LADA: Learning Automata-Based Approach for Task Assignment on Heterogeneous Cloud Computing platform

Soulmaz Gheisari
So_gheisari@pardisiau.ac.ir
Department of Computer Engineering,
Pardis Branch, Islamic Azad University, Pardis, Iran

Hamid Shokr Zadeh
Shokrzadeh@gmail.com
Department of Computer Engineering,
Pardis Branch, Islamic Azad University, Pardis, Iran

Abstract

A cloud computing environment is a distributed system in which idle resources are available in a wide area network like the Internet. Since these resources have diverse specifications, computational clouds are highly heterogeneous systems. Dispatching cloud applications onto processing nodes is known as task scheduling. Finding proper machines or even virtual machines (VMs) that can execute jobs in order to gain high utilization in such a heterogeneous environment is perceived as a multi-objective optimization problem. In this paper, a dynamic learning automata-based task scheduling algorithm is proposed. In the algorithm, which is named LADA, each application is modeled as a *Directed Acyclic Graph* (DAG) that contains tasks as nodes and data dependencies between them as edges. At first, tasks are grouped based on their data dependencies in order to collect independent tasks to one group. Then, a variable-structure learning automaton is associated with each task in the groups to discover an appropriate pair task-machine combination. Main goals of LADA are minimizing makespan and energy consumption through efficient task placement that leads to load balance and maximizes resource utilization. In addition, an improvement is added, which upgrades the results by using a different grouping policy before task assignment. Eventually, computer simulation outcomes reveal the superior performance of the proposed algorithms on high heterogeneous environment compared with the state-of-the-art algorithms. Total execution time and energy consumption decrease up to 50% and 37%, respectively.

Keywords: Cloud Computing; Directed Acyclic Graph (DAG); Learning Automata (LA); Task Scheduling

1. Introduction

A cloud computing platform, which usually works on a "Pay for Use" policy, provides global on-demand availability to a shared pool of resources such as storage space and computing servers with diverse specifications and high heterogeneity. Clients submit their requests to the cloud environments to access these resources steadily with full confidence and no stress. In other words, the main purpose of a cloud infrastructure is to provide an easy-to-use and reliable workspace for dynamic applications of different users. However, cloud computing is an example of *heterogeneous computing* (HC) environments, in which dynamically fluctuating delays, changing demands of quality of services, and unpredictable behavior are inevitable due to high heterogeneity in resources and also different types of users. Hence, task scheduling in the cloud environment, which means assigning tasks to these high heterogeneous servers to be executed properly, is a major concern of multitude of efforts in recent years. [1-4]

Generally in HC environments and specifically in cloud environments, there are various definitions for task assignment based on the applications and the platform characterizations. In such a system, users aim to achieve efficient scheduling parameters after allocating all their tasks to the available resources.

To begin with, a Directed Acyclic Graph (DAG) is used to show the data dependencies between various tasks of an application. It represents that some of the tasks should be executed before the others, and how much data should be transferred in order to execute further tasks. The tasks, are allocated to machines or virtual machines (VMs). Then, each VM executes a single task at a moment, in a non-preemptive manner. The number of machines or VMs and the application characterization is known beforehand. Assigning tasks to the machines or VMs and scheduling them to be executed is known as a nondeterministic polynomial time (NP)-hard problem [5] and it is responsible for giving tasks to machines or VMs in a way that increases the resource utilization, reduces response time and energy consumption, and keeps the whole system balanced [5-11].

In this article a new learning automata-based algorithm for task assignment in the cloud environments is proposed; it is called LADA. In LADA, optimization aim is achieved by optimizing the weighted sum of various metrics. These metrics are total execution time (makespan), load of each machine and energy consumption.

While previous approaches face many shortcomings such as **High latency**, **Inefficient machine selection** and **poor resource management**, contributions of our paper can be listed as the following:

- **Low task assignment latency**: Both modeling the applications in form of DAGs instead of considering separated tasks and using learning automata, whose time complexity is O(1), to assign tasks increase the speed of the proposed model LADA.
- **Cost-effective resource provisioning:** Another advantage of LADA is decreasing makespan as well as energy consumption during task scheduling. These achieve by reinforcement learning which can discover the most suitable pairs task-machine during the time.

- **Efficient resource management:** Efficient resource utilization is a crucial consideration in preserving the energy of resources. the proposed solution will prevent resource overloaded and underloaded.

While LADA can perform many real world applications, i.e. executing real user requests in less time and within sufficient energy consumption, the main benefit of it can also be mentioned in two following significant points:

- **Prevent global warming:** LADA aims to reduce energy consumption and use resources more efficiently. This means that the servers will need less electricity to run. This is important because high electricity useage contributes to global warming.
- **Boost cloud provider's throughput:** The proposed model LADA considers load balancing and less energy consumption simultaneously. This means that it makes resource management easier for providers.

The rest of the paper is organized as follows. Section 2 discusses the related works. Since Learning automata is used in the proposed model, it is described in section 3. Section 4 explains the model of the cloud platform, as well as the definitions used in later sections. The proposed learning automata-based algorithm is illustrated in section 5. In section 6 simulation results are discussed, and finally in section 7 we summarized the whole work and examine the conclusion.

2. Related Works

In recent years of the survey, it has been noticed that many heuristics or meta-heuristics task scheduling algorithms are introduced to optimize the performance and efficiency of cloud computing systems. The main purpose of task scheduling algorithms is assigning tasks to machines or VMs in a way that maximize the resource utilization, minimize the total execution time and energy consumption, while the whole system remains balanced. In the following some highlighted algorithms are explained.

In [1], the authors provided a new variation of heuristic-based algorithm called Heterogeneous Earliest Finish Time (HEFT) to carry out task scheduling and resources allocation in cloud environments. The HEFT algorithm schedules the DAG tasks into heterogeneous processors in two stages: the rank generation (based on the average communication and computation cost) and the processor selection stages (based on decreasing the schedule length of the task).

The authors in [12] proposed a new task priority strategy and applyed task duplication methods, for solving task scheduling problem for dependent tasks in heterogeneous cloud computing systems. They use optimistic cost table downward (OCTd) and optimistic cost table upward (OCTu) procedures to prioritize tasks in an efficient ordered list; then, Heterogeneous Earliest Finish Time (HEFT)-duplication method is used to perform task duplication method which pointedly decreases makespan.

In [13], a Grouping Genetic Algorithm (GGA) with different mutation operators (called 2-Items Reinsertion) designed to solve the Parallel-Machine scheduling problem with unrelated machines and makespan minimization. Experimental results indicate that makespan is considerably reduce by replacing the original mutation operator with the new one.

The authors in [14] provided a conceptual framework to achieve an effective job scheduling technique. A new priority-based scheduling method to fair job scheduling is presented in this paper.

In [15], the authors proposed a third-generation multi-objective optimization method called Nondominated Sorting Genetic Algorithm (NSGA-III) to schedule a set of user tasks on a set of available virtual machines (VMs) in cloud environments based on a new multi-objective adaptation function to improve the energy consumption and the cost.

An enhanced sunflower optimization (ESFO) meta-heuristic algorithm has been proposed in [16] to improve the performance of the existing task scheduling algorithms. The ESFO improves the pollination operator of the conventional SFO algorithm to accurately explore the solution space. The Authors claim that ESFO can find near-optimal scheduling in a polynomial time complexity.

The authors in [17] combine Genetic Algorithm and Electro Search algorithm (HESGA) in order to consider the task scheduling parameters such as load balancing, makespan, resources utilization, and the cost of multi-cloud environments. Proposed method uses genetic algorithm advantages to provide the best local optimal solutions and Electro search algorithm to provide the best global optima solutions. HESGA is implemented in Cloudsim and is compared with other algorithms such as HPSOGA, ES, GA, and ACO in terms of makespan, cost, and response time.

Another meta heuristic task scheduling algorithm has been proposed in [18] based on using particle swarm optimization (PSO) in heterogeneous multi-cloud environments. In the article, using the working mechanism of particle swarm optimization (PSO) algorithm, a set of solutions or schedules is created. A solution with the most efficient QoS parameters i.e., makespan, cloud utilization, and energy consumption is chosen for allocating the task into the heterogeneous multi-cloud environment.

In [19], another heuristic model is presented based on mean grey wolf optimization algorithm. The simulation results show that the proposed mean-GWO algorithm improves the performance of task scheduling compared with the existing PSO and standard GWO techniques.

The authors in [20] proposed a game theory-based approach for managing dynamic cloud services, such as resource allocation and task assignment. Their main

contribution is to guarantee the reliability of cloud services, based on their claim.

In [21], the authors suggested an algorithm for scheduling the tasks. It uses the standard deviation of the estimated task execution time on the resources available in the computing environment. This approach considers the heterogeneity of the task. The authors presented an improved version of the algorithm in [22]. It is used for compilation of a scheduling list, where the priorities of the tasks are computed.

Two novel algorithms for heterogeneous processors have been proposed in [23]. The main goal of the algorithm is achieving fast scheduling time and high performance. The experimental results showed that they outperformed existing algorithms in terms of the quality and the cost of schedules.

In [24], a state-of-the-art duplication-based algorithm was proposed to reduce the delay and makespan of task assignment. The proposed algorithm schedules tasks with the least redundant duplications.

Moreover, in [25] authors proposed a load balancing technique based on using modified PSO task scheduling (LBMPSO) to schedule tasks over the available cloud resources. They claimed that LBMPSO minimizes the makespan and maximizes resource utilization by having proper information among the tasks and resources.

A multi-objective genetic algorithm is proposed in [26]. The article proposed an initialization scheduling sequence scheme, in which each task's data size is considered when initializing its virtual machine instance to achieve a proper trade-off between the makespan and the energy consumption. In the early evolution stage, traditional crossover and mutate operators are performed to keep the population's exploration and the Longest Common Subsequence (LCS) of multiple elite individuals is saved during the later evolution stage. The integration of the LCS in GA can satisfy different requirements in different evolution stages, and then to attain a balance between the exploration and the exploitation.

In [27], a novel methodology for dynamic load balancing among the virtual machines was proposed. The algorithm uses hybridization of modified Particle swarm optimization (MPSO) and improved Q-learning algorithm. The hybridization process is used to adjust the velocity of the MPSO through the gbest and pbest based on the best action generated through the improved Q-learning. The aim of hybridization is to enhance the performance of the machine by balancing the load among the VMs, maximize the throughput of VMs and maintain the balance between priorities of tasks by optimizing the waiting time of tasks.

Moreover, some learning automata-based task scheduling algorithms were proposed in [28-32]. The learning process in [28] begins with an initial population of randomly generated learning automata. Each automaton by itself represents a stochastic scheduling, and the scheduling is optimized within a learning process. The authors claimed that compared with genetic approaches to DAG scheduling, it achieves better results because in their approach learning automata is applied to find the most suitable position for the genes in addition to looking for the best chromosomes, whilst genetic approaches look for the best chromosomes within genetic populations.

In [29], three learning automata-based algorithms for mapping of a class of independent tasks over highly heterogeneous grids were presented. The main difference between the algorithms is related to the implementation of their reinforcement signals.

The authors in [30] proposed a framework based on a learning automata model for task assignment in heterogeneous computing systems. The proposed model can be used for dynamic task assignment and scheduling and can adapt itself to changes in the hardware or network environment.

A stochastic model for decentralized control of task scheduling in distributed processing systems is presented in [31]. The main feature of the model is applying learning automata on each host which causes low execution overhead as well as potential reliability.

In [32], authors used learning algorithm to cope with the weakness of GA based method. In fact they used the Learning automata as local search in the memetic algorithm.

3. Learning Automata

A learning automaton (LA) is an abstract machine learning model which belongs to reinforcement learning category. It randomly selects one action out of its finite action-set and performs it on a random environment. The environment then evaluates the selected action and responses to the automaton with a reinforcement signal. Based on the selected action, and the received signal, the automaton updates its internal state and selects its next action. Interconnection between the learning automaton and the environment is shown in Fig. 1 [33-34].

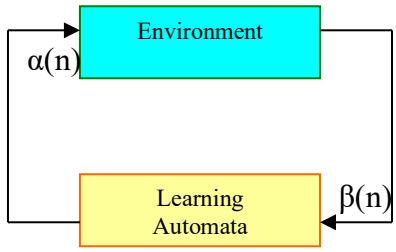


Fig 1: Learning automata connection with environment [33].

The environment is defined by a triple, $E = \{\alpha, \beta, c\}$ where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents a finite input set, $\beta = \{\beta_1, \beta_2, \ldots, \beta_n\}$ depicts the output set and $c = \{c_1, c_2, \ldots, c_r\}$ shows a set of penalty probabilities where each element c_i of c corresponds to one input action α_i. The environments in which β can take only binary values, 0 or 1, are referred to as P-models. If finite output set with more than two elements that take values in the interval [0, 1] is possible, the environment is

known as Q-model. Finally, when the output of is a continuous random variable in the interval [0, 1], it is called S-model environment. Also, a learning automaton is classified into fixed-structure [33], and variable-structure; here, the variable-structure LA (VSLA) is considerd.

A variable-structure automaton is defined by the quadruple $\{\alpha, \beta, p, T\}$, in which $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents the action set, $\beta = \{\beta_1, \beta_2, \ldots, \beta_n\}$ shows the input set, $p = \{p_1, p_2, \ldots, p_r\}$ is the action probability set, and finally $p(n+1) = T[\alpha(n), \beta(n), p(n)]$ illustrates the learning algorithm. This automaton operates as follows: based on the action probability set p, an action α_i, is randomly selected to perform on the environment. After receiving the environment's feedback in form of reinforcement signal, automaton updates its action probability set based on equation (1) for favorable responses, and equation (2) for unfavorable ones.

$$p_i(n+1) = p_i(n) + a\big(1 - p_i(n)\big) \qquad (1)$$
$$p_j(n+1) = p_j(n) + a\Big(p_j(n)\Big)\ \forall j\ j \neq i$$

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

In these two equations, a and b are reward and penalty parameters, respectively. If $a=b$, learning algorithm is called L_{R_P}[1], if $a<<b$, it is called $L_{R\varepsilon P}$[2], and if $b=0$, it is called $L_{R\text{-}I}$[3]. For more information about learning automata refer to [33].

A single automaton is generally sufficient for learning the optimal value of one parameter. However, for multi- dimensional optimization problems, we need a system consisting of as many automata as there are parameters [34]. Let $A_1, \ldots, A_n$ be the automata involved in an N-player game. Each play of game consists of choosing an action by each learning automaton and then getting the payoffs or reinforcement signals from the environment for this choice of actions by the group of learning automata. Let $p_1(k), \ldots, p_n(k)$ be the action probability distributions of N automata; at each instant k, each automaton, Ai, chooses an action $a_n(k)$ independently and at random according to $p_i(k)$, $1 \leq i \leq N$. This set of N actions is input to the environment, and the environment responds with N random payoffs, which are supplied as reinforcement signals to the corresponding automata [34]. In this paper, a game of τ learning automata is applied to map τ tasks to μ machines or VMs on them.

4. Scheduling System Model

4.1. Definitions

In this section a scheduling system model in cloud computing environments, which consist of heterogenous servers is described. Fig. 2 shows the schematic representation of the environment. The model includes a cloud environment, executable applications, and a scheduling system in the center. The scheduling system is composed of learning automata, a directed acyclic graph (DAG) which shows various tasks of an application and data dependencies between them, and a servers model that illustrates machines and as well as communication cost between them (machine to machine data transfer cost). The tasks of the DAG are assigned to the servers through a game of learning automata, then cost metrics such as total execution tim, the average load of servers and energy consumption are applied to calculate the reinforcement signals, given back to the learning automata. Each automaton updates its action probability set based on the received signal. This process repeats until the scheduling system assigns all tasks to the servers (VMs) effectively. Obviously, this model can successfully adapt with changes in applications, as well as servers include VMs or the network.

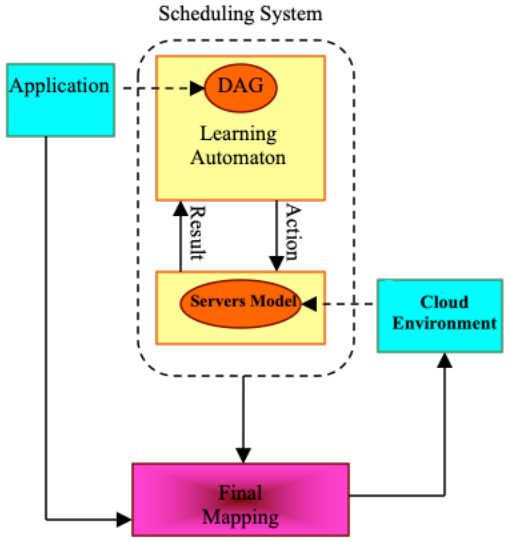


Fig 2: model of scheduling system in the cloud computing platform.

In the DAG, $T = \{t_1, t_2 \ldots t_\tau\}$ represents the set of tasks; the data dependency between each pair of tasks is represented by a $\tau \times \tau$ matrix DD, where τ is the number of tasks; $DD(i, j)$ shows the amount of data that should be transferred from task i to task j. In other words, it is the weight of the edge which connects nodes (tasks) i and j.

The expected execution time for each task on each machine's VMs is known prior and is contained within a $\tau \times \mu$ matrix ET (execution time), where τ is the number of tasks and μ is the number of machines. $ET(i, j)$ depicts the execution time of task i on the VMs of machine j (All VMs of a server are assumed to have the same configuration and can be assigned tasks equally). The weight of each node in the DAG w_i is computed by equation (3).

$$w_i = \frac{\sum_{j=1}^{\mu} ET(i,j)}{\mu} \qquad (3)$$

The cost of communication between any two machines as well as between VMs on different

[1] Linear Reward-Penalty
[2] Linear Reward epsilon Penalty
[3] Linear Reward Inaction

machines is also known as a priori and it is shown as a μ×μ matrix named *CC*.

$\psi(i) = j$ is defined as the assignment of task i to machine j; therefore, transfer cost between two tasks i and j shown as $\theta(i,j)$ is computed by $\theta(i,j) = DD(i,j) \times CC(\psi(i), \psi(j))$. If there is no certain assignment, it is calculated as equation (4):

$$\theta(i,j) = DD(i,j) \times \sum_{k=1,l=1}^{\mu} p_k \times p_l \times CC(k,l) \quad (4)$$

Where, p_k and p_l are the probability of choosing machine k and machine l, respectively.

Transfer cost is zero provided that two tasks which are connected by an edge are executed on the same machine, even the VMs are different.

4.2. Cost metrics

In order to evaluate task assignment performance some metrics must be measured; here, total execution time (makespan), the load of each machine, and energy consumption are considered. The purpose of task scheduling is to minimize makespan, balance the load of machines, and minimize total energy consumption.

The Load of a machine is defined as the time taken to execute all its assigned tasks; in other words, it is the time when the last assigned task finishes. The maximum load value over all μ machines is referred to as makespan; so, to minimize the makespan, the load on the maximum loaded machine should be minimized, which means that the load of all machines should be balanced.

If the total load of a particular machine like j at iteration n be represented as $L_j(n)$, we have:

$$L_j(n) = FT(t_i, j) \quad (5)$$

Where t_i is the last task executed on j; $FT(t_i, j)$ is finish time, when task t_i on machine j terminated; and it is defined by equation (6).

$$FT(t_i, j) = ET(t_i, j) + \max FT((t_{i-1}, j), ST(t_i, j)) \quad (6)$$

$ST(t_i, j)$ is the time when t_i can start executing on j. Accordingly, equation (7) is used to calculate the makespan T_μ, at iteration n.

$$T_\mu(n) = \max_{1 \le j \le \mu} (L_j(n)) \quad (7)$$

In order to balance the servers' load, the ideal average of load δ_{ideal} is computed; consider η as the average of transfer cost for all tasks, it is calculated as the following:

$$\eta = \frac{\sum_{i=1}^{\tau} \sum_{j \in s(i)} \theta(i,j)}{\tau} \quad (8)$$

Then,

$$\delta_{ideal} = \frac{\sum_{i=1}^{\tau} (\sum_{j=1}^{\mu} ET(i,j) + t_j^{idle} + \eta)}{\mu^2} \quad (9)$$

Where t_j^{idle} is idle time consuming between two adjacent VMs in machine j. Equation (10) represents the average load of all machines at iteration n. The closer $AVG_{load}(n)$ is to δ_{ideal}, the tasks scheduling of iteration n is better.

$$AVG_{load}(n) = \frac{\sum_{j=1}^{\mu} L_j(n)}{\mu} \quad (10)$$

Eventually, an energy model is needed to estimate the total amount of energy consumption which is used by the scheduling method. In order define the energy model, power and machine utilization are formulated [35], because the average of cloud utilization and energy consumption are linearly proportional. Energy consumption for machine j is mathematically defined by equation (11).

$$E_j = \int_t^{\Delta t} \left[(P_{max} - P_{min}) \times U_j + P_{min} \right] dt \quad (11)$$

Where, U_j is the utilization of machine j, which is computed as: $U_j = \frac{\sum_{i=1}^{n} ET(i,j)}{\max_{1 \le j \le \mu} (L_j(n))}$; P_{max} is the power used at its fully loaded, and P_{min} is minimum power consumption in its idle state. Notice that workload affects power supply value over time. Average energy consumption in iteration n is calculated using equation (12).

$$E(n) = \frac{\left(\sum_{j=1}^{\mu} E_j \right)}{\mu} \quad (12)$$

5. LADA: Learning Automata Approach for Task Assignment in Cloud Environment

5.1. The Proposed Algorithm

In this section, a new learning automata-based task scheduling algorithm for cloud environments is proposed.

The scheduling process consists of three phases: ranking the tasks, group creation based on tasks dependencies, and scheduling independent tasks within each group using a game of learning automaton.

First phase, ranking: in this stage, for each node like i, the rank value r_i is recursively defined using equation (13):

$$r_i = \frac{\sum_{j=1}^{\mu} ET(i,j)}{\mu} + \max_{\forall l \in S_i} ((DD(i,l) \times \sum_{k=1,m=1}^{\mu} p_{k,m} \times CC(k,m)) + r_l \quad (13)$$

Where S_i is the set of immediate successors of node i in related DAG.

Second phase, group creation: in this step, the nodes are sorted in descending order related to their rank value. The first node, which is the root of the DAG, is added to group 0. Then a new group (group1) is created and successive nodes are placed in it as long as they do not depend on any other nodes in the same group. If a dependency is found, a new group with higher number is created and the last node is moved to it. The final outcome is a set of ordered groups, each containing a number of independent tasks, and having a predetermined priority based on the original ranking (a lower group number indicates a higher priority).

Third phase, scheduling independent tasks in each group using a game of learning automata: After the previous two phases, we have some groups of independent tasks with different priorities. A simple way to schedule these groups is to use an algorithm like MLQ (Multi-Level Queue). In MLQ, a task in a lower-priority queue cannot be scheduled until all the tasks in a higher-priority queue are scheduled. This means that the tasks of group i will not be scheduled until all the tasks of group i-1 are scheduled.

On the other hand, the tasks in each group are scheduled using a game of learning automata. Each task t_i in each group has a variable structure learning

automaton assigned to it. The action set of each automaton corresponds to a server, and since any task can be assigned to any server, the action sets of all learning automata are identical. Therefore, for any automaton, α_i is $\{m_1, m_2, \ldots, m_\mu\}$ and input value, β_i is a continuous value in [0,1]; β_i closer to 0 indicates that the action taken by the automaton is favorable to the system, and closer to 1 indicates an unfavorable response. Fig. 3 shows the model.

Reinforce scheme which is used to update the action probabilities of learning automata is LRI.

In each iteration, the learning automaton assigned to the root of the DAG (the task in the first group) selects a server first. Then, each learning automaton in the second group of tasks selects a server, and this process continues until all learning automata in all groups select servers[4]. Next, the weighted sum of makespan, load of each machine, and total energy consumption are calculated using the equations introduced in section 4.2. Then, the results are given to the learning automata as reinforcement signals to update their action probabilities. This is repeated until a termination condition is met.

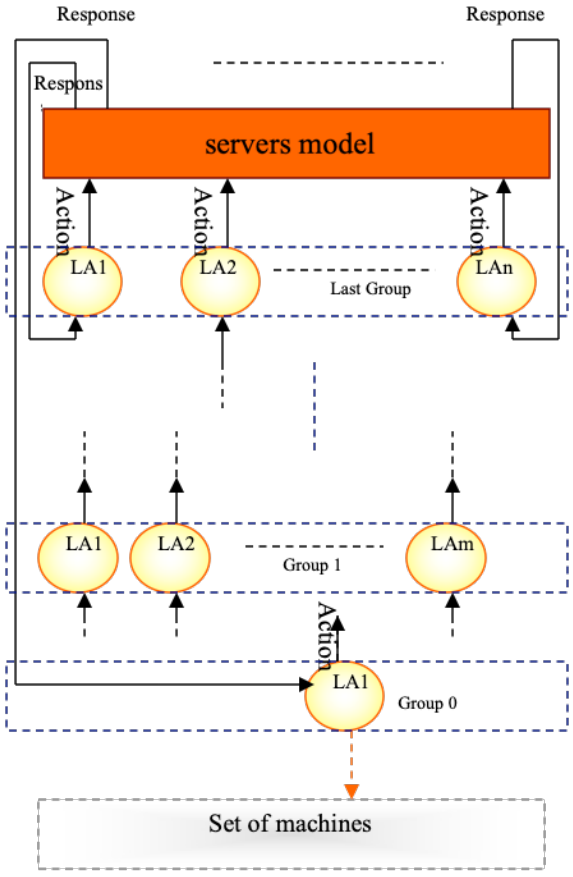


Fig 3: The learning automata model

To evaluate the quality of the selected actions (servers) via β_i, each automaton uses a generalized formulation, given by equation (14). It considers the difference between the makespan, load and energy level of the chosen server in the current iteration and their values of the server picked in the previous iteration.

$$\beta_i(n) = f_T\left(T_\mu^n, T_\mu^{n-1}\right)\lambda_T + f_L\left(L_{\psi(i)}^n, L_{\psi(i)}^{n-1}\right)\lambda_L + f_E\left(E_i^n, E_i^{n-1}\right)\lambda_E \quad (14)$$

λ_T , λ_L and λ_E are the weights associated with makespan, load and energy level of the server, respectively, and $\lambda_T + \lambda_L + \lambda_E = 1$. The greater the λ_T, the more the impact of variation in makespan has on evaluation of environment response. This statement is also true for both λ_L and λ_E . Functions f_T, f_L and f_E are given in equations (15-17). As mentioned before, we use S-model for learning automata.

$$f_T\left(T_\mu^n, T_\mu^{n-1}\right) = \begin{cases} 1-\phi & T_\mu^n \le T_\mu^{n-1} \\ 1+\phi & T_\mu^n > T_\mu^{n-1} \end{cases} \quad (15)$$

$$where\ \phi = \left|T_\mu^{n-1} - T_\mu^n\right|$$

$$f_L\left(L_{\psi(i)}^n, L_{\psi(i)}^{n-1}\right) = \frac{L_{\psi(i)}^n - \delta_{ideal}}{L_{\psi(i)}^{n-1} - \delta_{ideal}} \quad (16)$$

[5]

$$f_E(E_i^n, E_i^{n-1}) = \begin{cases} 1-\Gamma & E_i^n \le E_i^{n-1} \\ 1+\Gamma & E_i^n > E_i^{n-1} \end{cases} \quad (17)$$

$$where\ \Gamma = |E_i^{n-1} - E_i^n|$$

Algorithm1. LADA

```
//first phase:
For i=0 to τ-1
  Compute Wi;  //the weights of the nodes
For i= τ-1 to 0
  Compute Ri;  //the ranks of the nodes
Sort R[i] in descending order;
//second phase:
j=0;
Gj= {};
For i=0 to τ-1  //the index of the nodes in Ri
  If node I has dependency with a node in Gj
    j++;
    Gj= {};
  Add node I to Gj;
//third phase:
n=0;  //the number of iteration
While a termination condition is not fulfill
  For k=0 to j
    While Gk is not empty
      Select a machine from αi randomly;  // αi is the set of machines
  Compute Tn;  //makespan in iteration n
  For l=0 to μ  //for all machines
    Compute Lln, Eln;  //the load and the energy of the machine in iteration n

    For k=0 to j
      While Gk is not empty
        Compute βi;  // by using equation (14)
        If βi is close to zero
          Increase the P for selected machine;  //P is the probability of the selected machine in αi
          Decrease the P for others;
        Else
          Decrease the P for selected machine;
          Increase the P for others;
  n++;
```

Eventually, if the probability of an action in each automaton exceeds 95%, makespan doesn't change for more than 100 iterations, or the number of repetitions reaches a maximum limit, learning process will stop; in

[4] Note that the game of learning automata is used to assign the tasks to the servers; then, the assigned tasks can be easily dispatched among VMs on each server.

[5] δ_{ideal} is presented in equation (9)

the first and the second conditions, the automaton converges. Alogorithm1 describes the proposed method.

Since the task that is in group$_0$ is the root of the DAG and has the highest priority, it can be scheduled individually and without using learning automata. At the beginning of the algorithm, a VM with minimum execution time for this task is chosen, and then, in all iterations its VM will not be changed. This will make the algorithm better and faster in performance.

5.2. An Improvement to the Proposed Method

In order to enhance the performance of the algorithm, the second and the third phases are changed as follow.

Second phase, group creation: unlike before, here tasks are grouped in such a way that the tasks with the highest dependencies sit in the same sets. To do this, DAG is scanned in level order, and for each node four values are saved.

1. w_i which is the weight of node i, and is calculated using equation (3).
2. C_i as a successor or an ordered list of successors with the most transfer cost;

$$C_i = \underset{\forall j \in s(i)}{\max}\{w_j + \theta(i,j)\}$$

where $S(i)$ is the set of immediate successors of node i and ∞ is considered for the nodes in the last level (nodes with no successors). $\theta(i,j)$ has been given in equation (4).

3. P_i as a parent or an ordered list of parents with the most transfer cost;

$$P_i = \underset{\forall i \in pr(j)}{\max}\{w_j + \theta(i,j)\}$$

where $pr(j)$ is the set of the parents of node i. and for the first node (the root), P_i is ∞.

4. F_i is a flag which shows that a task is a member of a group (F_i =1) or not (F_i =0).

From the first node to the last one, if a node such as i does not belong to a group, a new group is created and i is put into it. Then, other node such as j is checked and if C_i=j and P_j=i , j is added to the group that i belongs to. After that, the weights of all nodes such as j where P_j =i are updated using the following equations.

$$if\ C_i = j\ \&\&\ P_j = i\ then$$

$$w_j = w_j + w_i + \underset{\forall k \in pr(j)}{\max}\{f(w_k + \theta(k,j), w_i)\}$$

$$else\ w_j = w_j + \underset{\forall k \in pr(j)}{\max}\{w_k + \theta(k,j)\}$$

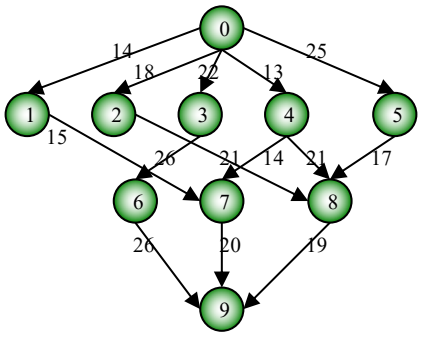
Where,

$$f(x,y) = \begin{cases} x - y & x > y \\ zero & x \le y \end{cases}$$

For all other nodes P_i is updated, and after reaching the last node, all nodes are put in their groups and F_i=1 for all of them. This creates some groups of the nodes with the highest dependencies.

An example

Fig. 4 represents a sample DAG. The numbers that have been written near the edges correspond to the amount of data that must be passed from a task to its immediate successor. The cost for executing each task on three different machines and the cost for transferring a data unit for any combination of machines are given in the tables. Fig. 5 shows grouping steps for the DAG.



A sample DAG

the execution time for each node on three machines							
Node	M0	M1	M2	Node	M0	M1	M2
0	17	19	21	5	4	8	9
1	22	27	23	6	17	16	15
2	15	15	9	7	49	49	46
3	30	27	18	8	25	22	16
4	17	14	20	9	23	27	19

the communication cost table for interconnected machines	
Machines	Time for transfer a data unit
M0-M1	0.9
M1-M2	1.0
M0-M2	1.4

Fig 4: An example of a DAG with computation and communication values.

Step 1; after level order scanning the graph				
Node	w_i	n_i	p_i	f_i
0	19	3	∞	0
1	24	7	0	0
2	13	8	0	0
3	25	6	0	0
4	17	7	0	0
5	7	8	0	0
7	48	9	1,4	0
8	21	9	4,2,5	0
6	16	9	3	0
9	23	∞	7	0

Group0: {0, 3}

Step 2: after putting 0, 3 in a group				
Node	w_i	n_i	p_i	f_i
0	19	3	-	1
1	53.22	7	-	0
2	45.14	8	-	0
3	44	6	-	1
4	45.49	7	-	0
5	44.25	8	-	0
7	48	9	1.4	0
8	21	9	2,4,5	0
6	16	9	3	0
9	23	∞	7	0

Group0: {0, 3, 6}; Group1: {1, 7}; Group3: {2, 8}; Group4: {4}; Group5: {5}

Step 3: after grouping nodes in level 1, 2, and 3				
Node	w_i	n_i	p_i	f_i
0	19	3	-	1
1	53.22	7	-	1
2	45.14	8	-	1
3	44	6	-	1
4	45.49	7	-	1
5	44.25	8	-	1
7	103.71	9	-	1
8	80.74	9	-	1
6	60	9	-	1
9	23	∞	7	0

Group0: {0, 3, 6}; Group1: {1, 7, 9}; Group3: {2, 8}; Group4: {4}; Group5: {5}

After grouping all nodes				
Node	w_i	n_i	p_i	f_i
0	19	3	-	1
1	53.22	7	-	1
2	45.14	8	-	1
3	44	6	-	1
4	45.49	7	-	1
5	44.25	8	-	1
7	103.71	9	-	1
8	80.74	9	-	1
6	60	9	-	1
9	23	∞	-	1

Fig.5: grouping steps for the sample DAG in Fig 4.

It is assumed that the cost to transfer data between two tasks t5hat are executed on the same machine (even different VMs) is zero.

Third phase, scheduling each group by using the a of learning automata: after grouping the tasks, each group should be scheduled to be executed. To do this, each group g_i is associated with a variable structure learning automaton. The actions of an automaton are associated with machines, and since the groups of tasks can be assigned to any of the μ machines, the action set of all learning automata are identical.

Therefore, for any automaton action set α_i is $\{m_1, m_2, \ldots, m_\mu\}$ and input value β_i is a continuous value in [0,1], which is computed using equation (14). β_i closer to 0 indicates that the action taken by the automaton is favorable, and closer to 1 indicates an unfavorable response. The learning automata model is simply depicted in Fig. 6. Reinforce scheme, which is used to update action probabilities of learning automata, is L_{RI} to avoid of sticking in local optimum.

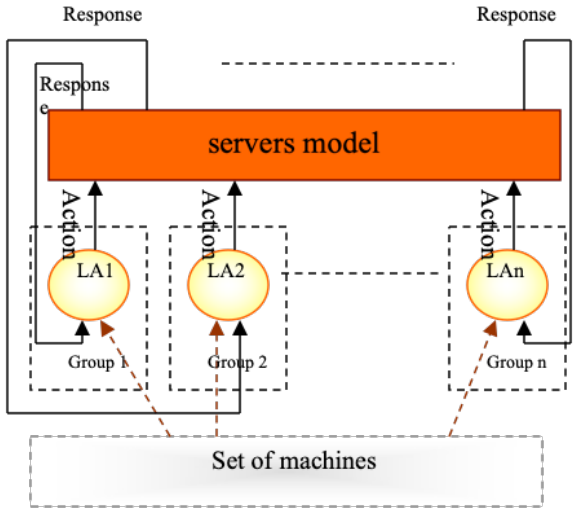


Fig 6: learning automata model

In each iteration, the game of learning automata select machines, then, makespan, load and energy consumption of each machine are computed and the results are passed to the learning automata in order to update their actions' probabilities. It is repeated, until one of the termination conditions is fulfilled.

To determine the goodness of an action taken by an automaton, equation (14) is applied. Learning algorithm stopping criteria is the same as previous. Algorithm2 explains above steps.

5.3 Time complexity analysis

The algorithms' complexity is estimated in three steps: the ranking, the group creation, and the tasks assignment. Let n be the number of iterations, τ be the number of tasks and μ be the number of machines. In order to calculate the ranking value, we have $O\big(n \times (\mu + \mu^2)\big)$, which can be rewritten as $O(n \times \mu^2)$. In the second phase, a sorting algorithm is needed which is $O(n \times \tau \log \tau)$, then, all tasks will be scanned to add to a group; so, total time complexity is $O(n(\tau \log \tau + \tau))$, simplify to $O(n \times \tau \log \tau)$. Finally in the scheduling stage, a game of learning automata is applied, in which the time complexity of each automaton is equal to $O(1)$; therefore, for τ learning automata in n iterations we have $O(n \times \tau)$. Consequently, LADA's time complexity is considered as $O(n \times (\mu^2 + \tau \log \tau))$.

Algorithm2. Improved-LADA

```
// second phase (grouping):
For i=0 to τ-1
  Compute Wi, Ni, Pi ;  /*the weights of the nodes, successor node with the most
                        transfer cost, parent node with the most transfer cost*/
  Fi=0;
k=0;
Gk= {};
For i=0 to τ-1
  If Fi=0 then
    Add node i to Gk;
    Fi=1
      For all i's successors like j
        If Ni=j && Pj=i
          Add node j to Gk;
          Fj=1;
          Else
                        K++;
                        Add node j to Gk;
      For all i's successors like j
        If Fj=0
          Update their weights;
// third phase (scheduling):
n=0;  // number of iterations
While a termination condition is not fulfill
  For l=0 to k
    Select a machine from αi randomly;  // αi is the set of machines
  Compute Tn;  //makespan in iteration n
  For l=0 to μ  //for all machines
    Compute Lln, Eln;  //the load and the energy of the machine in iteration n
  For l=0 to k
    Compute βi;  // by using equation (14)
    If βi is close to zero
      Increase the P for selected machine;  /*P is the probability of the selected machine in αi*/
      Decrease the P for others;
    Else
      Decrease the P for selected machine;
      Increase the P for others;
  n++;
```

5.4 Convergence analysis

However the game of learning automata is competitive and each automaton receives its own payoff and tries to improve it, it is expected that the algorithm reaches a *Nash equilibrium*. For good discussion on Nash equilibrium please refer to [32]. The m-tuple of actions $(\alpha_1, \ldots, \alpha_m)$ is called a Nash equilibrium of this game if for each j, for $1 \le j \le m$, $d^j(\alpha_1, \ldots, \alpha_{j-1}, x, \alpha_{j+1}, \ldots, \alpha_m) \le d^j(\alpha_1, \ldots, \alpha_m)\ \forall x \in S_i$.

Where d^j is the payoff function for automaton A_j $(d^j(x_1, \ldots, x_m) = E[\beta_j(t) | \alpha_j(t) = x_j, 1 \le j \le m])$, and S_i is the action set of A_j.

It is proven in [32] that, if the learning algorithms of all the learning automata are L_{R-I}, this game would converge to a Nash equilibrium.

6. Performance Evaluation

To assess the performance of the proposed algorithms, we conducted simulations in various network scenarios using CloudSim Plus toolkit. This toolkit provides in-depth analysis and simulation of research concepts for cloud environments [37, 38]. It is an advanced version of the CloudSim simulator [39].

6.1 Simulation assumptions

The simulation model is based on several assumptions: the network consists of a fixed number of nodes (servers) that do not change during the simulation; each pair of nodes has a communication cost that is small but not negligible due to the cloud platform connection; however, the VMs of the servers can be modified and communication cost among them is negligible.

To generate the DAG of an application, we first create a root node (as the entry node) and a leaf node (as the exit node). We then divide all other nodes into different levels, with at least two nodes per level. We construct the graph gradually and randomly select up to half of the remaining nodes for each level. We also ensure that each node at a given level is connected to at least one node of the next level and vice versa. The number of tasks is randomly distributed between 10 and max_τ and there are 3 to max_μ powerful servers.

We define consistent heterogeneity as a situation where machine i (or a VM on it) is faster than machine j for any task, meaning that it can run a specific task faster than machine j.

parameters max_τ and max_μ take values of 640 and 10, respectively. So maximum 10 powerful processing servers are considered, each server consists of 20 VMs.

Total execution time (makespan), the load and the energy consumption of each machine are used for evaluating the proposed scheme. Also, a utility function U is given in equation (18).

$$U = f_T\left(T_\mu^n, T_\mu^{n-1}\right)\lambda_T + f_L\left(AVG_{load}(n), AVG_{load}(n-1)\right)\lambda_L + f_E(E^n, E^{n-1})\lambda_E \quad (18)$$

Where T_μ is the total execution time that is calculated using equation (7) in each iteration. AVG_{load} and E are computed by equation (10) and equation (12), respectively. Descriptions of Functions f_T, f_L and f_E are given in equations (15-17).

6.2 The state-of-the-art algorithms

To quantitatively compare the performance of the proposed algorithms, we selected three state-of-the-art metaheuristic algorithms: (1) load balancing technique by using modified PSO task scheduling (LBMPSO) [25], (2) modified particle swarm optimization and improved Q-learning algorithm (QMPSO) [27], and (3) multi-objective genetic algorithm (MOGA) [26]. We also chose the method proposed in [28] as LA-scheduling, which is one of the learning automata-based DAG scheduling on heterogeneous systems in [28-32]. These heuristics efficiently manage the application placement in heterogeneous network environments.

6.3 Simulation results analysis

We conduct a sensitivity analysis on the learning parameter a to study its effect on the performance of LADA and improved-LADA, and to find its optimal value. We repeat the experiment 10 times with 10 different values for a, ranging from 0.1 to 0.9 (for readability, we convert them to 1 to 9). In each repetition, LADA runs until the utility does not change for a pre-defined number of iterations . Fig. 7 shows the results. Utility is a continuous value in [0,1] and closer value to 0 indicates better performance. The best results are obtained with a between 0.4 and 0.5; smaller and larger values have worse utility values. Generally, we can say that increasing the value of a makes the probability vectors change rapidly and continuously. Conversely, decreasing the value of learning parameter slows down the learning automata and they reach the desired values very slowly.

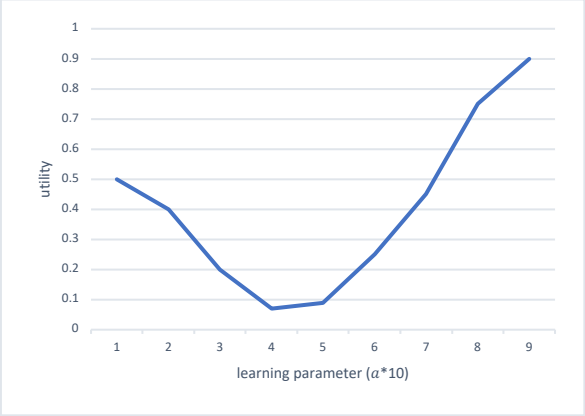


Fig. 7. Analysis the effect of learning parameter a on LADA

In the remainder of this section, the performance of LADA and improved-LADA is being evaluated in comparison against the state-of-the-art algorithms that have been mentioned.

Four aspects of the algorithms have been examined: total execution time, degree of load balance, energy consumption, and utility. The number of tasks in the experiments are varied. Fig. 8 shows the results.

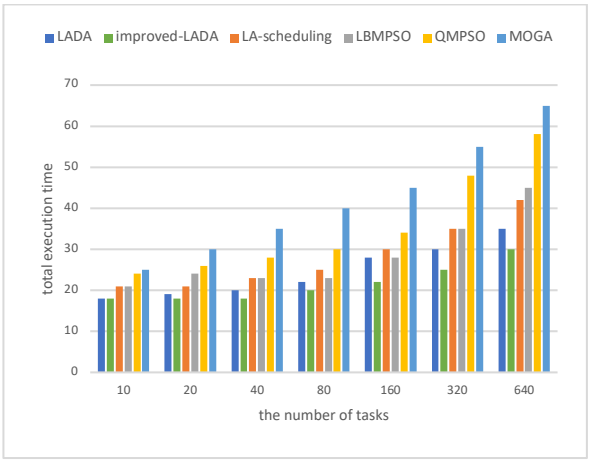
Fig. 8 (a) represents the average total execution time comparison of the algorithms as a function of workload size or the number of tasks which ranges from 10 to 640 tasks. LADA produces a lower completion time compared to LBMPSO, QMPSO, MOGA, and the LA-scheduling. The chart shows that the difference between the algorithms becomes noticeable when the workload increases; however, LADA still achieves lower makespan. The reason of this advancement relates to appropriate balancing workloads over servers. Furthermore, the algorithm continuously recomputes the execution time in each iteration until a better total finish time is reached. In addition, improved-LADA acquires even lower execution time in comparison with LADA for almost all assigned tasks. For example, in a workload with 640 tasks, improved-LADA achieves 14%, 37%, 48%, and 53% enhancement compared to LADA, LA-scheduling, LBMPSO, QMPSO and MOGA, respectively.

In order to assess the degree of load balance, we have to measure degree of load imbalance as the following:

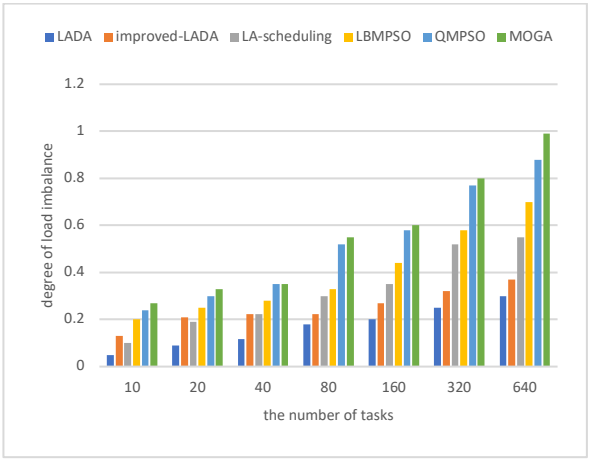
$$DoLI = \frac{L_{max} - L_{min}}{AVG_{load}}$$

Where L_{max} and L_{min} indicate maximum and minimum load, respectively and AVG_{load} denote the average load amongst allocated servers. Fig. 8 (b) compares the algorithms based on the *DoLI* metric; Both LADA and improved-LADA distribute work more evenly than the others. LADA performs better here, because improved-LADA assigns groups of dependent tasks to servers instead of assigning individual tasks, and the group lengths may vary a lot. However, improved-LADA balances the load better in larger workloads, but not as well as LADA. Overall, LADA reduces the degree of load imbalance by 18.5% to 30.30%.

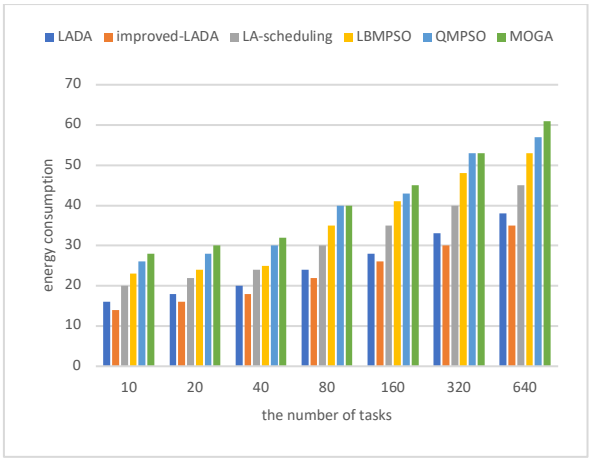
In addition to total execution time and the degree of load balancing, energy depletion is regarded as the main meter of task scheduling. Generally, energy consumption is increased significantly when the number of tasks increases. Fig. 8(c) shows how the proposed algorithms have advanced energy conservation performance analysis when tasks are assigned to machines (VMs on them). Simulation studies have represented that the energy consumption of LADA and improved-LADA is lower than that of other alternative algorithms. Totally, LADA and improved-LADA reduce energy utilization by 31.2% and 37.5%, respectively.



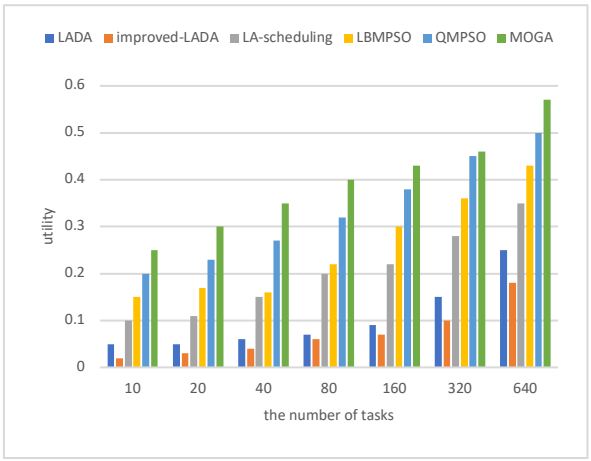
(a) total finish time (makespan) vs. number of tasks



(b) degree of load imbalance vs. number of tasks



(c) energy consumption vs. number of tasks



(d) energy consumption vs. number of tasks

Fig. 8 Analysis the performance of the algorithms -regarding to total execution time, degree of load imbalance, energy consumption, and utility via workload size

Eventually, Fig. 8(d) shows the utility value which is calculated using equation (18) as a function of total execution time, average load and average energy consumption for the algorithms. Utility is a continuous value in [0,1] and closer value to 0 indicates better performance. As expected, again LADA and improved-LADA acquire the best results.

In next studies, impact of the number of iterations on total execution time, degree of load balance, the energy consumption, and the utility have been investigated. The number of tasks and the number of available servers remained unchanged during the experiments and both of them took the maximum values; 640 for workload size and 10 for servers. The number of iterations is varied from 10 to 100.

In the beginning, total execution time is examined. Fig. 9(a) shows the proposed algorithms' superiority compared to other approaches on a different number of iterations.

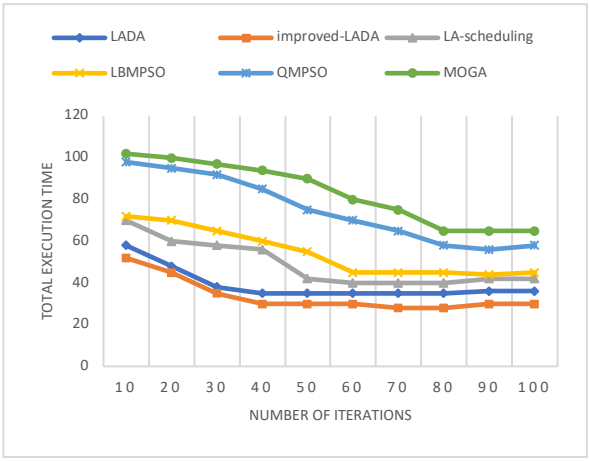
Another significant metric which is used to evaluate the efficiency of task scheduling algorithms is degree of load balance. Fig. 9(b) depicts the degree of load imbalance, its opposite measurement. The results

illustrate that the proposed algorithms perform better compared with other algorithms.

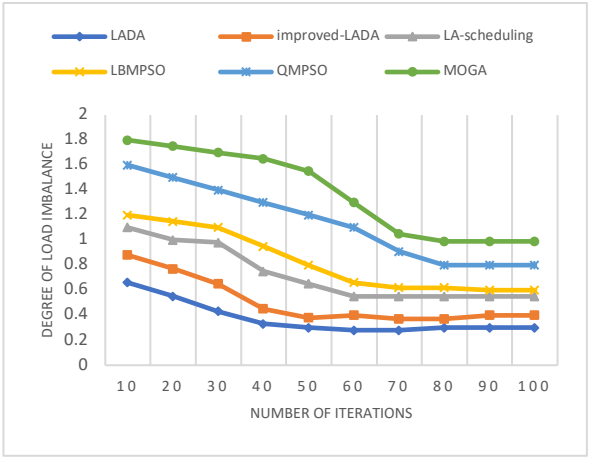
Fig. 9(c) represents energy consumption via the number of iterations. LADA and improved-LADA perform superior in both energy consumption level and convergence speed.

In the end, utility is assessed in Fig. 9(d); and as a function of three previous metrics, it shows the same outcome. The proposed algorithms achieve lower utility values, which indicates better performance.

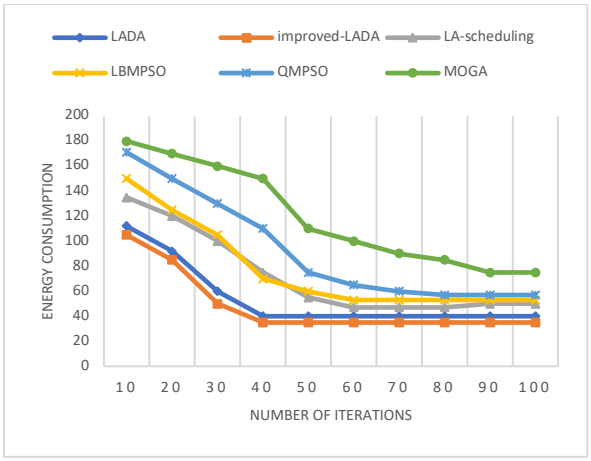
In conclusion, almost all learning automata-based algorithms converge faster than the other algorithms. LADA and improved-LADA often meet their best outcomes after 40 iterations; it is 50 and even 60 for LA-scheduling. The other algorithms converge after 70 repetitions.



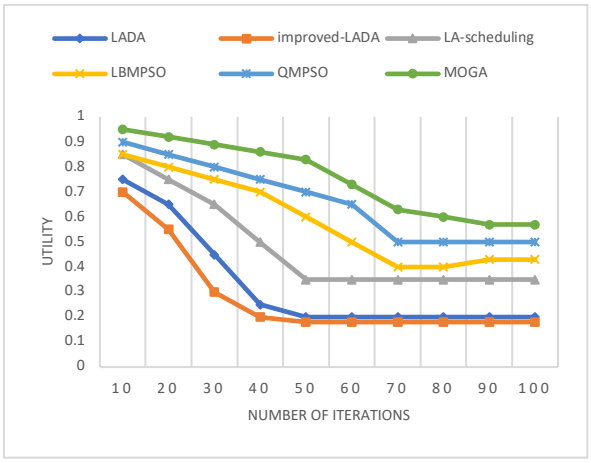
(a) total finish time (makespan) vs. number of iterations



(b) degree of load imbalance vs. number of iterations



(c) energy consumption vs. number of iterations



(d) utility vs. number of iterations

Fig. 9 Impact of the number of iterations on the performance of the algorithms

Although cloud utilization and energy consumption are linearly proportional, since servers' (VMs') idle times, which is considered in resource utilization, directly impact the cloud's efficiency and throughput, we have decided to measure cloud utilization here, unlike the most existing works. The servers' (VMs') idle times are regarded as their startup time, their shutdown time and the idle time consuming between two adjacent servers (VMs). Resource utilization is determined by equation (19).

$$U_j = \frac{\sum_{i=1}^{n} ET(i,j)}{\max\limits_{1 \le j \le \mu} (L_j(n))} \tag{19}$$

Where,

$$\max_{1 \le j \le \mu} \left(L_j(n) \right) = \sum_{i=1}^{n} ET(i,j) + t_j^{startup} + t_j^{shutdown} + t_j^{idle}$$

In later experiments resource utilization for different algorithms are evaluated. Resource utilization via workload size and via the number of iterations are shown in Fig. 10 and Fig. 11, respectively.

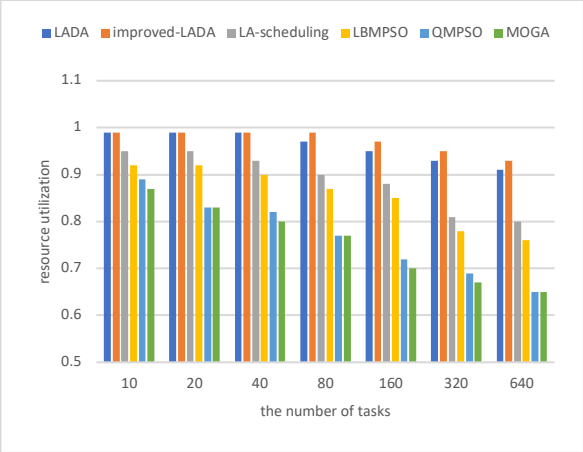


Fig. 10 resource utilization via workload size

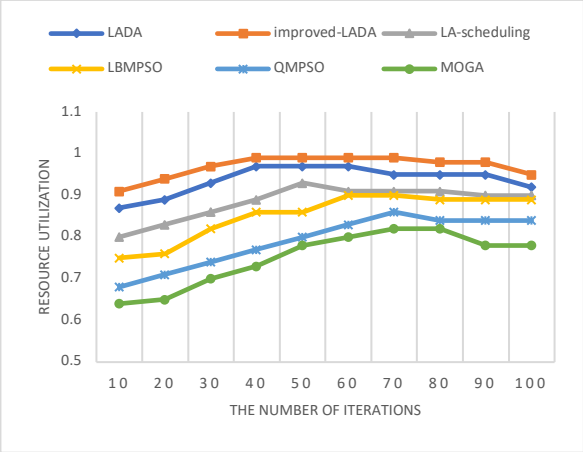


Fig. 11 resource utilization via the number of iterations.

7. Conclusion and Future Works

In this paper, a new machine learning-based algorithm for task scheduling on a heterogeneous cloud environment is proposed; it uses a game of learning automata and is named LADA. To find the best pairs of task-server (VM), first, each application is modeled as a *Directed Acyclic Graph* (DAG) in which nodes represent tasks and edges depict data dependencies between the tasks. Then, a rank value is given to each task based on its execution times and data dependency costs, regarding all possible servers. After that, concerning the rank values, independent tasks are grouped. Finally, the learning model is constructed by associating each task in the groups with a variable action-set structure learning automaton. The game of the learning automata performs, until it reaches an appropriate tasks mapping scheme based on the QoS parameters such as total execution time (makespan), load balancing, energy consumption, and cloud utilization.

An improvement to LADA is also proposed, in which the most dependent tasks are placed in one group, then a learning automaton is assigned to each group. Once more, a competitive game of learning automata runs until a proper task assignment solution is achieved.

Some experiments have been conducted to evaluate the algorithms; the results reveal the superior performance of the proposed algorithm compared with the state-of-the-art algorithms. Both LADA and its improved version reduce total execution time up to 53% and energy consumption up to 37.5%, while they increase resource utilization up to 28% and balance the load of each server. They also converge to the satisfied solution faster than the others.

For the future, writers are eager to keep on their study to improve the proposed algorithms. An important issue, which may affect the effectiveness of the scheduling, is the distribution of the tasks assigned to a server among the VMs. Another enhancement will be gained by considering a dynamic limitation for the maximum load of each server without increasing total execution time. Moreover, due to the increasing popularity of platforms such as the Internet of Things and cyber-physical systems, task scheduling on heterogeneous environments, such as cloud will continue to be one of the active topics in research; for example, special types of clouds with different criteria require a comprehensive investigation for proper scheduling tasks based on the applications.

References

[1] Hai, Tao, Jincheng Zhou, Dayang Jawawi, Dan Wang, Uzoma Oduah, Cresantus Biamba, and Sanjiv Kumar Jain. "Task scheduling in cloud environment: optimization, security prioritization and processor selection schemes." *Journal of Cloud Computing* 12, no. 1 (2023): 15.

[2] Zomaya, Albert Y. "Parallel and distributed computing handbook." (1996).

[3] Buyya, Rajkumar, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." *Future Generation computer systems* 25, no. 6 (2009): 599-616.

[4] Tziritas, Nikos, Samee Ullah Khan, Cheng-Zhong Xu, and Jue Hong. "An optimal fully distributed algorithm to minimize the resource consumption of cloud applications." In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pp. 61-68. IEEE, 2012.

[5] Ibarra, Oscar H., and Chul E. Kim. "Heuristic algorithms for scheduling independent tasks on nonidentical processors." *Journal of the ACM (JACM)* 24, no. 2 (1977): 280-289.

[6] Pradhan, Roshni, Sanjaya Kumar Panda, and Sujaya Kumar Sathua. "K-means min-min scheduling algorithm for heterogeneous grids or clouds." *International Journal of Information Processing* 9, no. 4 (2015): 89-99.

[7] Zhou, Xiumin, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, and Shiyan Hu. "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT." *Future Generation Computer Systems* 93 (2019): 278-289.

[8] Ibarra, Oscar H., and Chul E. Kim. "Heuristic algorithms for scheduling independent tasks on nonidentical processors." *Journal of the ACM (JACM)* 24, no. 2 (1977): 280-289.

[9] Casavant, Thomas L., and Jon G. Kuhl. "A taxonomy of scheduling in general-purpose distributed computing systems." *IEEE Transactions on software engineering* 14, no. 2 (1988): 141-154.

[10] Wang, Lee, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski. "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach." *Journal of parallel and distributed computing* 47, no. 1 (1997): 8-22.
[11] Topcuoglu, Haluk, Salim Hariri, and Min-You Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing." *IEEE transactions on parallel and distributed systems* 13, no. 3 (2002): 260-274.
[12] NoorianTalouki, Reza, Mirsaeid Hosseini Shirvani, and Homayun Motameni. "A heuristic-based task scheduling algorithm for scientific workflows in heterogeneous cloud computing platforms." *Journal of King Saud University-Computer and Information Sciences* 34, no. 8 (2022): 4902-4913.
[13] Ramos-Figueroa, Octavio, Marcela Quiroz-Castellanos, Efrén Mezura-Montes, and Nicandro Cruz-Ramírez. "An Experimental Study of Grouping Mutation Operators for the Unrelated Parallel-Machine Scheduling Problem." *Mathematical and Computational Applications* 28, no. 1 (2023): 6.
[14] Murad, Saydul Akbar, Abu Jafar Md Muzahid, Zafril Rizal M. Azmi, Md Imdadul Hoque, and Md Kowsher. "A review on job scheduling technique in cloud computing and priority rule based intelligent framework." *Journal of King Saud University-Computer and Information Sciences* (2022).
[15] Imene, Latreche, Slatnia Sihem, Kazar Okba, and Batouche Mohamed. "A third generation genetic algorithm NSGAIII for task scheduling in cloud computing." *Journal of King Saud University-Computer and Information Sciences* 34, no. 9 (2022): 7515-7529.
[16] Emami, Hojjat. "Cloud task scheduling using enhanced sunflower optimization algorithm." *Ict Express* 8, no. 1 (2022): 97-100.
[17] Velliangiri, S., P. Karthikeyan, VM Arul Xavier, and D. Baswaraj. "Hybrid electro search with genetic algorithm for task scheduling in cloud computing." *Ain Shams Engineering Journal* 12, no. 1 (2021): 631-639.
[18] Pradhan, Roshni, and Suresh Chandra Satapathy. "Particle Swarm Optimization-Based Energy-Aware Task Scheduling Algorithm in Heterogeneous Cloud." In *Communication, Software and Networks: Proceedings of INDIA 2022*, pp. 439-450. Singapore: Springer Nature Singapore, 2022.
[19] Natesan, Gobalakrishnan, and Arun Chokkalingam. "Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm." *ICT Express* 5, no. 2 (2019): 110-114.
[20] Shin KS, Park M-J, Jung J-Y (2014) Dynamic task assignment and resource management in cloud services by using bargaining solution. Concurr Comput Pract Exp 26(7):1432–1452
[21] E. U. Munir, S. Mohsin, A. Hussain, M.W. Nisar, and S. Ali, (2013) "SDBATS: A novel algorithm for task scheduling in heterogeneous computing systems." in Proc. IEEE IPDPS Workshops (IPDPSW).
[22] Andrei Radulescu and Arjan J. C. van Gemund. (2000) Fast and effective task scheduling in Heterogeneous system. Proceedings of the 9th Het- erogeneous Computing Workshop.
[23] Topcuouglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distributed Syst 13(3):260–274
[24] Jing Mei KL, Li K (2014) A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems. J Super- computer 68(3):1347–1377
[25] Pradhan, Arabinda, and Sukant Kishoro Bisoy. "A novel load balancing technique for cloud computing platform based on PSO." *Journal of King Saud University-Computer and Information Sciences* 34, no. 7 (2022): 3988-3995.
[26] Xia, Xuewen, Huixian Qiu, Xing Xu, and Yinglong Zhang. "Multi-objective workflow scheduling based on genetic algorithm in cloud environment." *Information Sciences* 606 (2022): 38-59.
[27] Jena, U. K., P. K. Das, and M. R. Kabat. "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment." *Journal of King Saud University-Computer and Information Sciences* 34, no. 6 (2022): 2332-2342.
[28] Moti Ghader, Habib, Davood KeyKhosravi, and Ali HosseinAliPour. "DAG scheduling on heterogeneous distributed systems using learning automata." In *Intelligent Information and Database Systems: Second International Conference, ACIIDS, Hue City, Vietnam, March 24-26, 2010. Proceedings, Part II 2*, pp. 247-257. Springer Berlin Heidelberg, 2010.
[29] Ghanbari, S., and Mohammad Reza Meybodi. "Learning automata based algorithms for mapping of a class of independent tasks over highly heterogeneous grids." In *Advances in Grid Computing-EGC 2005: European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers*, pp. 681-690. Springer Berlin Heidelberg, 2005.
[30] Venkataramana, Raju D., and N. Ranganathan. "Multiple cost optimization for task assignment in heterogeneous computing systems using learning automata." In *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, pp. 137-145. IEEE, 1999.
[31] Mirchandaney, Ravi, and John A. Stankovic. "Using stochastic learning automata for job scheduling in distributed processing systems." *Journal of Parallel and Distributed Computing* 3, no. 4 (1986): 527-552.
[32] Jahanshahi, M., M. R. Meybodi, and M. Dehghan. "A new approach for task scheduling in distributed systems using learning automata." In *2009 IEEE International Conference on Automation and Logistics*, pp. 62-67. IEEE, 2009.
[33] K. Narendra and M. A. L. Thathachar. Learning Automata: An Introduction. *Prentice Hall, Englewood Cliffs, New Jersey,* 1989.
[34] Thathachar, Mandayam AL, and P. Shanti Sastry. "Varieties of learning automata: an overview." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 32, no. 6 (2002): 711-722.
[35] Zhabelova, Gulnara, Mattias Vesterlund, Sascha Eschmann, Yulia Berezovskaya, Valeriy Vyatkin, and Damien Flieller. "A comprehensive model of data center: From cpu to cooling tower." *IEEE Access* 6 (2018): 61254-61266.
[36] Sastry, P. Shanti, Vijay V. Phansalkar, and M. A. L. Thathachar. "Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information." *IEEE Transactions on systems, man, and cybernetics* 24, no. 5 (1994): 769-777.
[37] Silva Filho, Manoel C., Raysa L. Oliveira, Claudio C. Monteiro, Pedro RM Inácio, and Mário M. Freire. "CloudSim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness." In *2017 IFIP/IEEE symposium on integrated network and service management (IM)*, pp. 400-406. IEEE, 2017.
[38] Andrade, Ermeson, and Bruno Nogueira. "Performability evaluation of a cloud-based disaster recovery solution for IT environments." *Journal of Grid computing* 17 (2019): 603-621.
[39] Goyal, Tarun, Ajit Singh, and Aakanksha Agrawal. "Cloudsim: simulator for cloud computing infrastructure and modeling." *Procedia Engineering* 38 (2012): 3566-3572.