[11] Abualigah, L.—Diabat, A.: A Novel Hybrid Antlion Optimization Algorithm for Multi-Objective Task Scheduling Problems in Cloud Computing Environments. Cluster Computing, Vol. 24, 2021, pp. 205–223, doi: 10.1007/s10586-020-03075-5.

[12] Bacanin, N.—Bezdan, T.—Tuba, E.—Strumberger, I.—Tuba, M.—Zivkovic, M.: Task Scheduling in Cloud Computing Environment by Grey Wolf Optimizer. 2019 27th Telecommunications Forum (TELFOR), IEEE, 2019, pp. 1–4, doi: 10.1109/TELFOR48224.2019.8971223.

[13] Jang, S. H.—Kim, T. Y.—Kim, J. K.—Lee, J. S.: The Study of Genetic Algorithm-Based Task Scheduling for Cloud Computing. International Journal of Control and Automation, Vol. 5, 2012, No. 4, pp. 157–162.

[14] Yiqiu, F.—Xia, X.—Junwei, G.: Cloud Computing Task Scheduling Algorithm Based on Improved Genetic Algorithm. 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2019, pp. 852–856, doi: 10.1109/ITNEC.2019.8728996.

[15] Hota, A.—Mohapatra, S.—Mohanty, S.: Survey of Different Load Balancing Approach-Based Algorithms in Cloud Computing: A Comprehensive Review. In: Behera, H. S., Nayak, J., Naik, B., Abraham, A. (Eds.): Computational Intelligence in Data Mining (CIDM 2017). Springer, Singapore, Advances in Intelligent Systems and Computing, Vol. 711, 2019, pp. 99–110, doi: 10.1007/978-981-10-8055-5_10.

[16] Afzal, S.—Kavitha, G.: Load Balancing in Cloud Computing – A Hierarchical Taxonomical Classification. Journal of Cloud Computing, Vol. 8, 2019, No. 1, Art. No. 22, doi: 10.1186/s13677-019-0146-7.

[17] Jyoti, A.—Shrimali, M.—Tiwari, S.—Singh, H. P.: Cloud Computing Using Load Balancing and Service Broker Policy for IT Service: A Taxonomy and Survey. Journal of Ambient Intelligence and Humanized Computing, Vol. 11, 2020, No. 11, pp. 4785–4814, doi: 10.1007/s12652-020-01747-z.

[18] Ala'anzy, M.—Othman, M.: Load Balancing and Server Consolidation in Cloud Computing Environments: A Meta-Study. IEEE Access, Vol. 7, 2019, pp. 141868–141887, doi: 10.1109/ACCESS.2019.2944420.

[19] Kumar, P.—Kumar, R.: Issues and Challenges of Load Balancing Techniques in Cloud Computing: A Survey. ACM Computing Surveys, Vol. 51, 2019, No. 6, Art. No. 120, doi: 10.1145/3281010.

[20] Gamal, M.—Rizk, R.—Mahdi, H.—Elnaghi, B. E.: Osmotic Bio-Inspired Load Balancing Algorithm in Cloud Computing. IEEE Access, Vol. 7, 2019, pp. 42735–42744, doi: 10.1109/ACCESS.2019.2907615.

[21] Mapetu, J. P. B.—Chen, Z.—Kong, L.: Low-Time Complexity and Low-Cost Binary Particle Swarm Optimization Algorithm for Task Scheduling and Load Balancing in Cloud Computing. Applied Intelligence, Vol. 49, 2019, No. 9, pp. 3308–3330, doi: 10.1007/s10489-019-01448-x.

[22] Hu, J.—Gu, J.—Sun, G.—Zhao, T.: A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment. 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming, IEEE, 2010, pp. 89–96, doi: 10.1109/PAAP.2010.65.

[23] Toporkov, V.—Toporkova, A.—Bobchenkov, A.—Yemelyanov, D.: Resource Selection Algorithms for Economic Scheduling in Distributed Systems. Procedia Computer Science, Vol. 4, 2011, pp. 2267–2276, doi: 10.1016/j.procs.2011.04.247.

[24] Malawski, M.—Juve, G.—Deelman, E.—Nabrzyski, J.: Algorithms for Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds. Future Generation Computer Systems, Vol. 48, 2015, pp. 1–18, doi: 10.1016/j.future.2015.01.004.

[25] Buyya, R.: Economic-Based Distributed Resource Management and Scheduling for Grid Computing. Ph.D. Thesis. Monash University, Melbourne, Australia, 1968, doi: 10.48550/arXiv.cs/0204048.

[26] Toporkov, V.—Bobchenkov, A.—Toporkova, A.—Tselishchev, A.—Yemelyanov, D.: Slot Selection and Co-Allocation for Economic Scheduling in Distributed Computing. In: Malyshkin, V. (Ed.): Parallel Computing Technologies (PaCT 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6873, 2011, pp. 368–383, doi: 10.1007/978-3-642-23178-0_32.

[27] Menouer, T.—Cérin, C.—Hsu, C. H.: Opportunistic Scheduling and Resources Consolidation System Based on a New Economic Model. The Journal of Supercomputing, Vol. 76, 2020, No. 12, pp. 9942–9975, doi: 10.1007/s11227-020-03231-z.

[28] Chaudhary, D.—Kumar, B.: Cost Optimized Hybrid Genetic-Gravitational Search Algorithm for Load Scheduling in Cloud Computing. Applied Soft Computing, Vol. 83, 2019, Art. No. 105627, doi: 10.1016/j.asoc.2019.105627.

[29] Alkhanak, E. N.—Lee, S. P.: A Hyper-Heuristic Cost Optimisation Approach for Scientific Workflow Scheduling in Cloud Computing. Future Generation Computer Systems, Vol. 86, 2018, pp. 480–506, doi: 10.1016/j.future.2018.03.055.

[30] Belgacem, A.—Beghdad-Bey, K.: Multi-Objective Workflow Scheduling in Cloud Computing: Trade-Off Between Makespan and Cost. Cluster Computing, Vol. 25, 2022, No. 1, pp. 579–595, doi: 10.1007/s10586-021-03432-y.

[31] Wu, C. Q.—Lin, X.—Yu, D.—Xu, W.—Li, L.: End-to-End Delay Minimization for Scientific Workflows in Clouds Under Budget Constraint. IEEE Transactions on Cloud Computing, Vol. 3, 2014, No. 2, pp. 169–181, doi: 10.1109/TCC.2014.2358220.

[32] Thanasias, V.—Lee, C.—Hanif, M.—Kim, E.—Helal, S.: VM Capacity-Aware Scheduling Within Budget Constraints in IaaS Clouds. PloS ONE, Vol. 11, 2016, No. 8, Art. No. e0160456, doi: 10.1371/journal.pone.0160456.

[33] Bittencourt, L. F.—Madeira, E. R. M.: HCOC: A Cost Optimization Algorithm for Workflow Scheduling in Hybrid Clouds. Journal of Internet Services and Applications, Vol. 2, 2011, pp. 207–227, doi: 10.1007/s13174-011-0032-0.

[34] Mansouri, N.—Javidi, M. M.: Cost-Based Job Scheduling Strategy in Cloud Computing Environments. Distributed and Parallel Databases, Vol. 38, 2020, No. 2, pp. 365–400, doi: 10.1007/s10619-019-07273-y.

[35] Toporkov, V.—Yemelyanov, D.—Bobchenkov, A.—Potekhin, P.: Preference-Based Economic Scheduling in Grid Virtual Organizations. Procedia Computer Science, Vol. 80, 2016, pp. 1071–1082, doi: 10.1016/j.procs.2016.05.411.

[36] Asnaashari, M.—Meybodi, M. R.: Irregular Cellular Learning Automata and Its Aplication to Clustering in Sensor Networks. Proceedings of 15th Conference on Electrical Engineering (15th ICEE), Volume on Communication, 2007.

[37] Ranjbari, M.—Torkestani, J. A.: A Learning Automata-Based Algorithm for Energy and SLA Efficient Consolidation of Virtual Machines in Cloud Data Centers. Journal of Parallel and Distributed Computing, Vol. 113, 2018, pp. 55–62.

[38] Che, H.—Li, S. Q.—Lin, A.: Adaptive Resource Management for Flow-Based IP/ATM Hybrid Switching Systems. IEEE/ACM Transactions on Networking, Vol. 6, 1998, No. 5, pp. 544–557, doi: 10.1109/90.731188.

[39] Sarhadi, A.—Meybodi, M. R.: New Algorithm for Resource Selection in Economic Grid with the Aim of Cost Optimization Using Learning Automata. 2010 International Conference on Challenges in Environmental Science and Computer Engineering, IEEE, Vol. 1, 2010, pp. 32–35, doi: 10.1109/CESCE.2010.185.

[40] Sarhadi, A.: Learning Automata Based Method for Grid Computing Resource Valuation with Resource Suitability Criteria. International Journal of Grid Computing and Applications, Vol. 2, 2011, No. 4, pp. 1–9, doi: 10.5121/ijgca.2011.2401.

[41] Sahoo, S.—Sahoo, B.—Turuk, A. K.: A Learning Automata-Based Scheduling for Deadline Sensitive Task in the Cloud. IEEE Transactions on Services Computing, Vol. 14, 2019, No. 6, pp. 1662–1674, doi: 10.1109/TSC.2019.2906870.

[42] Zhu, L.—Huang, K.—Hu, Y.—Tai, X.: A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata. IEEE Access, Vol. 9, 2021, pp. 81236–81252, doi: 10.1109/ACCESS.2021.3078773.

[43] Krishna, P. V.—Misra, S.—Nagaraju, D.—Saritha, V.—Obaidat, M. S.: Learning Automata Based Decision Making Algorithm for Task Offloading in Mobile Cloud. 2016 International Conference on Computer, Information and Telecommunication Systems (CITS), 2016, pp. 1–6, doi: 10.1109/CITS.2016.7546451.

[44] Rahmanian, A. A.—Ghobaei-Arani, M.—Tofighy, S.: A Learning Automata-Based Ensemble Resource Usage Prediction Algorithm for Cloud Computing Environment. Future Generation Computer Systems, Vol. 79, 2018, pp. 54–71, doi: 10.1016/j.future.2017.09.049.

[45] Kunz, T.: The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. IEEE Transactions on Software Engineering, Vol. 17, 1991, No. 7,

pp. 725–730, doi: 10.1109/32.83908.

[46] Misra, S.—Krishna, P. V.—Kalaiselvan, K.—Saritha, V.—Obaidat, M. S.: Learning Automata-Based QoS Framework for Cloud IaaS. IEEE Transactions on Network and Service Management, Vol. 11, 2014, No. 1, pp. 15–24, doi: 10.1109/TNSM.2014.011614.130429.

[47] Velusamy, G.—Lent, R.: Dynamic Cost-Aware Routing of Web Requests. Future Internet, Vol. 10, 2018, No. 7, 57 pp., doi: 10.3390/fi10070057.

[48] Meybodi, M. R.: Learning Automata and Its Application to Priority Assignment in a Queueing System with Unknown Characteristics. Ph.D. Thesis. The University of Oklahoma, 1983.

[49] Howell, M. N.—Frost, G. P.—Gordon, T. J.—Wu, Q. H.: Continuous Action Reinforcement Learning Applied to Vehicle Suspension Control. Mechatronics, Vol. 7, 1997, No. 3, pp. 263–276, doi: 10.1016/S0957-4158(97)00003-2.

[50] El-Osery, A. I.—Baird, D.—Abd-Almageed, W.: A Learning Automata Based Power Management for Ad-Hoc Networks. 2005 IEEE International Conference on Systems, Man and Cybernetics, Vol. 4, 2005, pp. 3569–3573, doi: 10.1109/ICSMC.2005.1571701.

[51] Narendra, K. S.—Thathachar, M. A. L.: Learning Automata – A Survey. IEEE Transactions on Systems, Man, and Cybernetics, 1974, No. 4, pp. 323–334, doi: 10.1109/TSMC.1974.5408453.

[52] Atlasis, A. F.—Saltouros, M. P.—Vasilakos, A. V.: On the Use of a Stochastic Estimator Learning Algorithm to the ATM Routing Problem: A Methodology. Computer Communications, Vol. 21, 1998, No. 6, pp. 538–546, doi: 10.1016/S0140-3664(98)00122-4.

[53] Billard, E.—Lakshmivarahan, S.: Simulation of Period-Doubling Behavior in Distributed Learning Automata. Proceedings of the 1998 ACM Symposium on Applied Computing (SAC '98), 1998, pp. 690–695, doi: 10.1145/330560.331066.

[54] Economides, A. A.—Ioannou, P. A.—Silvester, J. A.: Decentralized Adaptive Routing for Virtual Circuit Networks Using Stochastic Learning Automata. IEEE INFOCOM '88, Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies. Networks: Evolution or Revolution?, 1988, pp. 613–622, doi: 10.1109/INFCOM.1988.12972.

[55] Narendra, K. S.—Thathachar, M. A. L.: On the Behavior of a Learning Automaton in a Changing Environment with Application to Telephone Traffic Routing. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 10, 1980, No. 5, pp. 262–269, doi: 10.1109/TSMC.1980.4308485.

[56] Beigy, H.: Intelligent Channel Assignment in Cellular Networks: A Learning Automata Approach. Ph.D. Thesis. Amirkabir University of Technology, Tehran, Iran, 2004.

[57] Beigy, H.—Meybodi, M. R.: A Mathematical Framework for Cellular Learning Automata. Advances in Complex Systems, Vol. 7, 2004, No. 03n04, pp. 295–319, doi: 10.1142/S0219525904000202.

[58] Hariri, A.—Rastegar, R.—Navi, K.—Zamani, M. S.—Meybodi, M. R.: Cellular Learning Automata Based Evolutionary Computing (CLA-EC) for Intrinsic

Hardware Evolution. 2005 NASA/DoD Conference on Evolvable Hardware (EH '05), 2005, pp. 294–297, doi: 10.1109/EH.2005.12.

[59] Mnih, V.—Badia, A. P.—Mirza, M.—Graves, A.—Lillicrap, T.—Harley, T.—Silver, D.—Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: Balcan, M. F., Weinberger, K. Q. (Eds.): Proceedings of the 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 48, 2016, pp. 1928–1937.

[60] Schulman, J.—Wolski, F.—Dhariwal, P.—Radford, A.—Klimov, O.: Proximal Policy Optimization Algorithms. 2017, doi: 10.48550/arXiv.1707.06347.

[61] Haarnoja, T.—Zhou, A.—Abbeel, P.—Levine, S.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: Dy, J., Krause, A. (Eds.): Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 80, 2018, pp. 1861–1870.

[62] Cassandra, A. R.: A Survey of POMDP Applications. Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes, 1998, pp. 17–24.

[63] Mnih, V.—Kavukcuoglu, K.—Silver, D.—Rusu, A. A.—Veness, J. et al.: Human-Level Control Through Deep Reinforcement Learning. Nature, Vol. 518, 2015, No. 7540, pp. 529–533, doi: 10.1038/nature14236.

[64] Shojafar, M.—Javanmardi, S.—Abolfazli, S.—Cordeschi, N.: FUGE: A Joint Meta-Heuristic Approach to Cloud Job Scheduling Algorithm Using Fuzzy Theory and a Genetic Method. Cluster Computing, Vol. 18, 2015, pp. 829–844, doi: 10.1007/s10586-014-0420-x.

[65] Parthasarathy, S.—Jothi Venkateswaran, C.: Scheduling Jobs Using Oppositional-GSO Algorithm in Cloud Computing Environment. Wireless Networks, Vol. 23, 2017, pp. 2335–2345, doi: 10.1007/s11276-016-1264-5.

[66] Kim, S. S.—Byeon, J. H.—Yu, H.—Liu, H.: Biogeography-Based Optimization for Optimal Job Scheduling in Cloud Computing. Applied Mathematics and Computation, Vol. 247, 2014, pp. 266–280, doi: 10.1016/j.amc.2014.09.008.

[67] Ghanbari, S.—Othman, M.: A Priority Based Job Scheduling Algorithm in Cloud Computing. International Conference on Advances Science and Contemporary Engineering 2012 (ICASCE 2012), 2012.

[68] Lawrance, H.—Silas, S.: Efficient QoS Based Resource Scheduling Using PAPRIKA Method for Cloud Computing. International Journal of Engineering Science and Technology (IJEST), Vol. 5, 2013, No. 3, pp. 638–643.

[69] Zhao, J.—Zeng, W.—Liu, M.—Li, G.: Multi-Objective Optimization Model of Virtual Resources Scheduling Under Cloud Computing and It's Solution. 2011 International Conference on Cloud and Service Computing, IEEE, 2011, pp. 185–190, doi: 10.1109/CSC.2011.6138518.

[70] Tayal, S.: Tasks Scheduling Optimization for the Cloud Computing Systems. IJAEST – International Journal of Advanced Engineering Sciences and Technologies, Vol. 5, 2011, No. 2, pp. 111–115.

[71] Raju, R.—Babukarthik, R. G.—Chandramohan, D.—Dhavachelvan, P.—

VENGATTARAMAN, T.: Minimizing the Makespan Using Hybrid Algorithm for Cloud Computing. 2013 3rd IEEE International Advance Computing Conference (IACC), 2013, pp. 957–962, doi: 10.1109/IAdCC.2013.6514356.

[72] GOGULAN, R.—KAVITHA, A.—KUMAR, U. K.: An Multiple Pheromone Algorithm for Cloud Scheduling with Various QoS Requirements. International Journal of Computer Science Issues (IJCSI), Vol. 9, 2012, No. 3, pp. 232–238.

[73] LUO, J.—RAO, L.—LIU, X.: Temporal Load Balancing with Service Delay Guarantees for Data Center Energy Cost Optimization. IEEE Transactions on Parallel and Distributed Systems, Vol. 25, 2013, No. 3, pp. 775–784, doi: 10.1109/TPDS.2013.69.

[74] CHAISIRI, S.—LEE, B. S.—NIYATO, D.: Optimization of Resource Provisioning Cost in Cloud Computing. IEEE Transactions on Services Computing, Vol. 5, 2011, No. 2, pp. 164–177, doi: 10.1109/TSC.2011.7.

[75] BEIGY, H.—MEYBODI, M. R.: Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problems. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 14, 2006, No. 5, pp. 591–615, doi: 10.1142/S0218488506004217.

[76] TORKESTANI, J. A.—MEYBODI, M. R.: A Cellular Learning Automata-Based Algorithm for Solving the Vertex Coloring Problem. Expert Systems with Applications, Vol. 38, 2011, No. 8, pp. 9237–9247.

[77] ECONOMIDES, A. A.: Real-Time Traffic Allocation Using Learning Automata. 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Vol. 4, 1997, pp. 3307–3312, doi: 10.1109/ICSMC.1997.633133.

[78] FONTANA, R.—PISTONE, G.—ROGANTIN, M. P.: Classification of Two-Level Factorial Fractions. Journal of Statistical Planning and Inference, Vol. 87, 2000, No. 1, pp. 149–172, doi: 10.1016/S0378-3758(99)00173-1.

[79] VIJAYALAKSHMI, R.—VASUDEVAN, V.: Static Batch Mode Heuristic Algorithm for Mapping Independent Tasks in Computational Grid. Journal of Computer Science, Vol. 11, 2015, No. 1, pp. 224–229, doi: 10.3844/jcssp.2015.224.229.

[80] XHAFA, F.—BAROLLI, L.—DURRESI, A.: Batch Mode Scheduling in Grid Systems. International Journal of Web and Grid Services, Vol. 3, 2007, No. 1, pp. 19–37, doi: 10.1504/IJWGS.2007.012635.

[81] CALHEIROS, R. N.—RANJAN, R.—BELOGLAZOV, A.—DE ROSE, C. A. F.—BUYYA, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience, Vol. 41, 2011, No. 1, pp. 23–50, doi: 10.1002/spe.995.

[82] BUX, M.—LESER, U.: DynamicCloudSim: Simulating Heterogeneity in Computational Clouds. Future Generation Computer Systems, Vol. 46, 2015, pp. 85–99, doi: 10.1016/j.future.2014.09.007.

[83] KONG, L.—MAPETU, J. P. B.—CHEN, Z.: Heuristic Load Balancing Based Zero Imbalance Mechanism in Cloud Computing. Journal of Grid Computing, Vol. 18, 2020, No. 1, pp. 123–148, doi: 10.1007/s10723-019-09486-y.

[84] HASHEM, W.—NASHAAT, H.—RIZK, R.: Honey Bee Based Load Balancing in Cloud Computing. KSII Transactions on Internet and Information Systems (TIIS), Vol. 11,

2017, No. 12, pp. 5694–5711, doi: 10.3837/tiis.2017.12.001.

[85] Wang, T.—Wei, X.—Tang, C.—Fan, J.: Efficient Multi-Tasks Scheduling Algorithm in Mobile Cloud Computing with Time Constraints. Peer-to-Peer Networking and Applications, Vol. 11, 2018, No. 4, pp. 793–807, doi: 10.1007/s12083-017-0561-9.

[86] Mathew, T.—Sekaran, K. C.—Jose, J.: Study and Analysis of Various Task Scheduling Algorithms in the Cloud Computing Environment. 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, 2014, pp. 658–664, doi: 10.1109/ICACCI.2014.6968517.

Ali Sarhadi received his B.Sc. and M.Sc. degrees in computer engineering in Iran, in 2004 and 2007, respectively. He is also Ph.D. candidate under Javad Akbari Torkestani in the Department of Computer Engineering, Islamic Azad University, Arak Branch, Arak, Iran.

Javad Akbari Torkestani received his B.Sc. and M.Sc. degrees in computer engineering in Iran, in 2001 and 2004, respectively. He also received his Ph.D. degree in computer engineering from the Science and Research University, Iran, in 2009. He joined the Computer Engineering Department at the Arak Azad University as Lecturer in 2005. Currently, he is Associate Professor at the Department of Computer Engineering faculty, Islamic Azad University, Arak Branch, Arak, Iran.