The Journal of Supercomputing (2024) 80:24106–24137
https://doi.org/10.1007/s11227-024-06292-6

LATA: learning automata-based task assignment on heterogeneous cloud computing platform

Soulmaz Gheisari[1] · **Hamid ShokrZadeh**[2]

Accepted: 7 June 2024 / Published online: 3 July 2024

Abstract

A cloud computing environment is a distributed system where idle resources are accessible across a wide area network, such as the Internet. Due to the diverse specifications of these resources, computational clouds exhibit high heterogeneity. Task scheduling, the process of dispatching cloud applications onto processing nodes, becomes a critical challenge in such environments. Ensuring high utilization in this heterogeneous environment entails identifying suitable machines or virtual machines capable of efficiently executing jobs, constituting a multi-objective optimization problem. This paper proposes a dynamic Learning Automata-based Task Assignment algorithm, named LATA, to address this challenge. In the algorithm, each application is represented as a Directed Acyclic Graph, with tasks as nodes and data dependencies as edges. Initially, tasks are grouped based on their data dependencies to consolidate independent tasks into one group. Subsequently, a variable-structure learning automaton is assigned to each group of tasks to identify appropriate task-machine combinations. The primary objectives of LATA include minimizing makespan and energy consumption by facilitating efficient task placement to achieve load balance and maximize resource utilization. Additionally, an enhancement is proposed, involving the use of a different grouping policy prior to task assignment to further improve performance. Computer simulation results demonstrate the superior performance of the proposed algorithms in highly heterogeneous environments compared to state-of-the-art algorithms. Notably, total execution time and energy consumption decrease by up to 50% and 37%, respectively.

Keywords Cloud computing · Directed acyclic graph · Learning automata · Task scheduling

✉ Soulmaz Gheisari
S.Gheisari@soton.ac.uk

Hamid ShokrZadeh
Shokrzadeh@gmail.com

[1] Electronics and Computer Science, University of Southampton, Southampton, UK

[2] Department of Computer Engineering, Pardis Branch, Islamic Azad University, Pardis, Iran

1 Introduction

A cloud computing platform, typically operating under a "Pay for Use" policy, offers global, on-demand availability to a shared pool of resources, including storage space and computing servers with diverse specifications and high heterogeneity. Clients submit their requests to cloud environments to access these resources reliably and stress-free. In essence, the primary goal of a cloud infrastructure is to provide an easy-to-use and dependable workspace for dynamic applications used by various users. However, cloud computing exemplifies heterogeneous computing (HC) environments, characterized by dynamically fluctuating delays, changing demands for quality of service, and unpredictable behavior due to the high heterogeneity of resources and user types. Consequently, task scheduling in the cloud environment, involving the proper assignment of tasks to these highly heterogeneous servers, has become a major focal point of numerous efforts in recent years. [1–4]

In HC environments, including cloud environments, task assignment definitions vary based on the applications and platform characteristics. In such systems, users strive to achieve efficient scheduling parameters by allocating all their tasks to the available resources.

To begin with, a Directed Acyclic Graph (DAG) is used to illustrate the data dependencies among various tasks within an application. It indicates which tasks must be executed before others and the amount of data that needs to be transferred to execute subsequent tasks. These tasks are then assigned to machines or virtual machines (VMs). Each VM executes a single task at a time in a non-preemptive manner. The number of machines or VMs and the application characteristics are known in advance. Assigning tasks to the machines or VMs and scheduling their execution constitutes a nondeterministic polynomial time (NP)-hard problem [5]. This problem involves allocating tasks to machines or VMs in a manner that maximizes resource utilization, minimizes response time and energy consumption, and maintains system balance [5–10].

In this article, we propose a new Learning Automata-based algorithm for Task Assignment in cloud environments, named LATA. LATA aims to optimize the weighted sum of various metrics, including total execution time (makespan), the load of each machine, and energy consumption. While previous approaches suffer from shortcomings such as high latency, inefficient machine selection, and poor resource management, our contributions can be summarized as follows:

- Low task assignment latency: By modeling applications as DAGs and using learning automata (LATA) with time complexity O(1) for task assignment, LATA achieves higher efficiency.
- Cost-effective resource provisioning: LATA reduces makespan and energy consumption during task scheduling through reinforcement learning, facilitating the discovery of optimal task-machine pairs.
- Efficient resource management: LATA ensures efficient resource utilization, preventing both resource overloading and underloading.

Moreover, LATA offers benefits in real-world applications, such as executing user requests faster and with sufficient energy consumption. Two significant advantages are highlighted:

- Preventing global warming: By reducing energy consumption and enhancing resource efficiency, LATA contributes to reducing electricity usage in servers, thereby mitigating its impact on global warming.
- Boosting cloud provider throughput: LATA's load balancing and energy-efficient resource allocation make resource management easier for providers.

The remainder of the paper is structured as follows: Sect. 2 discusses related works, while Sect. 3 covers preliminaries such as learning automata and cloud platform models, along with definitions used in subsequent sections. Section 4 presents the proposed learning automata-based algorithm, followed by a discussion of simulation results in Sect. 5. Finally, Sect. 6 provides a summary of our work and conclusions.

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 [11] proposed a new task priority strategy and applied 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 [12], 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 [13] 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 [14], 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 [15] 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 [16] 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 [17] 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 [18], 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 [19] 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 [20], 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 [21]. 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 [10]. 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 [22], 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 [23] 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 [24]. 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 [25], 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 [26–30]. The learning process in [26] 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 [27], 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 [28] 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 [29]. The main feature of the model is applying learning automata on each host which causes low execution overhead as well as potential reliability.

In [30], 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.

A hybrid algorithm that combined Gray Wolf Optimization (GWO) and Genetic Algorithm (GA) for multi-objective task scheduling in cloud computing was proposed in [31]. The authors claimed that the algorithm minimizes energy consumption and makespan. However, load balancing was not considered.

In [32], a Chameleon and Remora Search Optimization Algorithm (CRSOA) was proposed for task scheduling in cloud environments, exploring uncertain factors such as millions of instructions per second (MIPS) and network bandwidth during the scheduling process. However, the algorithm experienced degradation in makespan and migration cost in real-world computing scenarios.

In [33], an Improved Coati Optimization Algorithm-based model for distribution and scheduling of tasks was developed, considering factors such as VMs, cost, and time. The model included a multi-objective fitness function aiming to minimize makespan while maximizing resource utilization rate. However, it could only handle task scheduling in homogeneous cloud computing environments and faced challenges in heterogeneous environments. Additionally, the model's performance decreased with an increase in the number of tasks.

Table 1 provides a concise overview of the literature. Time complexity emerges as a significant concern in many previous studies [1, 11, 12, 14, 19, 22–24, 26, 27, 31, and 33], representing a major challenge in dynamic environments like clouds. This issue often arises from the high complexity of the optimization methods employed and/or the initial scheduling states. Furthermore, several methods encounter scalability challenges regarding workload size and/or type [1, 11, 16–23, 25, 26, 28, 30–31]. Dependency on optimization algorithm parameters presents another hurdle in [15, 27–30].

In this paper, two grouping strategies are implemented to address scalability concerns. By leveraging Learning Automata (LA), whose time complexity is O(1), we aim to mitigate algorithmic complexity. While the performance of the proposed algorithm is influenced by the learning parameter of LA, employing a straightforward learning algorithm seeks to minimize its impact while preserving performance.

3 Preliminaries

In this section, we provide the necessary fundamentals of the proposed solution. Firstly, we introduce learning automata, a reinforcement learning technique pivotal to our approach. Then, we describe the scheduling system model, laying the foundation for the subsequent discussion.

3.1 Leaning automata

A learning automaton (LA) is an abstract machine learning model that falls under the category of reinforcement learning. It randomly selects one action from its finite action set and executes it within a random environment. The environment subsequently evaluates the action selected by the automaton and provides feedback in

Table 1 Summarizing the aforementioned related works

Articles	Optimization methods	Objectives	Limitations
Tao Hai et al. [1]	Heterogeneous earliest finish time (HEFT)	Improvement of HEFT Algorithm The makespan of the VMs' workflow submissions	Trade-off between time complexity and performance Dependency on problem characteristics Static problem settings Scope of optimization (resource allocation, load balancing, or fault tolerance is missing)
NoorianTalouki et al. [11]	OCTd + OCTu HEFT	Task prioritization & duplication Makespan, speedup, SLR, and efficiency	Specific application domain Scalability concerns Algorithm complexity
Ramos-Figueroa et al. [12]	GGA	Decrease Makespan	Algorithm performance
Imene et al. [14]	NSGA-III	The energy consumption and the cost	Relatively high runtime Low speed convergence
Emami et al. [15]	ESFO	Optimal scheduling with reduced complexity Makespan and energy consumption	The performance of ESFO and Algorithm Tuning Generalization of Results
Velliangiri et al. [16]	HESGA	Makespan, load balancing, utilization of resources, and cost of the multi-cloud	Simplification of real-world scenarios Certain assumptions and simplifications
Pradhan et al. [17]	PSO	QoS parameters like makespan, cloud utilization and energy consumption	Scalability: when the number of tasks and clouds increases
Natesan et al. [18]	GWO	Makespan and energy consumption Performance enhancement	Simplification: Depending solely on simulated experiments might limit the generalizability of the results to real-world scenarios
Shin et al. [19]	Game theory	Dynamic task assignment Resource management Performance improvement	Utility function assumptions Single resource type assumption Algorithmic complexity
Munir et al.[20] & Radulescu et al. [21]	FCP, FLB	Improving Performance and cost	Task model assumptions Simplifying assumptions The algorithms prioritize tasks based on specific criteria, such as earliest start time or finish time, which may not always optimize overall system performance

Table 1 (continued)

Articles	Optimization methods	Objectives	Limitations
Topcuouglu et al. [10]	HEFT-CPOP	Total execution time, quality and cost Low complexity	Assumed network structure: The algorithms assume a fully connected network Dynamic adaptation: The algorithms do not address dynamic changes in system conditions
Jing Mei et al. [22]	RADS	Resource-aware scheduling Reduced task duplication	Simplifying assumptions Algorithm complexity Scalability
Pradhan et al. [23]	LBMPSO	Optimal task scheduling Load balancing Resource utilization and makespan	Complexity Scalability Resource constraints Dependency handling and performance overhead
Xia et al. [24]	MOGA	Makespan And Energy Consumption	Algorithm performance
Jena et al. [25]	MPSO, QMPSO	Resource utilization Response time Energy consumption and QoS	Simplified model Scalability
Moti et al. [26]	DAG,LA	Distributed heterogeneous scheduling Optimizing Finish Time	Complexity Scalability Algorithmic efficiency
Ghanbari et al. [27]	LA	Optimization of task assignment	Dependency on proper learning algorithm selection Sensitivity to parameter tuning Computational cost
Venkataramana et al. [28]	LA	Optimizing multiple cost criteria simultaneously Efficient task assignment	Sensitivity to reward and penalty parameters Complexity of cost metrics optimization Performance and scalability
Mirchandaney et al. [29]	SLA	Development of decentralized job scheduling algorithm Improvement of SLA model and performance Low execution overhead	Communication overhead Dependency on feedback Sensitivity to parameter tuning

Table 1 (continued)

Articles	Optimization methods	Objectives	Limitations
Jahanshahi et al. [30]	LA	Communication cost CPU utilization and makespan	Scalability Simplification of environment Sensitivity to reward and penalty parameters
Ipsita et al. [31]	GA-GWO	Cost, makespan, and energy consumption	Complexity Scalability
Pabitha et al. [32]	CRSOA	Task completion rate, load balance, scheduling cost and makespan Handling the load balancing	Scalability Theoretical assumptions simplifications
Tamilarasu et al. [33]	ICOATS	Cost, time, makespan Enhances the turnaround efficiency	Homogeneous environment Complexity handling Limited factor consideration

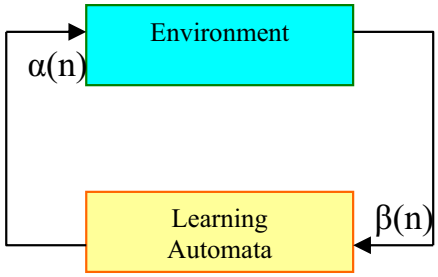


Fig. 1 Learning automata connection with environment [34]

the form of a reinforcement signal. Using this feedback and the selected action, the automaton updates its internal state and determines its next action. The interaction between the learning automaton and the environment is depicted in Fig. 1 [34, 35].

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. Environments where β takes only binary values (0 or 1) are referred to as P-models. If the output set comprises more than two elements, each taking values in the interval [0, 1], the environment is known as a Q-model. Lastly, when the output is a continuous random variable in the interval [0, 1], it is termed an S-model environment. Additionally, learning automata are classified into fixed-structure [34] and variable-structure types; here, the variable-structure learning automata (VSLA) is considered in an S-model environment.

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, $\beta=\{\beta_1, \beta_2, \ldots, \beta_n\}$ denotes the input set, $p=\{p_1, p_2, \ldots, p_r\}$ represents 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 feedback from the environment in the form of a reinforcement signal, the automaton updates its action probability set based on Eq. (1) for favorable responses and Eq. (2) for unfavorable ones.

$$p_i(n+1) = p_i(n) + a\big(1 - p_i(n)\big) \tag{1}$$

$$p_j(n+1) = p_j(n) + a\big(p_j(n)\big) \forall jj \neq i$$

$$p_i(n+1) = (1-b)p_i(n) \tag{2}$$

$$p_j(n+1) = \frac{b}{r-1} + (1-b)p_j(n) \forall jj \neq i$$

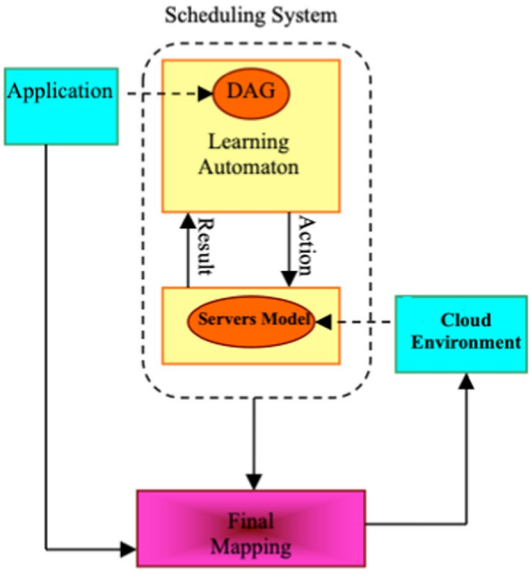


Fig. 2 model of scheduling system in the cloud computing platform

In these two equations, a and b represent reward and penalty parameters, respectively. If $a=b$, the 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 [34].

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 [35]. Let $A_1, \ldots, A_n$ be the automata involved in an N-player game. Each play of the game consists of choosing an action by each learning automaton and then getting the receiving 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 [35]. In this paper, a game of τ learning automata is applied to map τ tasks to μ machines or VMs on them.

3.2 Scheduling system model

This section describes a scheduling system model in cloud computing environments, consisting of heterogeneous servers. Figure 2 shows the schematic representation

[1] Linear Reward-Penalty.

[2] Linear Reward epsilon Penalty.

[3] Linear Reward Inaction.

of the environment. The model comprises a cloud environment, executable applications, and a central scheduling system. The scheduling system consists of learning automata, a directed acyclic graph (DAG) representing various tasks of an application and the data dependencies between them, and a server model illustrating machines as well as communication costs between them (machine-to-machine data transfer costs). The tasks of the DAG are assigned to the servers through a game of learning automata. Then, cost metrics such as total execution time, average server load, and energy consumption [14, 17, 24, 32] are applied to calculate the reinforcement signals, which are fed back to the learning automata. Each automaton updates its action probability set based on the received signal. This process repeats until the scheduling system effectively assigns all tasks to the servers (VMs). This model can adapt successfully to changes in applications, servers including VMs, or the network.

3.2.1 Definitions

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 total number of tasks in the DAG. $DD(i, j)$ 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. $\mathrm{ET}(i,j)$ 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 element in the DAG w_i is computed by Eq. (3).

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

The cost of communication between any two machines, as well as between VMs on different machines, is also known as a priori and is represented by a $\mu\times\mu$ matrix named CC.

Let $\psi(i) = j$ be defined as the assignment of task i to machine j. Therefore, transfer cost between two tasks i and j, denoted as $\theta(i,j)$, is computed as $\theta(i,j) = \mathrm{DD}(i,j) \times \mathrm{CC}(\psi(i), \psi(j))$. If there is no certain assignment, it is calculated using Eq. (4):

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

where p_k and p_l are the probabilities of choosing machine k and machine l, respectively.

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

3.2.2 Cost metrics

In order to evaluate task assignment performance, several metrics must be considered, including total execution time (makespan), the load of each machine, and energy consumption. The objective 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 among all μ machines is referred to as the makespan; therefore, to minimize the makespan, the load on the most heavily loaded machine should be minimized, indicating that the load of all machines should be balanced.

If the total load of a particular machine, such as j, at iteration n is represented as $L_j(n)$, then we have Eq. (5).

$$L_j(n) = \mathrm{FT}\left(t_i, j\right) \tag{5}$$

where t_i is the last task executed on machine j; $\mathrm{FT}\left(t_i, j\right)$ represents the finish time when task t_i on machine j terminates, and it is defined by Eq. (6).

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

where $\mathrm{ST}(t_i, j)$ represents the time when t_i can start executing on machine j. Accordingly, Eq. (7) is used to calculate the makespan T_{μ} at iteration n.

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

To balance the servers' load, the ideal average of load δ_{ideal} is computed. Let η illustrate the average of transfer cost for all tasks, it is calculated as follows:

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

Then,

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

where t_j^{idle} represents the idle time between two adjacent VMs on machine j. Equation (10) calculates the average load of all machines at iteration n. The closer $\mathrm{AVG}_{\text{load}}(n)$ is to δ_{ideal}, the better the task scheduling of iteration n.

$$\mathrm{AVG}_{\text{load}}(n) = \frac{\sum_{j=1}^{\mu} L_j(n)}{\mu} \tag{10}$$

Eventually, an energy model is needed to estimate the total amount of energy consumption used by the scheduling method. To define the energy model, power and machine utilization are formulated [36], as the average of cloud utilization and

energy consumption are linearly proportional. Energy consumption for machine j is mathematically defined by Eq. (11).

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

where U_j depicts the utilization of machine j and is calculated using, $U_j = \frac{\sum_{i=1}^{n} \mathrm{ET}(i,j)}{\max\limits_{1 \le j \le \mu} \left(L_j(n) \right)}$; $P_{\max}$ represents the maximum power consumption when the machine is fully loaded, and $P_{\min}$ is the minimum power consumption when the machine is idle state. It is important to note that workload affects the power supply value over time. The average energy consumption in iteration n is calculated using Eq. (12).

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

4 Research methodology

Our research methodology section offers insights into the innovative strategies employed in LATA, including task grouping based on data dependencies and the dynamic assignment of learning automata to optimize task placement and workload distribution.

4.1 Learning automata-based task assignment algorithm

In this section, we propose a new task scheduling algorithm based on learning automata for cloud environments.

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

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

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

where S_i represents the set of immediate successors of node i in the 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 a higher number is created, and the last node is moved to it. The final outcome is a set of ordered groups, each containing

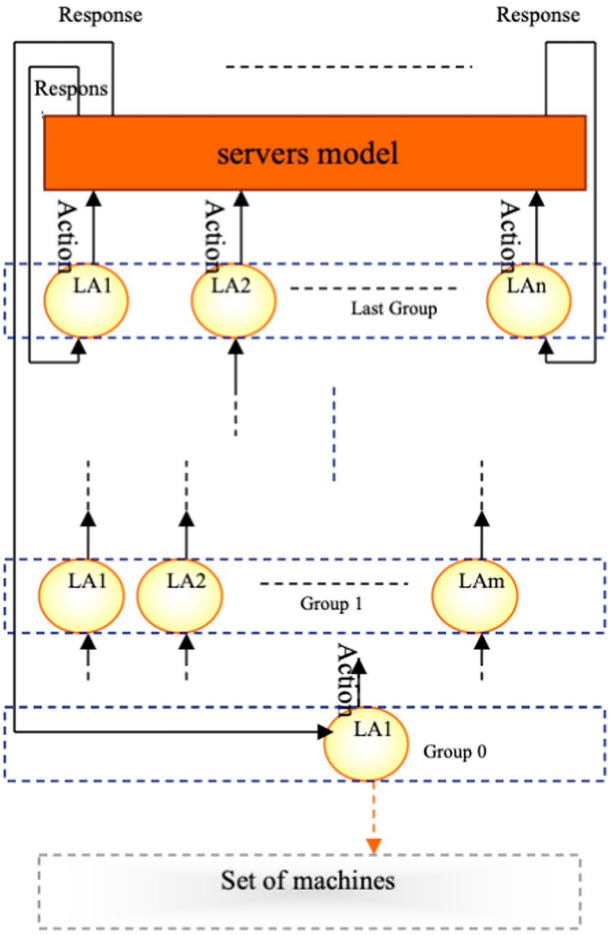


Fig. 3 The learning automata model

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 straightforward approach 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 implies that the tasks of group ii will not be scheduled until all the tasks of group i-1 are scheduled.

In contrast, 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 the input value, β_i is a continuous value in the range [0,1]; a β_i closer to 0 indicates that the action taken by the automaton is favorable to the system, while closer to 1 indicates an unfavorable response. Figure 3 illustrates the model.

The reinforce scheme used to update the action probabilities of learning automata is L_{RI}.

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 is calculated using the equations introduced in Sect. 4.2. Then, the results are provided to the learning automata as reinforcement signals to update their action probabilities. This process is repeated until a termination condition is met.

To evaluate the quality of the selected actions (servers) via β_i, each automaton uses a generalized formulation, given by Eq. (14). Since the problem is multi-objective [15, 26, 33, and 35], 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.

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

where 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 impact the variation in makespan has on the evaluation of the environment response. This statement holds true for both λ_L and λ_E. Functions f_T, f_L and f_E are given in Eqs. (15–17). As mentioned before, we use S-model for learning automata.[5]

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

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

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

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

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

[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 Eq. (9).

Algorithm 1 LATA

```
//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, the learning process will stop; in the first and the second conditions, the automaton converges. Algorithm 1 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 the 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.

4.2 Improved LATA

To enhance the performance of the algorithm, the second and third phases are modified as follows:

Second phase, group creation: unlike before, tasks are now grouped in a manner that places tasks with the highest dependencies in the same sets. To accomplish 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 Eq. (3).

(2) C_i as a successor or an ordered list of successors with the most transfer cost:

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

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). $\theta(i,j)$ is given in equation (4).

(3) P_i as a parent or an ordered list of parents with the most transfer cost:

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

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 that indicates whether 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 i does not belong to a group, a new group is created, and i is placed into it. Then, other node 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 j where $P_j = i$ are updated using the following equations.

$$\text{if } C_i = j \;\&\; \&\; P_j = i \text{ then}$$

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

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

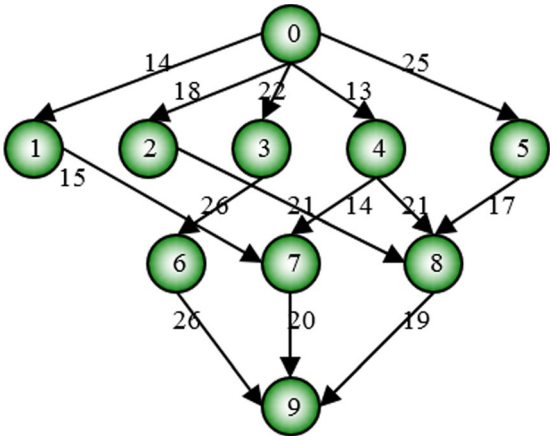
where,

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

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

4.2.1 An example

Figure 4 represents a sample DAG. The numbers written near the edges correspond to the amount of data that must be passed from one 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. Figure 5 shows the 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

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

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

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

In each iteration, the game of learning automata selects machines, after which makespan, load, and energy consumption of each machine are computed. These

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

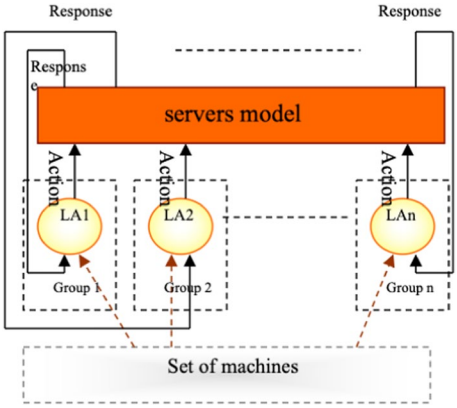


Fig. 6 learning automata model

results are then passed to the learning automata to update their action probabilities. This process is repeated until one of the termination conditions is fulfilled. To determine the efficacy of an action taken by an automaton, Eq. (14) is applied. The stopping criteria for the learning algorithm remain the same as before. Algorithm 2 clarifies the above steps.

4.3 Time complexity analysis

The algorithms' complexity is estimated in three steps: ranking, group creation, and tasks assignment. Let n be the number of iterations, τ be the number of tasks, and μ be the number of machines. To calculate the ranking value, we have $O\left(n \times \left(\mu + \mu^2\right)\right)$, 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 the total time complexity is $O(n(\tau \log \tau + \tau))$, simplified 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, LATA's time complexity is considered as $O(n \times (\mu^2 + \tau \log \tau))$.

Algorithm 2 Improved-LATA

```
// 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++;
```

4.4 Convergence analysis

While the game of learning automata is competitive, and each automaton receives its own payoff and strives to improve it, it is expected that the algorithm reaches a *Nash equilibrium*. For a comprehensive discussion on Nash equilibrium, please refer to [37]. The m-tuple of actions $(\alpha_1, \ldots, \alpha_m)$ is called a Nash equilibrium of this game if for each $j, 1 \leq j \leq m$, $d^j(\alpha_1, \ldots, \alpha_{j-1}, x, \alpha_{j+1}, \ldots, \alpha_m) \leq d^j(\alpha_1, \ldots, \alpha_m), \forall x \in S_i$.

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

It is proven in [37] that if the learning algorithms of all the learning automata are $L_{R\text{-}I}$, this game would converge to a Nash equilibrium.

5 Performance evaluation

To evaluate the performance of the proposed algorithms, we conducted simulations in various network scenarios using the CloudSim Plus toolkit. This toolkit offers comprehensive analysis and simulation capabilities for research concepts in cloud environments [38, 39]. It represents an advanced iteration of the CloudSim simulator [40].

5.1 Simulation assumptions

The simulation model operates on several assumptions: the network comprises a static number of nodes (servers) that remain constant throughout the simulation; each pair of nodes incurs a communication cost, which, while small, is not negligible given the connection within the cloud platform; however, the virtual machines (VMs) hosted on the servers can be adjusted, and the communication cost among them is considered negligible.

To generate the Directed Acyclic Graph (DAG) of an application, we initially establish a root node (representing the entry point) and a leaf node (representing the exit point). Subsequently, we partition the remaining nodes into distinct levels, ensuring each level comprises a minimum of two nodes. The graph is constructed incrementally, with up to half of the remaining nodes randomly selected for each level. Additionally, we ensure that each node within a level is interconnected with at least one node in the subsequent level, and vice versa. The number of tasks is randomly allocated between 10 and a maximum value, max_τ, while there are between 3 and a maximum of max_μ powerful servers available.

In consistent heterogeneity, machine i (or a VM on it) consistently outperforms machine j for any task, indicating that it can execute a given task more quickly than machine j.

The parameters $\max_\tau$ and $\max_\mu$ take values of 640 and 10, respectively. Thus, a maximum of 10 powerful processing servers are considered, with each server comprising 20 VMs. Total execution time (makespan), the load, and the energy

consumption of each machine are used for evaluating the proposed scheme. Additionally, a utility function U is defined by Eq. (18).

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

where T_μ represents the total execution time calculated using Eq. (7) in each iteration. $\mathrm{AVG}_{\mathrm{load}}$ and E are computed by Eqs. (10) and (12)), respectively. Descriptions of Functions f_T, f_L and f_E are given in Eqs. (15–17).

5.2 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) [23], (2) modified particle swarm optimization and improved Q-learning algorithm (QMPSO) [25], and (3) multi-objective genetic algorithms, (MOGA) and (GA-GWO) [24, 31]. Additionally, we included the method proposed in [26] as LA-scheduling, which is one of the learning automata-based DAG scheduling approaches for heterogeneous systems in [26–30]. These heuristics efficiently manage the application placement in heterogeneous network environments.

5.3 Simulation results analysis

We conduct a sensitivity analysis on the learning parameter a to study its effect on the performance of LATA and improved-LATA, 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 1–9). In each repetition, LATA runs until the utility does not change for a pre-defined number of iterations. Figure 7 shows the results. Utility is a continuous value in [0, 1], and a 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 the learning parameter slows down the learning automata, and they reach the desired values very slowly.

In the remainder of this section, we evaluate the performance of LATA and improved-LATA in comparison with the state-of-the-art algorithms mentioned earlier.

Four aspects of the algorithms were examined: total execution time, degree of load balance, energy consumption, and utility, with variations in the number of tasks in the experiments. Figure 8 shows the results. Figure 8a displays the average total execution time comparison of the algorithms as a function of workload size or the number of tasks, ranging from 10 to 640 tasks. LATA consistently yields lower completion times compared to LBMPSO, QMPSO, MOGA, and the LA-scheduling method. The chart reveals that the differences between the algorithms become

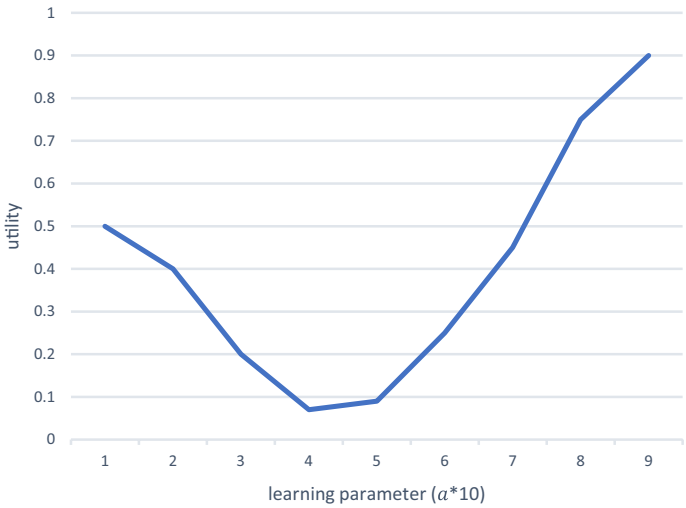


Fig. 7 Analysis the effect of learning parameter a on LATA

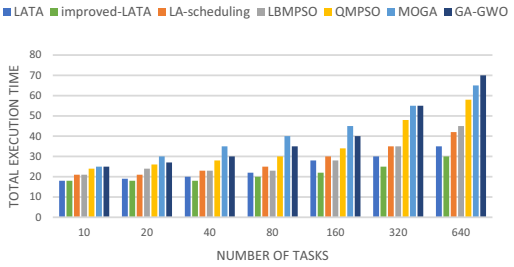
more pronounced as the workload increases; however, LATA consistently achieves a lower makespan. This improvement can be attributed to the effective workload balancing across servers facilitated by the algorithm. Moreover, the algorithm continuously recalculates the execution time in each iteration until a better total finish time is achieved. Additionally, improved-LATA achieves even lower execution times compared to LATA for almost all assigned tasks. For instance, in a workload with 640 tasks, improved-LATA achieves enhancements of 14%, 37%, 48%, and 53% compared to LATA, LA-scheduling, LBMPSO, QMPSO, and MOGA, respectively.

To assess the degree of load balance, the degree of load imbalance is measured as follows:

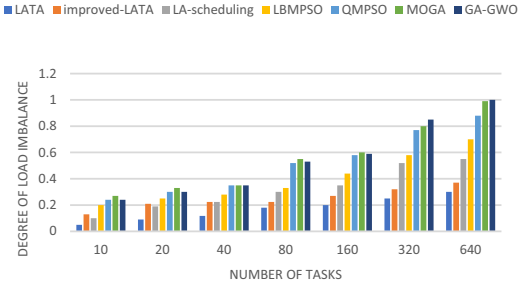
$$\mathrm{DoLI} = \frac{L_{\max} - L_{\min}}{\mathrm{AVG}_{\mathrm{load}}} \tag{19}$$

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

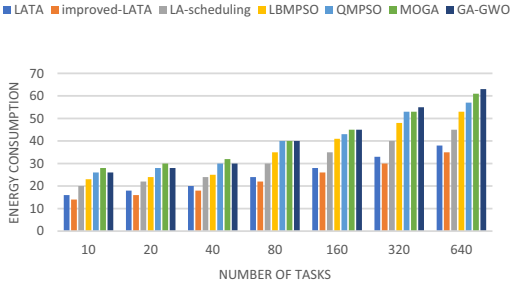
In addition to total execution time and the degree of load balancing, energy consumption is regarded as the main meter of task scheduling. Generally, energy consumption increases significantly as the number of tasks increases. Figure 8c shows how the proposed algorithms have advanced energy conservation performance analysis when tasks are assigned to machines (VMs on them). Simulation studies have



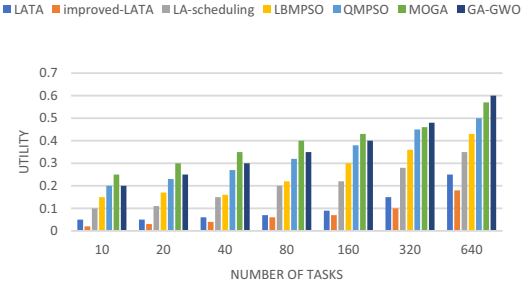
(a) total finish time (makespan) vs. number of tasks



(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

shown that the energy consumption of LATA and improved-LATA is lower than that of other alternative algorithms. Overall, LATA and improved-LATA reduce energy utilization by 31.2% and 37.5%, respectively.

Eventually, Fig. 8d shows the utility value, calculated using Eq. (18) as a function of total execution time, average load and average energy consumption for the algorithms. Utility is a continuous value in the range [0, 1], with values closer to 0 indicating better performance. As expected, once again LATA and improved-LATA achieve the best results.

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

At the beginning, the total execution time is examined. Figure 9a illustrate the superiority of the proposed algorithms compared to other approaches across different numbers of iterations.

Another significant metric which is used to evaluate the efficiency of task scheduling algorithms is degree of load balance. Figure 9b depicts the degree of load imbalance, its opposite measurement. The results illustrate that the proposed algorithms perform better compared with other algorithms.

Figure 9c 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. 9d; 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.

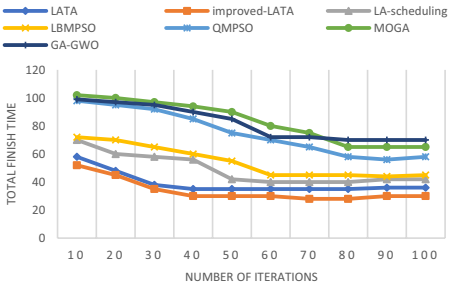
Although cloud utilization and energy consumption are linearly proportional, since servers' (VMs') idle times, which is considered in resource utilization, directly impact the cloud's efficiency and throughput, we have decided to measure cloud utilization here, unlike the most existing works. The servers' (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 (20)

$$U_j = \frac{\sum_{i=1}^{n} ET(i,j)}{\max\limits_{1\le j\le \mu}\left(L_j(n)\right)} \tag{20}$$

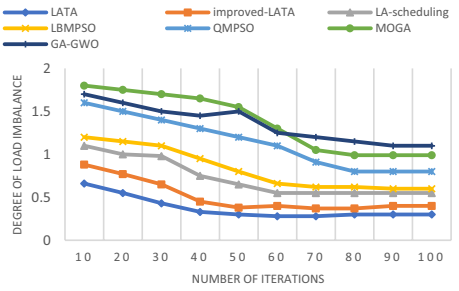
Where,

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

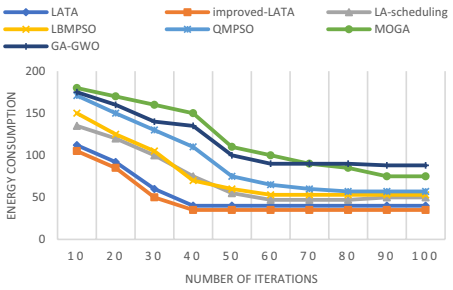
Fig. 9 Impact of the number of iterations on the performance of the algorithms



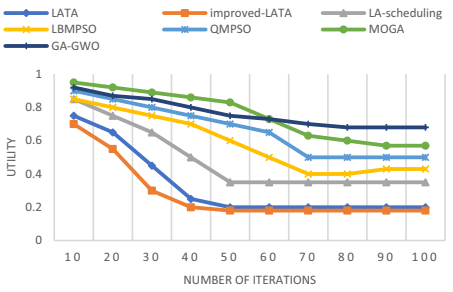
(a) total execution time (makespan) vs. number of iterations



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



(c) energy consumption vs. number of iterations



(d) utility vs. number of iterations

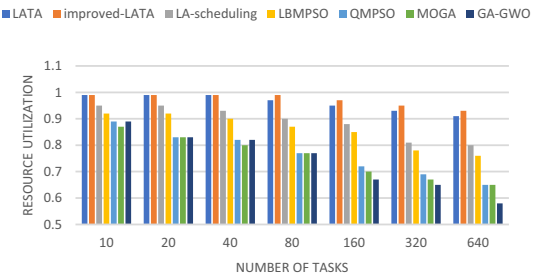


Fig. 10 resource utilization via workload size

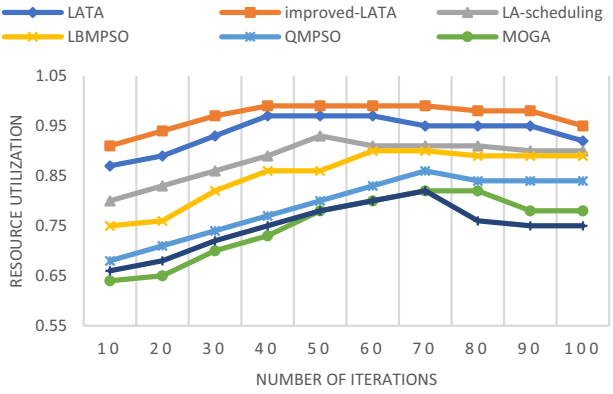


Fig. 11 resource utilization via the number of iterations

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

6 Conclusion and future works

In this paper, we propose a new machine learning-based algorithm for task scheduling in heterogeneous cloud environments, named LATA. This algorithm utilizes a game of learning automata to find the best pairs of tasks and servers (VMs). Initially, each application is represented as a Directed Acyclic Graph (DAG), where nodes represent tasks and edges depict data dependencies. Tasks are then assigned rank values based on their execution times and data dependency costs across all possible servers. Independent tasks are grouped based on these rank values, and each task within the groups is associated with a variable action-set structure learning automaton. The game of learning automata iteratively refines the task-server mapping until optimal solutions are found, considering QoS parameters such as total execution time (makespan), load balancing, energy consumption, and cloud utilization.

An improvement to LATA is also proposed, wherein the most dependent tasks are grouped together, and each group is assigned a learning automaton. This modified approach aims to enhance the efficiency of task assignment by focusing on highly dependent tasks within each group.

Experimental evaluations were conducted to assess the performance of the proposed algorithms. Results indicate that both LATA and its improved version significantly reduce total execution time (up to 53%) and energy consumption (up to 37.5%), while also improving resource utilization (up to 28%) and load balancing across servers. Additionally, these algorithms converge to satisfactory solutions faster than existing methods.

For future research directions, we aim to further enhance the proposed algorithms by investigating the distribution of tasks among VMs within a server to optimize resource allocation. Additionally, dynamic limitations on maximum server load could be incorporated without compromising total execution time. Furthermore, with the rising popularity of platforms like the Internet of Things and cyber-physical systems, task scheduling in heterogeneous environments will continue to be a key research area, requiring comprehensive investigations for tailored task scheduling based on specific application requirements.

Author contributions Both authors conceived the presented idea. Soulmaz Gheisari developed the theory, conducted the implementation and computations, and prepared the initial manuscript. Hamid Shokrzadeh contributed to the study by summarizing and discussing related works. Both authors verified the proposed solution and evaluation method, discussed the results, and contributed to the final manuscript.

Funding This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Declarations

Conflict of interest The authors declare that they have no competing interests.

References

1. Tao H, Zhou J, Jawawi D, Wang D, Oduah U, Biamba C, Kumar Jain S (2023) Task scheduling in cloud environment: optimization, security prioritization and processor selection schemes. J Cloud Comput 12(1):15
2. Zomaya AY (1996) Parallel and distributed computing handbook.

3. Rajkumar B, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
4. Nikos T, Khan SU, Xu CZ, and Hong J (2012) 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
5. Ibarra OH, Kim CE (1977) Heuristic algorithms for scheduling independent tasks on nonidentical processors. J ACM (JACM) 24(2):280–289
6. Roshni P, Panda SK, Sathua SK (2015) K-means min-min scheduling algorithm for heterogeneous grids or clouds. Int J Inf Process 9(4):89–99
7. Zhou X, Zhang G, Sun J, Zhou J, Wei T, Shiyan Hu (2019) Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. Future Gener Comput Syst 93:278–289
8. Casavant TL, Kuhl JG (1988) A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans Software Eng 14(2):141–154
9. Lee W, Siegel HJ, Roychowdhury VP, Maciejewski AA (1997) Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. J Parallel Distrib Comput 47(1):8–22
10. Topcuoglu H, Hariri S, Min-You Wu (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
11. Reza NT, Shirvani MH, Motameni H (2022) A heuristic-based task scheduling algorithm for scientific workflows in heterogeneous cloud computing platforms. J King Saud Univ Comput Info Sci 34(8):4902–4913
12. Ramos-Figueroa O, Quiroz-Castellanos M, Mezura-Montes E, Cruz-Ramírez N (2023) An experimental study of grouping mutation operators for the unrelated parallel-machine scheduling problem. Mathe Comput Appl 28(1):6
13. Akbar MS, Muzahid AJM, Hoque AMI, Kowsher M (2022) A review on job scheduling technique in cloud computing and priority rule based intelligent framework. J King Saud Univ Comput Sci 34:2309
14. Imene L, Sihem S, Okba K, Mohamed B (2022) A third generation genetic algorithm NSGAIII for task scheduling in cloud computing. J King Saud Univ-Comput Info Sci 34(9):7515–7529
15. Emami H (2022) Cloud task scheduling using enhanced sunflower optimization algorithm. Ict Express 8(1):97–100
16. Velliangiri S, Karthikeyan P, Arul Xavier VM, Baswaraj D (2021) Hybrid electro search with genetic algorithm for task scheduling in cloud computing. Ain Shams Eng J 12(1):631–639
17. Roshni P, and Satapathy SC (2022) Particle swarm optimization-based energy-aware task scheduling algorithm in heterogeneous cloud. In: communication, software and networks: Proceedings of India 2022. Springer Nature, Singapore, pp 439–450
18. Natesan G, Chokkalingam A (2019) Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm. ICT Express 5(2):110–114
19. 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
20. Munir EU, Mohsin S, Hussain A, Nisar MW, and Ali S (2013) SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems. In: Proceedings IEEE IPDPS workshops (IPDPSW)
21. Radulescu A and van Gemund AJC (2000) Fast and effective task scheduling in heterogeneous system. In: Proceedings of the 9th heterogeneous computing workshop
22. Jing Mei KL, Li K (2014) A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems. J Super Comput 68(3):1347–1377
23. Arabinda P, Bisoy SK (2022) A novel load balancing technique for cloud computing platform based on PSO. J King Saud Univ Comput Inf Sci 34(7):3988–3995
24. Xia X, Qiu H, Xing Xu, Zhang Y (2022) Multi-objective workflow scheduling based on genetic algorithm in cloud environment. Inf Sci 606:38–59
25. Jena UK, Das PK, Kabat MR (2022) Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. J King Saud Univ Comput Inf Sci 34(6):2332–2342
26. Habib MG, KeyKhosravi D, and Hosseinalipour A (2010) 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. Springer, Berlin Heidelberg, pp 247–257
27. Ghanbari S, and Meybodi MR (2005) 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. Springer, Berlin Heidelberg, pp 681–690
28. Venkataramana RD, and Ranganathan N (1999) Multiple cost optimization for task assignment in heterogeneous computing systems using learning automata. In: Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99). IEEE, pp 137–145
29. Mirchandaney R, Stankovic JA (1986) Using stochastic learning automata for job scheduling in distributed processing systems. J Parallel Distrib Comput 3(4):527–552
30. Jahanshahi M, Meybodi MR, and Dehghan M (2009) A new approach for task scheduling in distributed systems using learning automata. In: 2009 IEEE International Conference on Automation and Logistics. IEEE, pp 62–672009
31. Behera I, Sobhanayak S (2024) Task scheduling optimization in heterogeneous cloud computing environments: a hybrid GA-GWO approach. J Parallel Distrib Comput 183:104766
32. Pabitha P, Nivitha K, Gunavathi C, Panjavarnam B (2024) A chameleon and remora search optimization algorithm for handling task scheduling uncertainty problem in cloud computing. Sustain Comput Inf Syst 41:100944
33. Tamilarasu P, Singaravel G (2023) Quality of service aware improved coati optimization algorithm for efficient task scheduling in cloud computing environment. J Eng Res. https://doi.org/10.1016/j.jer.2023.09.024
34. Narendra K, Thathachar MAL (1989) Learning automata: an introduction. Prentice Hall, Englewood Cliffs, New Jersey
35. Thathachar MAL, Sastry PS (2002) Varieties of learning automata: an overview. IEEE Trans Syst Man Cybern Part B (Cybernetics) 32(6):711–722
36. Zhabelova G, Vesterlund M, Eschmann S, Berezovskaya Y, Vyatkin V, Flieller D (2018) A comprehensive model of data center: from CPU to cooling tower. IEEE Access 6:61254–61266
37. Sastry PS, Phansalkar VV, Thathachar MAL (1994) Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. IEEE Trans Syst man Cybern 24(5):769–777
38. Filho S, Manoel C, Oliveira RL, Monteiro CC, Inácio PRM, and Freire MM (2017) 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). IEEE, pp 400–406
39. Andrade E, Nogueira B (2019) Performability evaluation of a cloud-based disaster recovery solution for IT environments. J Grid Comput 17:603–621
40. Tarun G, Singh A, and Agrawal A (2012) Cloudsim: simulator for cloud computing infrastructure and modeling. In: Procedia engineering, 38: 3566-3572