Economic Based Scheduling and Load Balancing Algorithms in Cloud Computing Using Learning Automata

Ali. Sarhadi[1] **and Javad. Akbari**[2] **Abbas. Karimi**[3]

[1,2,3]Department of Computer Engineering
Arak Branch
Islamic Azad University
Arak, Iran
sarhadi_ali@iau-malayer.ac.ir
j-akbari@iau-arak.ac.ir
akarimi.iau@gmail.com

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 demanders. 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 work 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 with certain criteria to achieve the mentioned goal. Applying these two parameters will create a fair basis for load balancing and scheduling. To implement the proposed approach, learning automata were used as an effective and efficient technique in reinforcement learning. In this paper, the input flow of tasks was considered in batches. Finally, to show the effectiveness of the proposed algorithms we conducted simulations using CloudSim toolkit and compared proffered algorithms with other existing algorithms, like BCO, MCT, MET And KPB. Proffered methods can balanced the Final Cost and Total Cost of machines.

Keywords: cloud computing, load balancing, learning automata,efficiency.

2000 Computing Subject Classification: 49K05, 49K15, 49S05.

1 Introduction

Cloud computing, the new form of on demand computing gains its popularity in the last few years(Antonopoulos and Gillam, 2010). 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 (Grossman, 2009). Cloud computing offers services with minimum cost compared to the setting up of the 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. Cloud computing has faced many challenges, including security, efficient load balancing, resource scheduling, scaling, QoS management, data center energy consumption, data lock-in and service availability, and performance monitoring (Wang, Von Laszewski, Younge, He, Kunze, Tao and Fu, 2010; Marston, Li, Bandyopadhyay, Zhang and Ghalsasi, 2011). Load balancing is playing a main role in maintaining the organization of Cloud computing. The main goal of load balancing mechanism is that 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 as well as it provides efficient resource utilization (Kaur and Luthra, 2012). Load balancing concerns distribution of resources among the users or requests in uniform manner so that no node is overloaded or sitting idle. Like in, all other internet based distributed computing, tasks load balancing is an important aspect in cloud computing (Khiyaita, El Bakkali, Zbakh and El Kettani, 2012). In the absence of load balancing provision, efficiency of some overloaded nodes can sharply degrade at times, leading to violation of SLA (Zhao, Calheiros, Gange, Ramamohanarao and Buyya, 2015). Therefore, providing the efficient load-balancing algorithms and mechanisms is a key to the success of cloud computing environments. Cloud computing approach, ideas and strategy need a new field for researching and developing the economy-based resource load balancing and scheduling system. Following load balancing thinking and Cost-effective load balancing strategies, this paper proposes a new cloud resource load balancing and scheduling algorithm, which can not only increase the uses of resources and system utilization, but also by importing new parameters trying, Both the resource providers and the consumer will receive the economic benefits only when the available resources are load balanced based on economic criteria and correctly scheduled. The mapping and scheduling algorithms can be classified into two categories, immediate mode and batch mode . In the immediate mode, when new tasks arrive, they are scheduled to VMs directly (Xhafa, Carretero, Barolli and Durresi, 2007). In the batch mode, tasks are grouped into a batch before being sent; this type is also called mapping events (Ghosh and Gupta, 1997). We have proposed a set of economic scheduling and load balancing algorithms based on Learning Automata for batch mode. proffered approach for scheduling and load balancing and selecting the best resource based on economic economic parameters is designed and will prevent the oversupply of a resource cost. This paper is organized as follows. Section 2 discusses related works. Section 3 presents Cloud Computing Load Balancing Model. Section 4 describes Economic Load Balancing Principle and Advantage. Section 5 describes Learning automata. Section 6 proffered Definitions and Hypotheses.section 7 describe Proposed Algorithms.final section provides Experiments and Assumption, and describes comparison between proposed algorithms and existing ones.

2 Related work

Traditional load balancing algorithms for distributed platforms such as grids, and clouds, focus in minimizing the execution time or minimum makespan without considering economic parameter (Chawla and Bhonsle, 2012; Singh and Chana, 2016). Load balancing problems are extended to many forms of constraints and environment settings (Fang, Wang and Ge, 2010; Guo, Zhao, Shen and Jiang, 2012). Some static (Henzinger, Singh, Singh, Wies and Zufferey, 2011; Shah and Farik, 2015) and dynamic algorithms (Lee, Wang and Zhou, 2011; Zhang and Zhou, 2017; Kumar and Sharma, 2020) has been presented in the last few years.(Dasgupta, Mandal, Dutta, Mandal and Dam, 2013) Proposed genetic based algorithm based scheduling mechanism for load balancing challenge. This algorithm selects the low loaded VMs for job transfer. For time-critical parallel applications, the IaaS Cloud Partial Critical Paths (IC-PCP) method for deadline-constrained applications has been proposed on heterogeneous cloud environments (Li, Qiu, Ming, Quan, Qin and Gu, 2012). In (Al Nuaimi, Mohamed, Al Nuaimi and Al-Jaroodi, 2012), a massive study on the scheduling and load balancing algorithms of cloud computing was proposed with the objective of performance and makespan minimization. However, financial cost and economic criteria is another important parameter in the cloud computing load balancing and scheduling mechanism. For cost-critical parallel applications, cost-aware scheduling algorithms have been proposed for minimizing execution cost or satisfying the budget constraint on heterogeneous systems (Sarhadi and Meybodi, 2010; Selvarani and Sadhasivam, 2010; Mansouri and Javidi, 2019). However, very few papers have targeted cloud computing environment and designs for minimizing the schedule length of budget constrained applications. In (Verma and Kaushal, 2012), a heuristic algorithm of deadline early tree was presented. It minimized the cost of deadline constrained applications without considering the communication time between tasks. (Lin and Wu, 2013) Proposed a critical-greedy (CG) algorithm to minimize the end-to-end delay of budget constrained parallel applications. In this work, the CG algorithm defines a global budget level (GBL) parameter and pre assigns tasks with the budget level execution cost. (Li et al., 2012) Discussed task scheduling and resource allocation problem for implementing tasks in IaaS clouds; a novel provisioning and scheduling algorithm is presented to execute tasks under budget constraint while reducing the slowdown. The experimental results illustrated that their proposed algorithms minimized the slow-down in execution time to 70 percent. (Rodriguez and Buyya, 2017) introduced a scheduling algorithm which optimized the task workflow execution time regarding the budget constraint. The experiments proved that this scheduling algorithm has faster execution time and more effective performance on cloud resources where their proposed algorithm is capable of generating high-quality schedules to meet the budget constraint. (Arabnejad and Barbosa, 2017) Considered the profit of provider and they proposed Multi-QoS Profit-Aware scheduling algorithm (MQ-PAS) to assign each job with its budget priority. The performance evaluation results showed that the MQ-PAS increased the profit of provider and achieved success rates of completion jobs. There is alots of work on the use of learning automata in clud computing task scheduling. In (Hosseinzadeh et al., 2019) a dynamic learning automata (LA) based algorithm is proposed to solve the task scheduling problem in the cloud environment. (Sahoo, Sahoo and Turuk, 2019) proposed a novel learning automata-based scheduling framework for deadline sensitive tasks in the cloud.(Krishna, Misra, Nagaraju, Saritha and Obaidat, 2016; Qavami, Jamali, Akbari and Javadi, 2017) proposed other methods for task scheduling in cloud computing. Other AI thechnics have been used in cloud computing task scheduling which are mentioned in (Javanmardi, Shojafar, Amendola, Cordeschi, Liu and Abraham, 2014; Kaur and Kinger, 2014; Shojafar, Javanmardi, Abolfazli and Cordeschi, 2015; Zuo, Shu, Dong, Zhu and Hara, 2015).In (Leitão, Barbosa, Funchal and Melo, 2020; Rodrıguez and Corchado, 2020) a new mechanism based on multi-agent systems is presented and (Precup, Teban, Albu, Borlea, Zamfirache and Petriu, 2020; Yuhana, Fanani, Yuniarno, Rochimah, Koczy and Purnomo, 2020) applies an incremental online identification algorithm to develop a set of evolving fuzzy models (FMs). (Zamfirache, Precup, Roman and Petriu, 2021) presents a new Reinforcement Learning (RL)-based control approach that uses the Policy Iteration (PI) and a metaheuristic Grey Wolf Optimizer (GWO) algorithm to train the Neural Networks (NNs). In (Kumar and Dinesh, 2012; Shojafar et al., 2015) an optimized algorithm based on the Fuzzy optimization was proposed which makes a scheduling decision by evaluating the entire group of task in the job queue. In (Xie, Zhu, Wang, Cheng, Xu, Sani, Yuan and Yang, 2019) a new Bee Swarm optimization algorithms called Bees Life Algorithm (BLA) was proposed which applied to efficiently schedule computation jobs among processing resources onto the cloud datacenters.

3 Cloud Computing Load Balancing Model

Load balancing is the process of improving the performance of a parallel and distributed system through a Reassign of load among the processors or nodes (Panwar and Mallick, 2015). Load balancing is described in (Jain and Gupta, 2009) as follows “In a distributed network of computing hosts, the performance of the system can depend crucially on dividing up work effectively across the participating nodes”. 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. The model of balancing is shown in Fig. 1, where we can see the load balancer receives users’ requests and runs load-balancing algorithms to distribute the requests among the Virtual Machines (VMs). The load balancer decides which VM should be assigned to the next request. The data center controller is in charge of task management. Tasks are submitted to the load balancer, which performs load-balancing algorithm to assign tasks to a suitable VM. VM manager is in charge of VMs. Virtualization is a dominant technology in cloud computing. The main objective of virtualization is sharing expensive hardware among VMs.

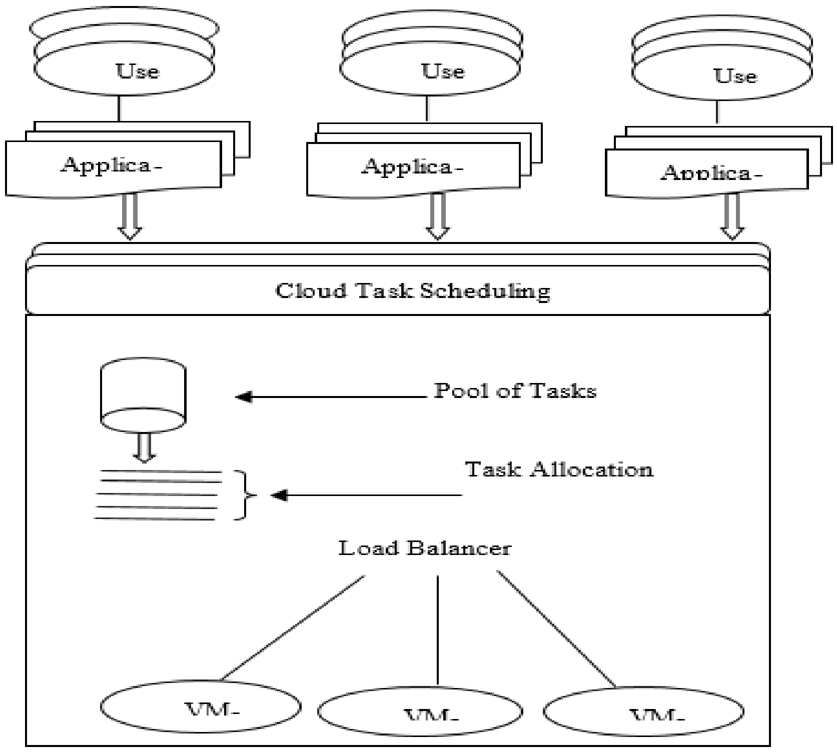


Figure 1: Simple model of load balancing

3.1 Economic Load Balancing Principle and Advantage

In distributed computing with a lot of different participants and contradicting requirements the well-known efficient approaches are based on economic principles (Chen and Lu, 2008; Sarhadi and Yousefi, 2011). Here, we will mainly discuss about economy-based cloud resource load balancing and task scheduling strategy. To address these issues, the economy-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(Buyya 2002). Buyya proposed distributed computational economy-based framework, called the Grid Architecture for Computational Economy (GRACE) (Buyya, Abramson and Giddy, 2001). This economic-based framework offers an incentive to resource owners for contributing and sharing resources. Listed researches addressed the idea of applying economic models to the scheduling task. The efficiency of this approach in terms of response and wait time minimization as well as utilization is evaluated.(Buyya et al., 2001) developed three heuristic scheduling algorithms for cost, time, and time-variant optimization strategies that support deadline and budget constraints. Our approach in this paper is to apply a new method to cloud computing resource load balancing based on economic criteria with learning automata that we will discuss in more detail later.

4 Learning automata

Learning Automata are adaptive decision-making devices operating on unknown random environments. A Learning Automaton has a finite set of actions and each action has a certain probability (unknown to the automaton) of getting rewarded by the environment of the automa-

ton. 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 be made to result in selection of the optimal action. Fig. 2 illustrates how a stochastic automaton works in feedback connection with a random environment. Learning Automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA) (Narendra and Thathachar, 1974). In the following, the variable structure learning automata which will be used in this paper is described.
Learning Automata are well suited for systems with noisy and incomplete information about the Environment in which use it (Agache and Oommen, 2002; Lakshmivarahan, 2012; Lakshmivarahan, 1981). The Environment is generally stochastic and the Learning Automata lacks prior knowledge as to which action is the optimal one. Stochastic Learning Automata, which are probabilistic finite state machines, attempt to solve this problem by choosing an initial action randomly, and then updating the action probabilities based on the response received.

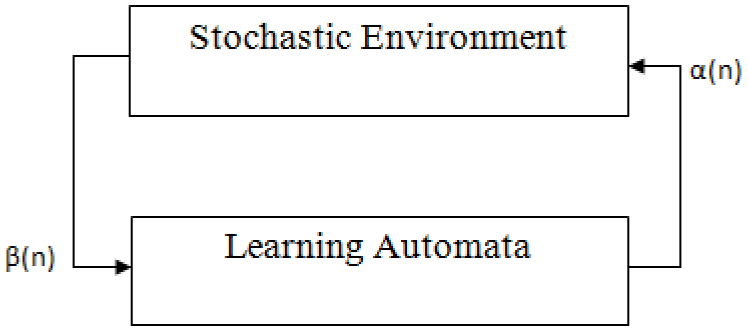


Figure 2: The interaction between learning automata and environment

A VSLAis a quintuple (α,β ,p,T(α,β ,p)) , where α,β are an action set with sactions, an environment response set and the probability set pcontaining s probabilities, each being the probability of performing every action in the current internal automatonstate, respectively. 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]. 6such 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 refer 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 (4.1) and (4.2). Let action be performed then:

$$P_j(n+1) = P_j(n) + \beta(n)[b/(r-1) - bp_j(n)] - [1-\beta(n)]\alpha p_j(n) \tag{4.1}$$

$$P_i(n+1) = P_i(n) - \beta(n)bp_i(n) + [1-\beta(n)]\alpha[1-P_i(n)] \tag{4.2}$$

Where a and bare reward and penalty parameters. When a=b, the automaton is called, L_{RP} if b=0the 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 (Asnaashari and Meybodi, 2007; Eraghi, Torkestani and Meybodi, 2011).

5 Definitions and Hypotheses

To analyze the performance of proposed approach, the cost of executing each task 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 bedescribed 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 $M * N$ lements. Therefore, all of the ECC elements range in $[1, \omega_b * \omega_r)$.

6 Learning Automaton Model for Task Mapping

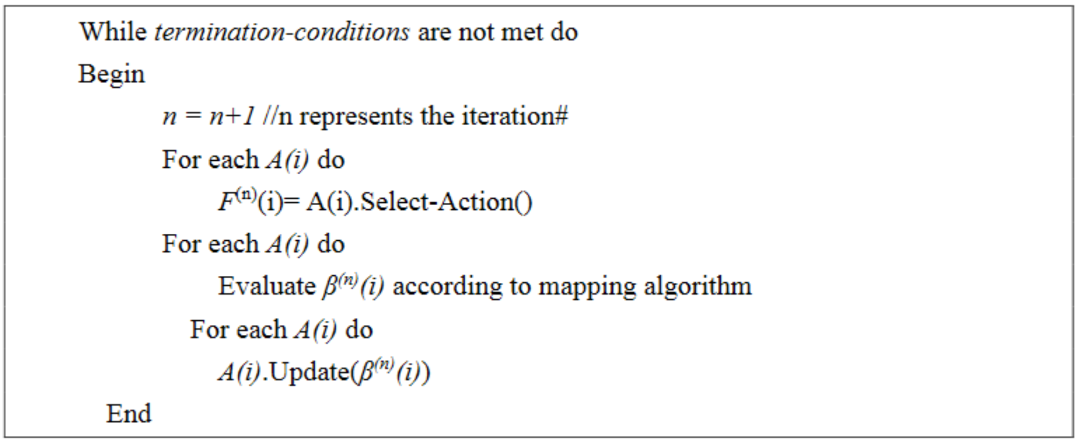
The general map $Q(i) = j$ was defined from the domain of tasks $i = 1 \ldots M$ to the domain of machines $j = 1 \ldots N$ and its general procedure is shown in Fig. 3. The required cost of executing all the tasks assigned to a machine in the n_{th} iteration is called the Total Cost (TC) of that machine, indicated by $C^{(n)}(j)$. It is determined through the (6.1):

$$c^{(n)}(j) = \sum ECC(k, j), j = Q(k) 1 \leq k \leq M \tag{6.1}$$

The maximum value of $c^{(n)}(j)$ on $1 \leq j \leq N$ is called the Final Cost(FC)in the n_{th} iteration. It is shown by $F^{(n)}$ .In the proposed model, a Learning Automata was assigned to each task T_i of the meta-task. Each Automata is shown by $(\alpha(i), \beta(i), A(i))$ since tasks can be assigned to each of N machines, Automata share the sameactions. Hence, there can be $\alpha(i) = m_1, m_2, \cdots, m_{n-1}$ 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 3.

The following reasons demonstrate the effectiveness of using a learning automata in our proposed approach:

- The learning automata have ability to perfect adapt themselves to environmental changes. This attribute is very advisable for use in cloud computing environments with a high degree of inconsistency.
- learning automata impose a small amount of communication and computational costs in collaborating with the environment. This attribute determins learning automata as a suitable substitute for use in environments such as cloud computing with energy constraints and bandwidth than the other models.
- collaborating with each other, the learning automata are able to perfectly model the distribution of cloud computing environments and in addition, simulate the changing behavioral



```
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 3: General proposed procedure used by Learning Automata mapping algorithms

patterns of the nodes in relation to each other and with the environment considering their learning ability and their adaptability to the environment.

- collaborating with each other, the learning automata are able to converge to the global optimal answer only based on the local decisions when solving optimization problems. Therefore, learning automata-based algorithms can be considered as an appropriate choice for the cloud computing as they can resolve the slag resulted from aggregation or dissemination of information in centralized algorithms.

7 Proposed Algorithms

In this section, two algorithms based on profitability for cloud users are presented which, Learning automata has been used to implement them.

7.1 Cost-Efficient Load-Balancing Profit-Based Algorithms

In the profit-based algorithms, an appropriateness index of actions is based on the Total Cost (TC) and Final Cost (FC) of each machine. In other words, an action is appropriate if it reduces the Total Cost of a selected machine by automata or reduces the Final Cost compared with the previous iteration.The profit-based algorithm includes two classes:
7.1.1 The profitability-based with specified rewarding algorithm(PS)
7.1.2 The Profitability-Based with Random Rewarding Algorithm(PR)

7.1.1 The profitability-based with specified rewarding algorithm(PS)

In this algorithm, abbreviated as PS, the appropriateness of an action is based on a comparison of the Total Cost with the previous iteration. 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 $(n-1)_{th}$ 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 $(n-1)_{th}$ iteration. Therefore, given the Final Cost and Total

Table 1: Probabilities of Rewards Allocated to the Nine Possible Cases

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	$\beta_F + \beta_C = 1$

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 appropriateness of the taken action by the automata. The environmental response to automata A(i) in the n_{th} iteration is determined in the following way:
This value indicates the appropriateness of the automat's action . (7.1) is employed to determine the environmental response to automata A(i) in the n_{th} iteration:

$$\beta^{(n)}(i) = 1 - (f(F^{(n-1)}, F^n)\beta_F + f(C^{(n-1)}(Q^{(n-1)}(i), C^n(Q^n(i)))\beta_C) \tag{7.1}$$

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. $f(x, y)$ can be defined as (7.2).

$$f(x,y) = \begin{cases} 0 & x < y \\ 1/2 & x = y \\ 1 & x > y \end{cases} \tag{7.2}$$

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 Table1, 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. otherwise, it will receive no reward (and will be fined).
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 automata is rewarded if the total cost of the selected machine issmaller 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 ($\beta_F > \beta_C$).

7.1.2 The Profitability-Based with Random Rewarding Algorithm(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 and total costs 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 nthiteration 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 n_{th} 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 and total costs 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 (7.3) for automata A(i) in the n_{th} iteration:

$$p^{(n)}(i) = f(F^{(n-1)}, F^n)P_F + f(C^{(n-1)}(Q^{(n-1)}(i)), \theta^n(Q^n(i)))P_C \tag{7.3}$$

In (7.3), $P_F + P_C = 1$, $P_F \neq 0$, $P_C \neq 0$, 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. According to (5), f(x,y) is defined. The environmental response to automata A(i), $\beta^{(n)}(i)$ is determined through the (7.4) in the n_{th} iteration:

$$\beta^{(n)}(i) = I(1 - p^{(n)}(i)) \tag{7.4}$$

In (7.4), I(q) is an indicator function(Kenny et al., 2003), 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.

7.2 Cost-Efficient Load-Balancing Threshold-Based Algorithms

Threshold-based algorithms benefit from a threshold in addition to the total cost and/or final cost in order to determine the appropriateness of automata's actions. If the total cost and/or final cost are lower than the threshold, automata's actions are evaluated appropriate; otherwise, they are considered inappropriate. A problem with 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 designated machine or the final cost of consecutive iterations might remain constant in this case. Therefore, profit-based algorithms interpret this case as inappropriate and cease to reward the automaton completely; as a result, they prevent the automaton from reaching the appropriate mapping. To avoid this problem, threshold-based algorithms use

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

Final Cost	Total Cost	Probability of receiving rewards
D	D	$p_f + p_c = 1$
D	U	$p_f + 1/_2 p_c$
D	I	p_f
U	D	$1/_2 p_f + p_c$
U	U	$1/_2 p_f + 1/_2 p_c$
U	I	$1/_2 p_f$
I	D	p_c
I	U	$1/_2 p_c$
I	I	0

a threshold to determine reward or penalty inaddition to considering the total cost or final cost. Threshold-based cost-effective load-balancing algorithms are divided into two categories:
7.2.1 The threshold-based cost-effective load-balancing algorithm with specific rewarding(TS)
7.2.2 The threshold-based cost-effective load-balancing algorithm with random rewarding(TR)

7.2.1 Threshold-Based Cost-Effective Load-Balancing Algorithm with specific Rewarding(TS)

An action is rewarded in proportion to its appropriateness in this algorithm in the same way as the PS algorithm. These 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 $(n-1)_{(th)}$ iteration. Likewise, the total cost of the selected machine by automaton A(i) in the $n_{(th)}$ iteration might be greater than, smaller than, or equal to that of the machine selected by automata in the $(n-1)_{(th)}$ 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 machine 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 as (7.5).

$$\beta^{(n)}(i) = \begin{cases} 1 - (f_l(F^{(n-1)}, F^n)\beta_F + f_l(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))\beta_C) & F^n < T \\ 1 - (f_g(F^{(n-1)}, F^n)\beta_F + f_g(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))\beta_C) & F^n \geq T \end{cases} \tag{7.5}$$

Where $\beta_F + \beta_C = 1$
In 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 machine is smaller than the total cost of the selected machine in the previous iteration. Moreover, T is the threshold. Functions $f_l(.,.) \in \{0, 1\}$ and $f_g(.,.) \in \{0, 1/_2, 1\}$, 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 foreach meta task, and then the resultant final cost is used as the threshold. The proposed algorithm is called TS in this model. When the final cost is smaller than the threshold in the TS algorithm, the equality of smallness (of the total cost of the selected machine or the final cost in comparison with the previous iteration) is regarded as the appropriate solution, whereas the greatness (of the total cost of the selected machine 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 (7.6) , (7.7).

$$f_g = \begin{cases} 0 & x < y \\ 1/2 & x = y \\ 1 & x > y \end{cases} \tag{7.6}$$

$$f_l = \begin{cases} 0 & x < y \\ 1 & x \geq y \end{cases} \tag{7.7}$$

7.2.2 Threshold-Based Cost-Effective Load-Balancing Algorithm with Random Rewarding (TR)

Called TR, this algorithm acts like the TS 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 $(n-1)_{(th)}$ iteration. Likewise, the total cost of the selected machine by automaton A(i) in the $n_{(th)}$ iteration might be greater than, smaller than, or equal to the total cost of the selected machine by automata in the $(n-1)_{(th)}$ 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 machine 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 as (7.8) for automaton A(i) in the $n_{(th)}$ iteration.

$$p^{(n)}(i) = \begin{cases} (f_l(F^{(n-1)}, F^n)P_F + f_l(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))P_C) & F^n < \Omega \\ (f_g(F^{(n-1)}, F^n)P_F + f_g(C^{(n-1)}(Q^{(n-1)}(i)), C^n(Q^n(i)))P_C) & F^n \geq \Omega \end{cases} \tag{7.8}$$

In (7.8), P_F is the probability of rewarding if the Final Cost is smaller than the previous iteration, and P_D is the probability of rewarding if the designated machine's Makespan is smaller than that of the selected machine in the previous iteration $(P_F + P_C = 1)$. Moreover, Ω shows the threshold. Functions $f_l(.,.) \in \{0, 1/_2, 1\}$ and $f_g(.,.) \in \{0, 1/_2, 1\}$ 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_o determines the reward and penalty policy when the Final Cost is greater than the threshold, and f_b 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 resultant Final Cost is used as the threshold. The environmental response to automata A(i) in the n_{th} iteration is calculated in the (7.9):

$$\beta^{(n)}(i) = I(1 - p^{(n)}(i)) \tag{7.9}$$

In (7.9), I(q) As described above is an indicator function, which returns 1 and 0 for the probabilities of qand 1-q, respectively. When the final cost is smaller than the threshold in the TR1 algorithm, the equality or smallness (of the total cost of the selected machine or the final cost in comparison with the previous iteration) is regarded as the appropriate 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_g and f_l are defined in previous section .When the final cost is smaller than the threshold in the TR algorithm, only the greatness (of the total cost of the selected machine 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 machine or the final cost in comparison with the previous iteration) are considered inappropriate.

8 Experiments and Assumption

Cloud computing provides inexpensive, scalable and omnipresent computing over distributed networks. It is based on reliable, secure, fault-tolerant, sustainable and scalable environment for providing services everywhere and all the time. 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. In this section, CloudSim architecture, configuration of network and the experiment results are discussed(Calheiros, Ranjan, Beloglazov, De Rose and Buyya, 2011).

8.1 CloudSim Architecture

CloudSim Support modeling and simulation of large scale Cloud computing environments, The multi-layered architecture of CloudSim(Sareen and Singh, 2016) is shown in Fig.4.
1) Network Layer: This layer of CloudSim has responsibility to make communication possible between different layers. This layer also identifies how resources in cloud environment are places and managed.

2) Cloud Resources: This layer includes different main resources like DataCenters, cloud co-ordinator (ensures that different resources of the cloud can work in a collaborative way) in the cloud environment.
3) Cloud Services: This layer includes different service provided to the user of cloud services. The various services of clouds include Information as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).
4) User Interface: This layer provides the interaction between user and the simulator.

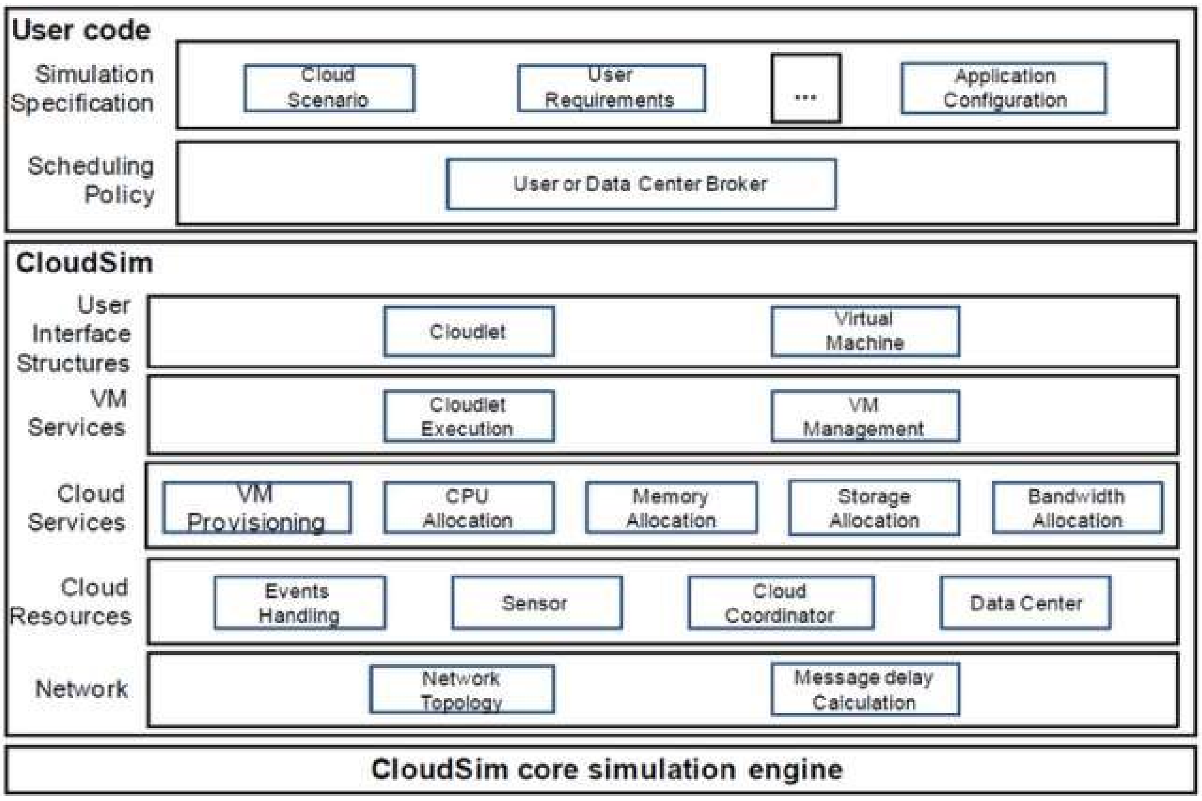


Figure 4: cloudSim architecture

8.2 CloudLet Creation

A cloudlet is presented to run the proposed scheduling strategy. This cloudlet implements only the job scheduler. However, some components of the cloud are implemented as a primary stage for executing the scheduler.

8.2.1 Data center creating

In order to run scheduling on the cloud one must perform some abstraction of the real cloud computing system. Creating and modeling the largest computing environment, the cloud, at least requires an Appropriate abstraction of real cloud components such as Data Center, Host, Data Center Broker, and Virtual Machine. Moreover, most important packages of the CloudSim are known as: “org.cloudbus.clousim”, “org.cloudbus.clousim.core” and some of its packages are not considered to provide the software appropriate to the proffered scheduling methods. The main idea of using different cloud environments is to test the proffered job scheduler in different size of cloud environment, such as, small scale and large scale cloud computing environments. In this paper, the scheduling methods and algorithms are implementing and evaluated in cloud simulator layer and User Code layer. The indCloudletToVM() routine in DatacenterBroker package of CloudSim simulator can allocate a job to a appropriate virtual machine and execute it. To increase the probability of allocation of appropriate virtual machine to specific jobs, Datacenter-Broker package of CloudSim were used.

8.3 Simulation Configuration

The main Dataset used in this paper is Google Cloud Jobs Dataset(GOcJ) that the most important information we want from this Dataset is given in Tables 3 and 4. Table 3 shows complete information on cloud resources which 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 that describe in above section are realized by applying packages in the CloudSim. Furthermore, “Datacenter” is the main classes applied to simulation of cloud environment. It is a cloud resource whose host list is virtualized. The CloudSim was configured with three DataCenter that includes 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); Tijuana). Table 3 presents the specifications of the configured simulation environment and resource features. Each host has between 10 and 20 virtual machines.

Table 3: cloud computing resource model

Data center	Host	RAM Size (MB)	MIPS	BandWith (MBPS)	Extended Memory (MB)
DC0	H1	4096	5100	300	100000
	H2	1024	4600	300	300000
	H3	2048	2500	300	1000000
DC1	H1	2048	8000	300	750000
	H2	512	6140	300	250000
	H3	4096	2570	300	125000
DC2	H1	1024	6500	300	300000
	H2	4096	1250	300	800000
	H3	512	3540	300	450000
	H4	2048	2700	300	700000

The execution unit of the instructions is based on million instruction per second (MIPS), RAM and extended memory units is based on mega byte (MByte), network bandwidth is introduced in Mega Byte Per Second (Mbps). The processing powers of each processor are specified. so, job scheduling issue must be assigned jobs to hosts and achieved 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. Tables 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 combination of data intensive and computation Intensive jobs.

Table 4: job features and specifications

Job Classes	Length(MI)	File Size	Output Size	Execution time (Normal distribution)	Execution Cost
J0	40000	2500	100	N(5,10)	Based on ECC Matrix
J1	37000	3000	120	N(5,10)	Based on ECC Matrix
J2	24000	7800	230	N(5,10)	Based on ECC Matrix
J3	18000	4300	180	N(5,10)	Based on ECC Matrix
J4	42000	11000	400	N(5,10)	Based on ECC Matrix
J5	36000	6500	230	N(5,10)	Based on ECC Matrix
J6	22000	3500	320	N(5,10)	Based on ECC Matrix
J7	31000	7600	160	N(5,10)	Based on ECC Matrix
J8	19000	10000	220	N(5,10)	Based on ECC Matrix
J9	24000	9600	290	N(5,10)	Based on ECC Matrix
J10	31000	4300	310	N(5,10)	Based on ECC Matrix

8.4 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, are trying to formed the cloud system to execute their jobs. In order to compare the impact of proffered model on the task scheduling and load balancing algorithms in the cloud environment, efficiency of four well- known scheduling algorithms in the forms of different case studies named BCO, MCT, MET and KPB with proposed algorithms based on specific parameters that will be explained below, are simulated . Due to the popularity of these scheduling algorithms, their code is not mentioned, but the description about the algorithms is provided in the following:

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 that require to resource(Buyya et al., 2001).

In MCT (Minimum Completion Time) algorithm assigns tasks to VMs or resources based on

the best predictable completion time for that task in random order. Each task is assigned to the VM or resource that has earliest completion time. More details are mentioned in (Llwaah, Thomas and Cala, 2015).
In MET (Minimum Execution Time) algorithm assigns tasks to VMs or resources based on the best predictable completion time for that task without regard to resource availability. More details are mention in(Hemamalini, 2012).
The KPB (k-percent best) heuristic considers only a subset of machines while mapping a task. The subset is formed by picking the k-percent best machines based on the execution times for the task. The task is assigned to a machine that provides the earliest completion time in the subset(Masdari, ValiKardan, Shahi and Azar, 2016).
Different ECC matrix stability was used to take more aspects of realistic mapping situations. An ECC matrix is said to be inconsistent if the ECC matrices are kept in the unordered, random state in which they were created. The ECC matrix announce consistent characteristics if a machine j executes any task i cheaper than machine k, then machine j executes all tasks cheaper than machine k. Termination condition is occurs when, no change in Final Cost is made for 2000 consecutive repeat, or number of repeat be more than 200000. The scheduling and load balancing policies can be categories into two method, immediate mode and batch mode. In the batch mode, jobs are gathered into a set, called metatask, which is further examined for mapping at prescheduled times. In this paper a set of load balancing algorithms based on Learning Automata for batch mode was introduced. The performance of job scheduling and load balancing methods was evaluated by different metrics such as, error rate, Final Cost, Total Cost, waiting time, imbalance degree and efficiency. Error rate: The error rate parameter in load balancing algorithms are regulated with regard to each allocation, for instance in failure cases of allocation, or in cases demanding reallocation. Total Cost: The required cost of executing all the tasks assigned to a machine in the nth iteration is called the Total Cost (TC) of that machine, indicated by $C^{(n)}(j)$:

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

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.
Waiting time: It is defined as the time that a process waits from its submission to completion in the queues.
imbalance degree: It is calculated based on the equation 8

$$\begin{gathered}
c^n_{avg} = \frac{c^n(1) + c^n(2) + ... + c^n(M)}{M} \\
c^n_{min} = min\{c^n(1) + c^n(2) + ... + c^n(M)\} \\
c^n_{max} = max\{c^n(1) + c^n(2) + ... + c^n(M)\} \\
imbalance\_d = \frac{c^n_{max} - c^n_{min}}{c^n_{avg}}
\end{gathered} \tag{8.1}$$

Efficiency: For the first time, we present an economic definition of efficiency in cloud environment based on equ 9.The denominator in equation 9 indicates the total cost obtained for the execution of all the assigned tasks in cloud environment. So, for each of algorithms a higher value represents better result. M is number of machines.

$$Efficiency = \frac{M}{\sum_{j=1}^{N} \sum_{k=1}^{M} ECC(k,j)} \quad (8.2)$$

8.4.1 Total Cost (TC) and Final Cost (FC) Simulation

The Total Cost (TC) is one of the most important parameters that 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. It can be defined as the total monetization by a machine for executing all allocated jobs. According to Fig.5, Fig.6 all proposed algorithms outperform than Existing algorithms in consistent and inconsistent environment. The results also show that algorithm PS performs better than others in inconsistent environment and in consistent environment TS performs better than others. Fig.7 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 Fig.8 and Fig.9, the results obtained for comparing the proposed algorithms with the existing algorithms are precisely similar to the Total Cost (TC) parameter. Fig.10 also shows that with versus 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.

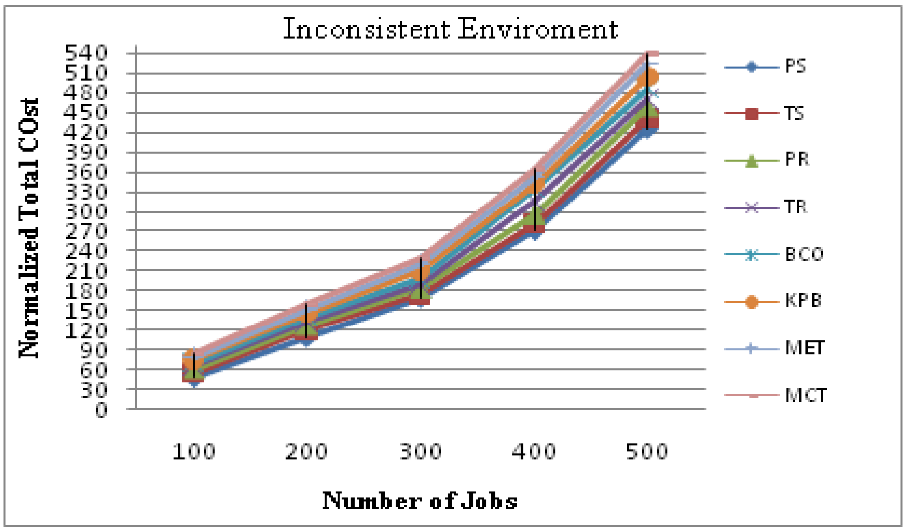


Figure 5: Total Cost versus number of jobs in proposed and existing algorithms in inconsistent environment

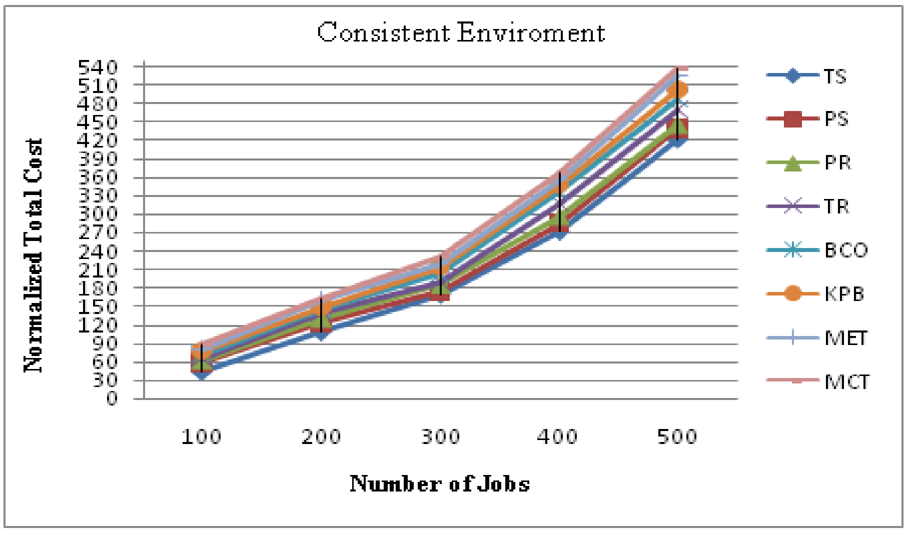


Figure 6: Total Cost versus number of jobs in proposed and existing algorithms in consistent environment

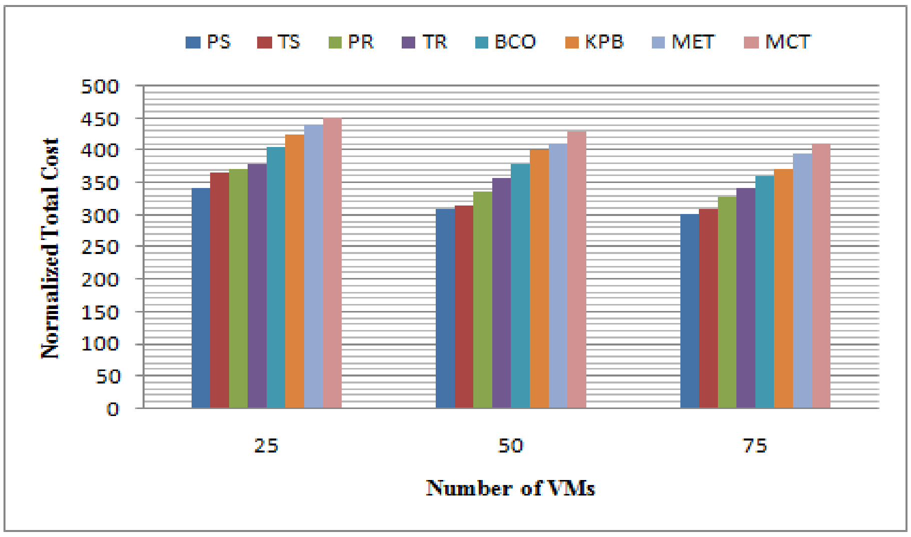


Figure 7: Total Cost versus number of VMs in proposed and existing algorithms in consistent environment

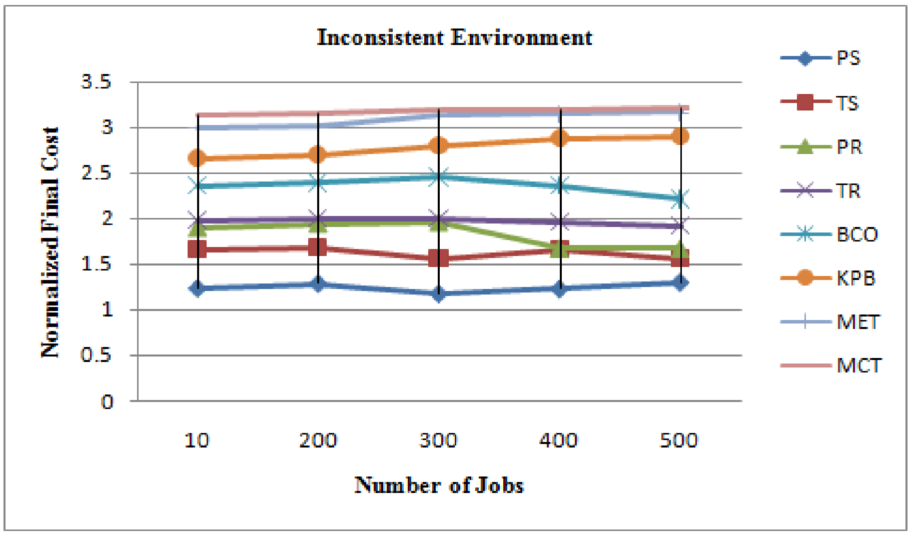


Figure 8: Final Cost versus number of jobs in proposed and existing algorithms in inconsistent environment

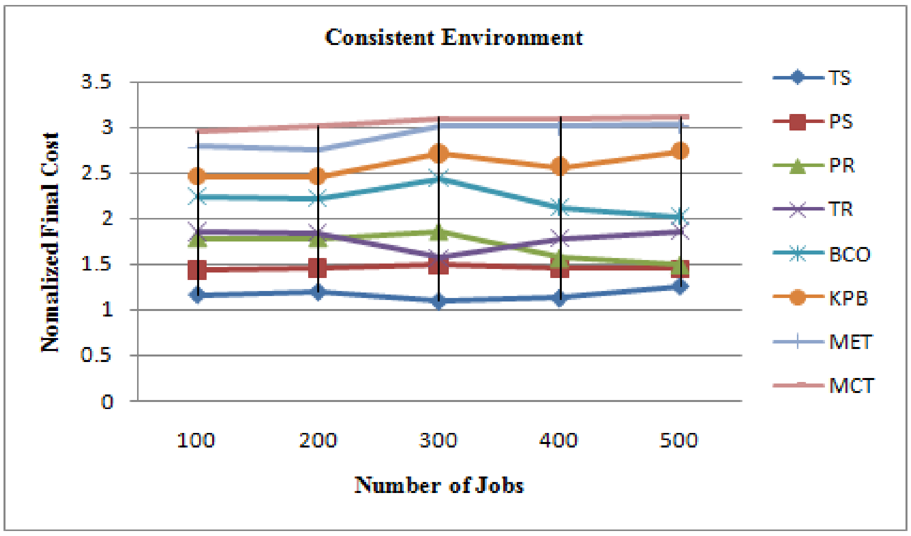


Figure 9: Final Cost versus number of jobs in proposed and existing algorithms in consistent environment

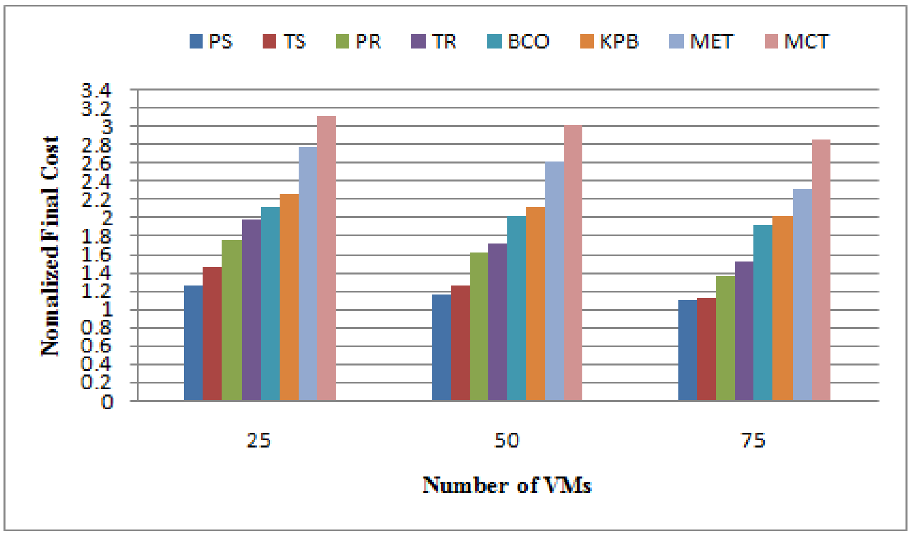


Figure 10: Final Cost versus number of VMs in proposed and existing algorithms in consistent environment

8.4.2 waiting time

Waiting time is the total time spent by the job in the ready state waiting for allocate to a machine and executed. Average of waiting time required by every job is presented in Fig.11. It shows that average waiting time for each algorithm has increased by increasing the number of jobs. Usually, load balancing policies lead to a reduction in waiting time. The reason for this is that in traditional load balancing algorithms is often the load balancing is based on time criteria, But in proposed algorithms the criterion of load balancing is the cost of resources and preventing excessive increase in the price of a resource, so unlike traditional approaches to load balancing, the waiting time of the proposed algorithms is longer than traditional models. According to Fig.11 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.

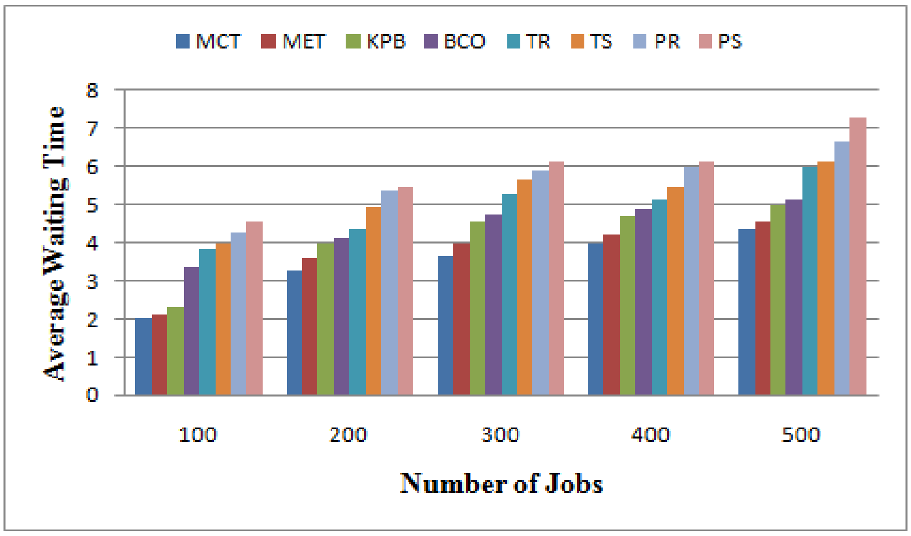


Figure 11: Average waiting time versus number of jobs comparisons

8.4.3 Imbalance Degree

Degree of imbalance is one of the most important parameters for the evaluation of the proposed algorithm, which is exactly equivalent to the load balancing. Imbalance degree indicates whether the jobs are distributed monotonously among virtual machines or not. The imbalance degree (ID) is a measure of the amount of load distribution amongst the VMs regarding their execution competencies. The small value of DI indicates that the load is more balanced. Traditional load balancing algorithms give a discussion on the imbalance degree (ID) by presenting its relationship with makespan, resource utilization and task's completion time. However, in this paper, due to the design of economic criteria in scheduling and load balancing algorithms, the imbalance factor is also based on economic criteria parameters which are shown in Formula 10. In the designed formula, the closer the difference between the maximum and minimum Total Cost (TC) of virtual machines to the average Total Cost (TC), will increase degree of imbalance. According to Fig.12 proposed algorithms outperform existing ones. between the proposed algorithms, the PS algorithm has the best results.

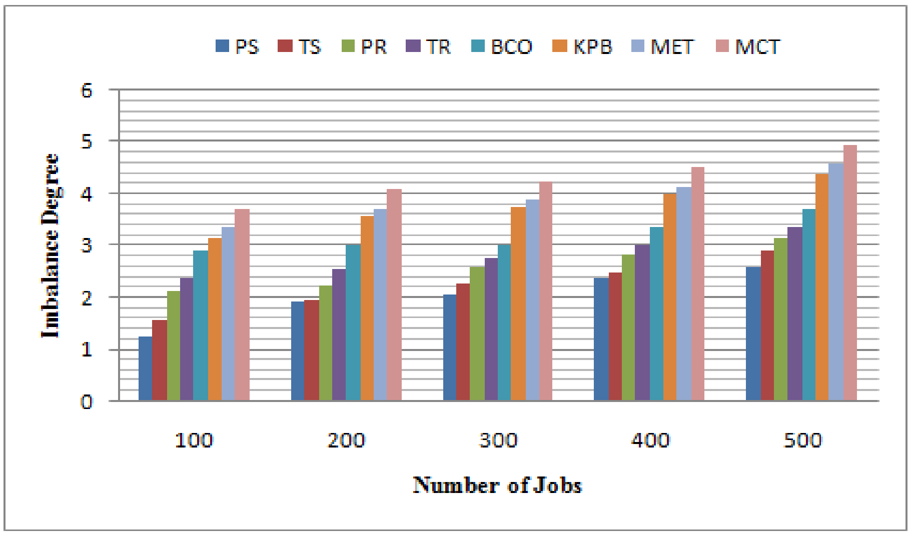


Figure 12: Imbalance Degree versus number of jobs comparisons

8.4.4 Efficiency

Other most important parameter in the evaluation of the proposed algorithms and existing algorithms is efficiency. Based on the purposes of this paper, we present a new definition of efficiency based on economic criteria that illustrated in equ 9. To recognize the quality of economic distribution of Jobs among VMs, So that jobs can be executed at a lower cost. So a higher value in formula 11 represents better result. As shown in the Fig.13 the proposed algorithm PS outperform other algorithm and also one of the existing algorithms called BCO has better result than proposed algorithm TR. Also, with increase number of machines, efficiency has increased.

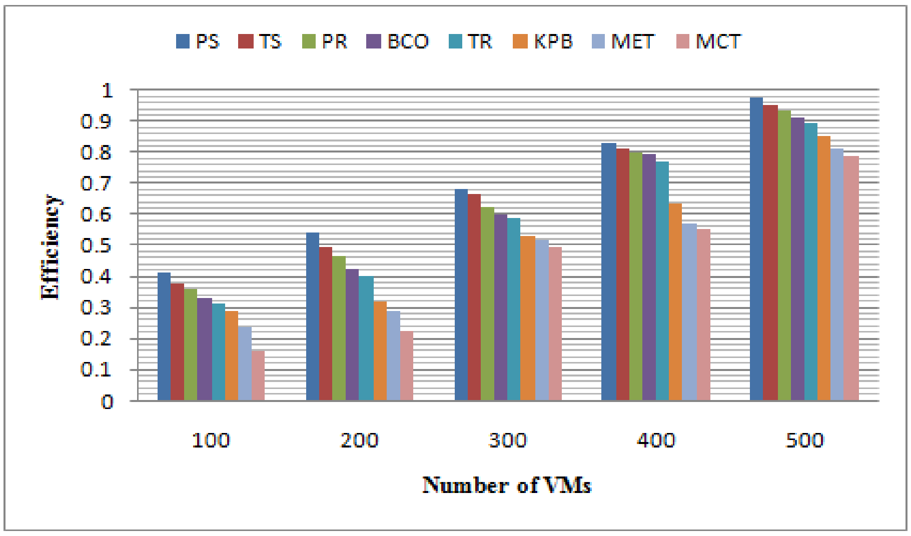


Figure 13: Efficiency versus number of jobs in proposed and existing algorithms

8.4.5 Error rate

The error rate is the number of jobs not completed successfully for any reason, to the total number of jobs submitted. As shown in Fig.14 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 due to many iterations and reallocation job to VMs to converge to desired response.

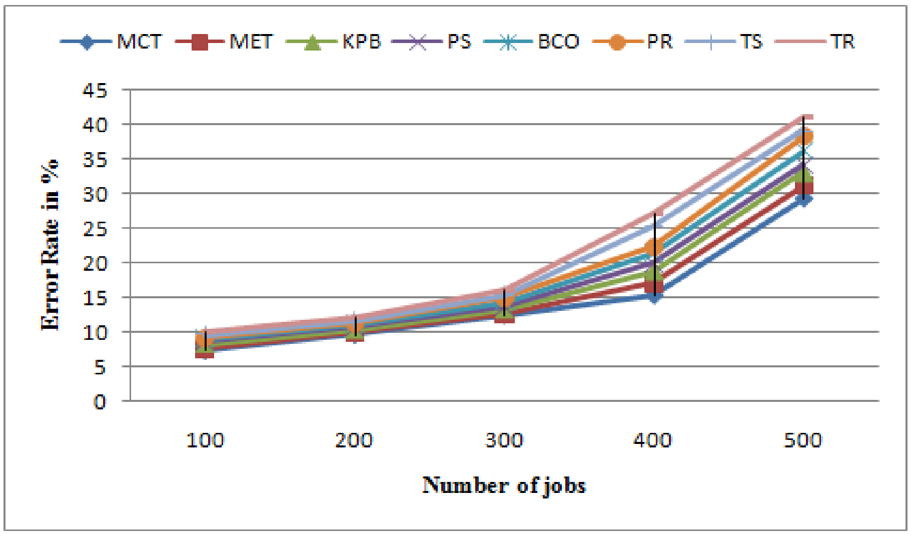


Figure 14: Error rate versus number of jobs in presented and existing algorithms

9 Conclusions

Cloud computing is a collection of distributed and parallel computing to create shared resources, including hardware, software, and information. One of the most important growing areas in the field of cloud computing is hand over computing as utility. One of the important research fields that the researchers have been trying to resolve is the scheduling ad load balancing methods. Usually available methods for scheduling and load balancing in cloud environments are often based on concepts such as Makespan, execution time, resource utilization. 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), efficiency, imbalance degree and etc for solving cloud computing load balancing challenge is presented. The main idea in designing the proposed algorithms is the fair allocation of tasks to virtual machines in order to prevent the excessive increase in the price of one machine and unemployment of other machine. 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 and TR were compared with the results of BCO, MET, MCT and KPB algorithms. Results show, the proposed algorithms outperform the above existing algorithms in terms of Total Cost (TC), Final Cost (FC), imbalance degree and efficiency.

References

Agache, M. and Oommen, B. J. 2002. Generalized pursuit learning schemes: New families of continuous and discretized learning automata, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **32**(6): 738–749.

Al Nuaimi, K., Mohamed, N., Al Nuaimi, M. and Al-Jaroodi, J. 2012. A survey of load balancing in cloud computing: Challenges and algorithms, *2012 second symposium on network cloud computing and applications*, IEEE, pp. 137–142.

Antonopoulos, N. and Gillam, L. 2010. *Cloud computing*, Springer.

Arabnejad, H. and Barbosa, J. G. 2017. Multi-qos constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems, *Future Generation Computer Systems* **68**: 211–221.

Asnaashari, M. and Meybodi, M. R. 2007. Irregular cellular learning automata and its application to clustering in sensor networks, *Proceedings of 15th Conference on Electrical Engineering (15th ICEE), Volume on Communication, Telecommunication Research Center, Tehran, Iran*.

Buyya, R., Abramson, D. and Giddy, J. 2001. A case for economy grid architecture for service-oriented grid computing., *IPDPS*, Vol. 1, Citeseer, pp. 20083–1.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, s. A. and Buyya, R. 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Software: Practice and experience* **41**(1): 23–50.

Chawla, Y. and Bhonsle, M. 2012. A study on scheduling methods in cloud computing, *International Journal of Emerging Trends and Technology in Computer Science (IJETTCS)* **1**(3): 12–17.

Chen, J. and Lu, B. 2008. Load balancing oriented economic grid resource scheduling, *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, Vol. 2, IEEE, pp. 813–817.

Dasgupta, K., Mandal, B., Dutta, P., Mandal, J. K. and Dam, S. 2013. A genetic algorithm (ga) based load balancing strategy for cloud computing, *Procedia Technology* **10**: 340–347.

Eraghi, A. E., Torkestani, J. A. and Meybodi, M. R. 2011. Cellular learning automata-based graph coloring problem, *Machine Learning and Computing: Selected, Peer Reviewed Papers from the 2009 International Conference on Machine Learning and Computing (ICMLC 2009), Perth, Australia*.

Fang, Y., Wang, F. and Ge, J. 2010. A task scheduling algorithm based on load balancing in cloud computing, *International Conference on Web Information Systems and Mining*, Springer, pp. 271–277.

Ghosh, J. B. and Gupta, J. N. 1997. Batch scheduling to minimize maximum lateness, *Operations Research Letters* **21**(2): 77–80.

Grossman, R. L. 2009. The case for cloud computing, *IT professional* **11**(2): 23–27.

Guo, L., Zhao, S., Shen, S. and Jiang, C. 2012. Task scheduling optimization in cloud computing based on heuristic algorithm, *Journal of networks* **7**(3): 547.

Hemamalini, M. 2012. Review on grid task scheduling in distributed heterogeneous environment, *International Journal of Computer Applications* **40**(2): 24–30.

Henzinger, T. A., Singh, A., Singh, V., Wies, T. and Zufferey, D. 2011. Static scheduling in clouds.

Hosseinzadeh, M. et al. 2019. Labts: a learning automata-based task scheduling algorithm in cloud computing, *International Journal of Information & Communication Technology Research* **11**(2): 49–61.

Jain, P. and Gupta, D. 2009. An algorithm for dynamic load balancing in distributed systems with multiple supporting nodes by exploiting the interrupt service, *International Journal of Recent Trends in Engineering* **1**(1): 232.

Javanmardi, S., Shojafar, M., Amendola, D., Cordeschi, N., Liu, H. and Abraham, A. 2014. Hybrid job scheduling algorithm for cloud computing environment, *Proceedings of the fifth international conference on innovations in bio-inspired computing and applications IBICA 2014*, Springer, pp. 43–52.

Kaur, R. and Kinger, S. 2014. Enhanced genetic algorithm based task scheduling in cloud computing, *International Journal of Computer Applications* **101**(14): 1–6.

Kaur, R. and Luthra, P. 2012. Load balancing in cloud computing, *Proceedings of international conference on recent trends in information, telecommunication and computing, ITC*, Citeseer.

Kenny, Q. Y. et al. 2003. Indicator function and its application in two-level factorial designs, *Annals of Statistics* **31**(3): 984–994.

Khiyaita, A., El Bakkali, H., Zbakh, M. and El Kettani, D. 2012. Load balancing cloud computing: state of art, *2012 National Days of Network Security and Systems*, IEEE, pp. 106–109.

Krishna, P. V., Misra, S., Nagaraju, D., Saritha, V. and Obaidat, M. S. 2016. Learning automata based decision making algorithm for task offloading in mobile cloud, *2016 International Conference on Computer, Information and Telecommunication Systems (CITS)*, IEEE, pp. 1–6.

Kumar, M. and Sharma, S. 2020. Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment, *International Journal of Computers and Applications* **42**(1): 108–117.

Kumar, V. V. and Dinesh, K. 2012. Job scheduling using fuzzy neural network algorithm in cloud environment, *Bonfring International Journal of Man Machine Interface* **2**(1): 01–06.

Lakshmivarahan, S. 1981. Time varying learning algorithms, *Learning Algorithms Theory and Applications*, Springer, pp. 109–135.

Lakshmivarahan, S. 2012. *Learning algorithms theory and applications: theory and applications*, Springer Science & Business Media.

Lee, Z., Wang, Y. and Zhou, W. 2011. A dynamic priority scheduling algorithm on service request scheduling in cloud computing, *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*, Vol. 9, IEEE, pp. 4665–4669.

Leitão, P., Barbosa, J., Funchal, G. and Melo, V. 2020. Self-organized cyber-physical conveyor system using multi-agent systems, *be published in the International Journal of Artificial Intelligence* .

Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X. and Gu, Z. 2012. Online optimization for scheduling preemptable tasks on iaas cloud systems, *Journal of Parallel and Distributed Computing* **72**(5): 666–677.

Lin, X. and Wu, C. Q. 2013. On scientific workflow scheduling in clouds under budget constraint, *2013 42nd International Conference on Parallel Processing*, IEEE, pp. 90–99.

Llwaah, F., Thomas, N. and Cala, J. 2015. Improving mct scheduling algorithm to reduce the makespan and cost of workflow execution in the cloud, *UK Performance Engineering Workshop*, Newcastle University.

Mansouri, N. and Javidi, M. M. 2019. Cost-based job scheduling strategy in cloud computing environments, *Distributed and Parallel Databases* pp. 1–36.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. and Ghalsasi, A. 2011. Cloud computing—the business perspective, *Decision support systems* **51**(1): 176–189.

Masdari, M., ValiKardan, S., Shahi, Z. and Azar, S. I. 2016. Towards workflow scheduling in cloud computing: a comprehensive analysis, *Journal of Network and Computer Applications* **66**: 64–82.

Narendra, K. S. and Thathachar, M. A. 1974. Learning automata-a survey, *IEEE Transactions on systems, man, and cybernetics* (4): 323–334.

Panwar, R. and Mallick, B. 2015. Load balancing in cloud computing using dynamic load management algorithm, *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, IEEE, pp. 773–778.

Precup, R.-E., Teban, T.-A., Albu, A., Borlea, A.-B., Zamfirache, I. A. and Petriu, E. M. 2020. Evolving fuzzy models for prosthetic hand myoelectric-based control, *IEEE Transactions on Instrumentation and Measurement* **69**(7): 4625–4636.

Qavami, H. R., Jamali, S., Akbari, M. K. and Javadi, B. 2017. A learning automata based dynamic resource provisioning in cloud computing environments, *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, IEEE, pp. 502–509.

Rodriguez, M. A. and Buyya, R. 2017. Budget-driven scheduling of scientific workflows in iaas clouds with fine-grained billing periods, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* **12**(2): 1–22.

Rodrıguez, S. and Corchado, J. M. 2020. Smart belt design by naıve bayes classifier for standard industrial protection equipment integration, *Int. J. Artif. Intell* **18**: 186–201.

Sahoo, S., Sahoo, B. and Turuk, A. K. 2019. A learning automata-based scheduling for deadline sensitive task in the cloud, *IEEE Transactions on Services Computing* .

Sareen, P. and Singh, T. D. 2016. Simulation of cloud computing environment using cloudsim, *Simulation* **4**(12).

Sarhadi, A. and Meybodi, M. R. 2010. 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*, Vol. 1, IEEE, pp. 32–35.

Sarhadi, A. and Yousefi, A. 2011. Proffering a new method for grid computing resource discovery based on economic criteria using ant colony algorithm, *International Journal of Computer Applications* **975**: 8887.

Selvarani, S. and Sadhasivam, G. S. 2010. Improved cost-based algorithm for task scheduling in cloud computing, *2010 IEEE International Conference on Computational Intelligence and Computing Research*, IEEE, pp. 1–5.

Shah, N. and Farik, M. 2015. Static load balancing algorithms in cloud computing: challenges and solutions, *International Journal Of Scientific and Technology Research* **4**(10): 365–367.

Shojafar, M., Javanmardi, S., Abolfazli, S. and Cordeschi, N. 2015. Fuge: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method, *Cluster Computing* **18**(2): 829–844.

Singh, S. and Chana, I. 2016. A survey on resource scheduling in cloud computing: Issues and challenges, *Journal of grid computing* **14**(2): 217–264.

Verma, A. and Kaushal, S. 2012. Deadline and budget distribution based cost-time optimization workflow scheduling algorithm for cloud, *IJCA Proceedings on international conference on recent advances and future trends in information technology (iRAFIT 2012)*, Vol. 4, Citeseer, pp. 1–4.

Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J. and Fu, C. 2010. Cloud computing: a perspective study, *New generation computing* **28**(2): 137–146.

Xhafa, F., Carretero, J., Barolli, L. and Durresi, A. 2007. Immediate mode scheduling in grid systems, *International Journal of Web and Grid Services* **3**(2): 219–236.

Xie, Y., Zhu, Y., Wang, Y., Cheng, Y., Xu, R., Sani, A. S., Yuan, D. and Yang, Y. 2019. A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment, *Future Generation Computer Systems* **97**: 361–378.

Yuhana, U. L., Fanani, N. Z., Yuniarno, E. M., Rochimah, S., Koczy, L. T. and Purnomo, M. H. 2020. Combining fuzzy signature and rough sets approach for predicting the minimum passing level of competency achievement, *Int J Artif Intell* **18**: 237–249.

Zamfirache, I. A., Precup, R.-E., Roman, R.-C. and Petriu, E. M. 2021. Policy iteration reinforcement learning-based control using a grey wolf optimizer algorithm, *Information Sciences* .

Zhang, P. and Zhou, M. 2017. Dynamic cloud task scheduling based on a two-stage strategy, *IEEE Transactions on Automation Science and Engineering* **15**(2): 772–783.

Zhao, Y., Calheiros, R. N., Gange, G., Ramamohanarao, K. and Buyya, R. 2015. Sla-based resource scheduling for big data analytics as a service in cloud computing environments, *2015 44th International Conference on Parallel Processing*, IEEE, pp. 510–519.

Zuo, L., Shu, L., Dong, S., Zhu, C. and Hara, T. 2015. A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing, *IEEE Access* **3**: 2687–2699.