Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Intelligent multi-agent reinforcement learning model for resources allocation in cloud computing

Ali Belgacem [a,*], Saïd Mahmoudi [b], Maria Kihl [c]

[a] *M'hamed Bougara University, Boumerdes, Algeria*
[b] *Mons University, Mons, Belgium*
[c] *Lund University, Lund, Sweden*

ARTICLE INFO

Article history:
Received 15 December 2021
Revised 11 February 2022
Accepted 13 March 2022
Available online 2 April 2022

Keywords:
Cloud computing
Resource allocation
Multi-agent system
Q-learning
Energy consumption
Fault tolerance
Load balancing

ABSTRACT

Now more than ever, optimizing resource allocation in cloud computing is becoming more critical due to the growth of cloud computing consumers and meeting the computing demands of modern technology. Cloud infrastructures typically consist of heterogeneous servers, hosting multiple virtual machines with potentially different specifications, and volatile resource usage. This makes the resource allocation face many issues such as energy conservation, fault tolerance, workload balancing, etc. Finding a comprehensive solution that considers all these issues is one of the essential concerns of cloud service providers. This paper presents a new resource allocation model based on an intelligent multi-agent system and reinforcement learning method (IMARM). It combines the multi-agent characteristics and the Q-learning process to improve the performance of cloud resource allocation. IMARM uses the properties of multi-agent systems to dynamically allocate and release resources, thus responding well to changing consumer demands. Meanwhile, the reinforcement learning policy makes virtual machines move to the best state according to the current state environment. Also, we study the impact of IMARM on execution time. The experimental results showed that our proposed solution performs better than other comparable algorithms regarding energy consumption and fault tolerance, with reasonable load balancing and respectful execution time.

1. Introduction

Cloud computing is a development of information technology (IT) that allows remote access to computing resources. It permits sharing the resources of several computers cooperatively (Velte et al., 2010). These resources can be dynamically configured according to the workload, allowing for optimal use of resources and their provision as a service to external customers (Buyya et al., 2010). Utilizing cloud services to enhance the competitiveness of companies has become a prominent trend across the world. With cloud computing, individuals and organizations can gain on-demand network access to a shared pool of managed and scalable IT resources, such as servers, storage, and applications (Sunyaev, 2020). Furthermore, in 2020, the interest in the cloud has increased, given its use to predict the growth and trend of the COVID-19 pandemic (Tuli et al., 2020). For this reason, the cloud is required not only for Internet services but also for the IT sector as a whole (Dikaiakos et al., 2009; Belgacem et al., 2020).

Cloud computing has a distributed architecture; the available resources can be located on different physical machines. Processing is spread across multiple servers, more generally, across multiple virtual machines (VMs). The latter are arranged in different configurations, with each group of them running on a single physical machine (PM) (Velte et al., 2010; Belgacem and Beghdad-Bey, 2021; Belgacem et al., 2018). The underlying technology responsible for this is virtualization. It includes a set of technologies and tools that facilitate data center infrastructure management. This technology permits provisioning, migration, and consolidation of VMs within a few seconds. It saves time and makes the service alive for customers, which makes the achievement of service level agreements and quality-of-service (QoS) specifications more

* Corresponding author.
E-mail addresses: a.belgacem@univ-boumerdes.dz (A. Belgacem), Said.MAHMOUDI@umons.ac.be (S. Mahmoudi), maria.kihl@eit.lth.se (M. Kihl).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

possible. Therefore, enhancing cloud resource allocation performance is an important research area.

The optimization-based agent is a critical technique widely used in many disciplines, such as artificial intelligence and mainstream computer science (Bellifemine et al., 2007; Wooldridge and Jennings, 1995; De la Prieta et al., 2019). One of the essential characteristics of agents is their ability to communicate with consumers and with system resources, thus forming a multi-agent system (MAS). MAS is a powerful tool for enhancing distributed systems performance. It is suitable for modeling and design of resources management strategies. So an optimization-based agent is suitable for allocating cloud resources due to its distributed and virtual nature (De la Prieta et al., 2019). Especially since resource allocation in the cloud suffers from many.

problems (Mishra et al., 2020). Reinforcement learning improvement methods are also necessary to solve prediction problems. Typically, they are used to create system models from data. These models should improve with more data than those observed during the training (Sutton and Barto, 2018). In this direction, Q-leaning is a reinforcement learning model that learns the value of an action in a particular state. The best advantage behind using the Q-learning method is its flexibility and ability to adapt to different cloud environment changes. It is an artificial intelligence field that allows agents to act intelligently in an environment to maximize the notion of cumulative reward. Therefore, reinforcement learning is a class of solution methods that can work well on the problem of dynamic resources allocation.

The allocation of resources in the cloud computing environment is surfacing many issues (Mishra et al., 2020; Hasan and Goraya, 2018; Hameed et al., 2016). Due to the variable workload, misallocation of resources can overload some virtual machines, while other VMs do not get the requested load. Knowing that the load balancing in the clouds may be between physical hosts or virtual machines. Correspondingly, fault tolerance requires the design of methods that allow a system to continue operating in a reduced fashion instead of completely failing when one of its components no longer functions properly. It is among the most imperative issues in the cloud to deliver reliable services (Hasan and Goraya, 2018). Likewise, the number of cloud resource consumers grows every year, making the energy consumption issue one of the hottest research trends in IT and the industry (Hameed et al., 2016; Kaur and Chana, 2015). Further, the emergence of big data centers will increase the electricity demand and thus impact on cloud operating budget. On the other hand, the virtual machine migration mechanism can facilitate hardware maintenance, load balancing, and disaster recovery. Additionally, the balancing method should distribute the dynamic workload equally among all nodes (hosts or virtual machines) (Mishra et al., 2020). Along with that, virtual machines must be reconfigured and modified dynamically with changing loads to make the best use of resources.

Traditional resource allocation methods do not consider the automation and distributed nature of the cloud, leading to high implementation complexity without achieving an optimal solution. To our knowledge, there is no practical way to provide energy consumption estimates before performing tasks. Moreover, executing many requests in the cloud will require fault tolerance to a higher degree of errors and failures than ordinary systems. Additionally, cloud computing needs to automatically tune the VMs state according to the increasing and decreasing workload, thereby balancing the system load. This is why optimizing resource allocation while simultaneously considering energy consumption, fault tolerance, and load balancing is the dream of cloud providers. In the literature, each of these problems is optimized separately by neglecting the impact of one on the others. This motivates us to think of a solution that takes into account these three issues while allocating resources at the same time. In other words, the critical contribution of this research is as follows:

- Present a mathematical formulation for fault tolerance, energy consumption, load balancing, and execution time.
- Propose a multi-agent model using Q-Learning to optimize resource allocation in the cloud considering fault tolerance, energy consumption, and load balancing. Also, we study the effect of IMARM on execution time.

The rest of this paper is organized as follows. Section 2 presents related works. Section 3 gives a mathematical formula for the problem under study. Section 4 explains our proposed solution. Section 5 evaluates its efficacy through simulation experiments. Section 6 discusses the advantages and limitations of our work. Finally, section 7 concludes the paper.

2. Related works

The resources of a cloud computing infrastructure are shared in real time between its customers. This creates serious problems for cloud service providers regarding fault tolerance, energy consumption, and load balancing. Indeed, these issues of resource allocation are widely discussed in the literature. However, every extant research has addressed only one or two problems, ignoring its impact on the other issues (Belgacem, 2022).

The work provided in (Tamilvizhi and Parvathavarthini, 2019) depicts an efficient cloud architecture that could tolerate failures while reducing energy consumption and workload overhead. However, the assessment experiments were not performed for a large number of consumers. In (Sharma et al., 2019), the authors have used a failure-aware VM consolidation mechanism (FCM) to save energy in a failure-prone cloud computing environment. In (Marahatta et al., 2019) they presented a solution that coordinates the optimization of resource utilization and energy consumption with a fault-tolerant mechanism. The proposed mechanism minimizes the task rejection ratio caused by machine failure and delay. However, the complementary features of tasks need to be investigated further. In addition, to reduce energy consumption and the resultant SLA violations, two workload consolidation techniques are presented in (Mustafa et al., 2018). However, this solution did not consider fault tolerance or load balancing. In (Adhikari and Amgoth, 2018), the authors combined server configuration and task-VM mapping to improve load balancing in an IaaS cloud. However, this work needs to consider other performance parameters such as VMs usage cost, deadline, etc.

On the other hand, Wanyuan et al. (Wang et al., 2016) introduced a decentralized multiagent (MA)-based VM allocation approach. The approach aims to allocate VMs to PMs while minimizing system energy costs. This method allows dispatching a cooperative agent to each PM to assist the PM in managing resources. However, this solution did not give consumers sufficient quality in using scalable VM resources. Another solution based on agent technology is proposed in (Bajo et al., 2016), called low-level resource distribution. It allows distributing computational resources throughout the entire cloud computing infrastructure, considering its complexity and associated computational costs. This monitoring and control of the system making possible to integrate the new features offered by virtualization. However, this work needs an extension to address new objectives and to include other infrastructure parameters. For efficient resource utilization and to minimize the cost of bandwidth, the researchers in (Gao et al., 2020) proposed a hierarchical multi-agent optimization (HMAO) algorithm. This algorithm outperforms other compared algorithms in solution quality, convergence time, and robustness as the number of tasks increases. Further, in (Singh et al., 2017), a new mechanism was proposed that deploys various intelligent agents to reduce the cost of virtual machines and resource allocation complexity. However, this technique needs more experimental results to prove its effectiveness.

In the paper (Gutierrez-Garcia and Ramirez-Nafarrate, 2015), different agents were used to balance workloads between heterogeneous servers. This mechanism is capable of balancing loads in a distributed and scalable manner. However, in the case of a large number of servers, the algorithm becomes complex. A flexible model that integrates an interaction between broker agents was presented in (Kemchi et al., 2018). The model considers several criteria in the processing of submitted customer requests. However, this solution still needs to be improved to be adapted to the allocation of resources. The authors in (Singh et al., 2015) presented an Autonomous Agent-Based Load Balancing Algorithm (AALB), which dynamically performs a proactive load calculation of VM according to a threshold value. However, the experiments did not show any efficient results. In (Jena et al., 2020), the authors hybridize between particle swarm optimization and an improved Q-learning algorithm to form a new approach named QMPSO. The proposed solution allows finding a suitable action from the set of possible VM actions according to the state of the VMs while achieving system load balancing.

To overcome the poor quality of service, the authors of (Jyoti and Shrimali, 2020) proposed a new approach that provides dynamic provisioning of resources based on load balancing and brokering of services. The paper used a deep reinforcement technique to predict environmental cloud activities and allocate resources. However, the article did not discuss the issue of fault tolerance. Another study presented in (Xu et al., 2020) has suggested a strategy based on machine learning for VM placement. It showed an accepted improvement in terms of load balance. In (Chinnathambi et al., 2019), the authors sought to optimize fault tolerance using a checkpoint mechanism. Its effectiveness was evaluated mainly on byzantine errors. The proposed model performs better than other algorithms and is suitable for real-time applications.

To minimize energy consumption, the authors of (Kurdi et al., 2018) proposed an algorithm named LACE. It is based on the simulation of behavioral phase changes in locusts. LACE allows the workload to be distributed among servers rather than centralized in a single component. The solution presented in (Kong et al., 2020) takes into account the end time and the earliest end time to achieve efficient scheduling with load balancing. However, this work did not consider the energy consumption in data centers and VM migration. In (Devaraj et al., 2020), the authors focus on load balancing. The presented algorithm integrates the advantages of two techniques to minimize the search space and respectively identify the improved response. However, this algorithm still needs improvements to be used for resource allocation.

In (Singhal and Singhal, 2021), the authors suggested an auction resource allocation model that promotes genuine providers with good feedback, discourages market saboteurs and promotes fairness in the system. However, this research did not address the energy consumption issue and has not assessed with different virtual machine setups. Thein et al. (Thein et al., 2020) proposed a solution based on reinforcement learning mechanism and fuzzy logic to achieve high-energy efficiency of the data center. This solution is not efficient for a large number of resources and is not applicable in real-time allocation. Liang et al. (Liang et al., 2019) proposed a model based on the semi-Markovian decision process and reinforcement learning for adaptive allocation of cloud resources. The evaluation results show that this approach still needs to be improved to allocate resources effectively. In (Praveenchandar and Tamilarasi, 2021), the authors used a prediction mechanism and a dynamic resource table update algorithm to minimize energy consumption. In (Pradhan and Bisoy, 2020), they gave a heuristic approach for cloud load balancing. Similarly, Karthiban et al. (Karthiban and Raj, 2020) proposed a resource allocation scheme based on the Deep Reinforcement Learning Model (DRLM). However, these works did not address the fault tolerance problem.

3. Mathematical formulation

The cloud provider receives and responds to customer requests through a browser. These requests reflect consumer resource requirements. Growing cloud consumers can lead to increased energy consumption due to, for example, cooling systems, fault tolerance mechanisms, etc. Implementation times must also be improved to ensure the quality of service. Additionally, the workload distribution across virtual machines should be balanced. Further, assigning and launching virtual machines during resource allocations can cause them to fail. Consequently, it is necessary to manage resources in this environment. The cloud computing environment studied in this paper appears in (Fig. 1). The main symbols used in this study are summarized in the Table 1.

3.1. Application model

We schedule a Bag of Task (BoT) application in this work. The BoT application is widely used in scientific and engineering disciplines and commercial organizations such as Facebook (Thai et al., 2018). A BoT application consists of a set of parallel tasks $T = \{T_i | 1 \le T_i \le m\}$, where each task is characterized by an identifier (id), and length (lg). Tasks are executed on set of $VM = \{VM_j | 1 \le VM_j \le n\}$, where each task is characterized by an identifier (id_{vm}), the speed of the CPU resource is represented by the parameter multi instruction per second ($mips$), and its bandwidth BW. The VMs are placed on a set of hosts $H = \{H_k | 1 \le H_k \le s\}$, where n, m, and s are the number of Tasks, VMs, and Hosts respectively.

3.2. Fault tolerance model

Implementing a fault-tolerant technique can be performed at different levels of cloud services (Hasan and Goraya, 2018). Some techniques are used to target specific types of failures, others detect and correct failures, etc. Briefly, faulttolerant (ϑ) is the ability of the system to control the state in order to deal with different interruptions (faults, errors, failure) before they occur. In this study, we are interested in preemptive fault tolerance. For that, we need to predict the failure through resource allocation operations, thereby a **checkpoint mechanism** is adopted (Fig. 2).

Checkpointing mechanisms allow the system to periodically save the task execution states. In case of any failure, the task is restarted from the last saved state rather than restarting the task from the beginning. Checkpoints are selected at regular intervals after a certain number of execution time units (quantum time Δ (Belgacem et al., 2020). During a period Δ, a checkpoint of length Γ is taken. This leads to the challenge of determining in which execution task situation the checkpoint is blocked or not. In other words, it is necessary to store checkpoint data during resource allocation processes using a proper architecture. For that, a slowdown factor Ω is defined to measure the time units lost in the event of a disrupting checkpoint jitter. Thus, the units of work wasted due to the disturbance of the checkpoint jitter are expressed as $(1 - \Omega\Gamma)\Gamma$ knowing that $0 \le \Omega \le 1$. $\Omega = 0$ corresponds to an entirely blocked checkpoint, while $\Omega = 1$ corresponds to a checkpoint overlapped with computations.

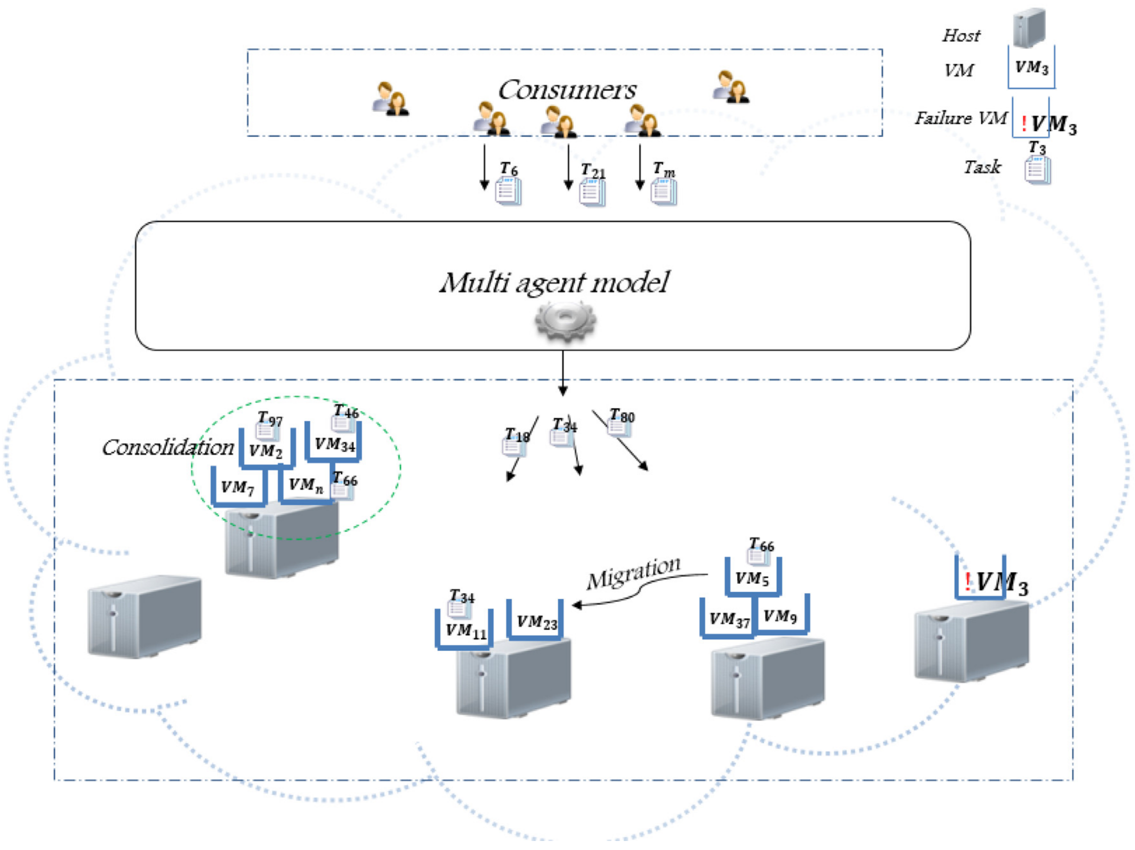


Fig. 1. Problem description.

Table 1
Symbols.

Symbols	Signification	Symbols	Signification
ϑ	Fault-tolerant	Δ	Quantum time
θ	Energy consumption	Γ	Checkpoint time length
φ	Execution time	$MTBF$	Mean Time Between Failures
σ	Load balancing	Ω	Slowdown factor
F	The number of failures	σ_B	Load-balancing threshold
v	Value of MTBF	Ψ	Objective function

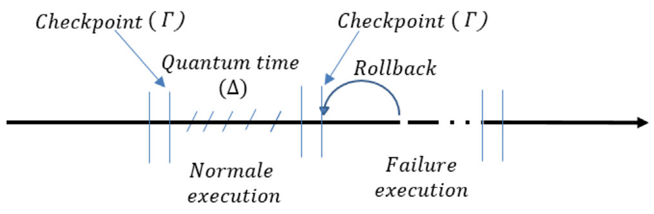


Fig. 2. Checkpoint mechanism steps.

The number of failures, F, is expected by $\frac{F}{v}$, knowing that v is the Mean Time Between Failures (MTBF) (Herault and Robert, 2015). Note that for n identical resources with MTBF = v_{ind}, the $= \frac{MTBF}{v_{ind}}$. In the event of a failure, a downtime of length Γ_D is happening, then recovery of length Γ_R. The task is re-executed from the last checkpoint before the failure (Rollback). It is noted that the shorter the period Δ, the less work to re-execute, but also the more overhead due to frequent checkpoints in a failure-free execution (Herault and Robert, 2015). So, the best trade-off is achieved according to the formula $\Delta = \sqrt{2\Gamma v} + \Gamma$ (Young, 1974). In the rest of the article, we assume that ϑ reflects the number of failures F.

3.3. Energy consumption model

In this study, we consider the energy consumption in different virtual machine states during resource allocation:

- Static energy (θ_s): This is the base energy consumption when operating the cloud system. More precisely, it corresponds to the energy consumed by VM when no tasks run inside it, which is the energy of the CPU idle (Belgacem et al., 2020). Each VM in the resource pool has energy profile information, such as minimum and maximum operating power consumption.
- Processing energy (θ_p): This is the energy consumption that corresponds to the case when there is workload performed on the VM, in addition to the static power (Herault and Robert, 2015).

$$\theta_p = \theta_s + \frac{\theta_s}{v} + (\Omega\Gamma + \frac{\Delta^2 - \Gamma^2 + \Omega\Gamma^2}{2\Delta}) \tag{1}$$

- Migration energy (θ_m): The migration power depends on VM_{cpu}(mips), network bandwidth (BW), and migration time (Γ_M). Therefore, if the average dissipated power is denoted θ_r, and the migration duration is $\Gamma_M = \frac{mips}{BW}$, θ_m is given by:

$$\theta_m = \theta_r * \Gamma_M \tag{2}$$

- Downtime energy (θ_d): This is the power consumed when one machine is down.

- Checkpoint energy (θ_c): This is the energy consumed in checkpointing.
- Failure energy (θ_f): This is the energy lost due to re-execution. According to (Herault and Robert, 2015); θ_f is calculated as follows:

$$\theta_f = \frac{\Delta^2 - 2\Gamma}{2\Delta} * \theta_p \tag{3}$$

From the above, the total VM energy consumption is computed as shown in equation (4). Hence, the total energy consumed by the cloud environment (θ_E) is the sum of all θ_{VM} (equation (4a)).

$$\theta_{VM} = \theta_c + \theta_p + \theta_m + \theta_d + \theta_f \tag{4}$$

$$\theta_E = \sum_{j=0}^{n} \theta_{VM_j} \tag{4a}$$

3.4. Load balancing model

In this research, we are interested in selecting an optimal host in a data center to place virtual machines on that host. To this end, VM migration is adopted to balance the cloud system. Selecting a suitable host is subject to some rules. The rules are defined according to the state of the system. The weight of VM_j (σ_j) is controlled periodically during a quantum of time (Δ). This way, if a virtual machine can handle several tasks, it is prevented from getting overloaded. Therefore, the controller will assign the task to the most powerful VM for each allocation. Equation (5) shows the VM weight expression, knowing that $\sum lg$ is the total amount of tasks (length) running on the VM_j, and $\sigma_B = \frac{\sum_{j=0}^{n}\sigma_j}{n}$ is load balancing factor of the system.

$$\sigma_j = \frac{\sum lg}{mips_j} \begin{cases} if & \sigma_j = 0 VM_j is shutdown \\ if & \sigma_j > \sigma_B VM_j is overloaded \\ if & \sigma_j < \sigma_B VM_j is loaded \\ if & \sigma_j = \sigma_B VM_j is balanced \end{cases} \tag{5}$$

In each period, the system maintains a record of the state of each virtual machine (busy, ideal, failure). If a task arrives for allocation, it is sent to the ideal virtual machine. In the case of VM failure, the task is migrated to another ready VM. The system energy consumption in the balanced state (θ_B) is calculated as shown in equation (6).

$$\theta_B = \sum_{j=0}^{n} \sigma_j * \theta_{p_j} \tag{6}$$

3.5. Execution time model

The first time considers in resource allocation is the checkpoint period without failure. It is the periodic checkpoint intervals during fault-free execution (φ_{bas}). During each Δ period, the system spends $\Omega\Gamma$ of time on checkpoints processes which makes the φ_{bas} calculated as follows:

$$\varphi_{bas} = \Delta - \Omega\Gamma \tag{7}$$

For each failure, the time lost is expressed according to the downtime Γ_D and recovery time Γ_R. During the Δ period, the probability that a failure happens while we are not checkpointing is assumed to be $\frac{\Delta - \Omega}{2}$. The likelihood that a loss occurs during checkpointing is $\Delta - \frac{\Omega}{2}$ (Hasan and Goraya, 2018). Thereby the failure time is expressed as shown in equation (8).

$$\varphi_{fail} = \Gamma_D + \Gamma_R + \frac{\Delta}{2} \tag{8}$$

Each task T_i is assigned to one VM_j, and executed at a period φ_{ij}.

$$\varphi_{ij} = \frac{lg_i}{mips_j} \tag{9}$$

Therefore, the total execution time of the task depends on φ_{bas}, *VM* settings, task length, and the time lost due to failures φ_{fail} (equation (10)).

$$\varphi_E = \sum_{i=0}^{m} \varphi_{ij} + \varphi_{bas} + \varphi_{fail} \tag{10}$$

3.6. Objective function

This research aims to minimize energy consumption and provide fault tolerance while maintaining cloud system load balancing, as shown in equation (11).

$$\Psi = Min(\theta_E) \tag{11}$$

S.t

$$\Delta = \Delta^* \tag{11a}$$

$$\sigma_j \leq \sigma_B \text{ for each } VM_j \tag{11b}$$

$$\forall j \in \{1, 2, \cdots, n\}, \sum_{j=0}^{n} VM_j \leq 1 \tag{11c}$$

The constraints mentioned express the following meanings:

- Constraint 11a means that we select an optimal checkpointing interval.
- Constraint 11b means that the load of each VM must be inferior or equal to the load balancing threshold of the cloud system.
- Constraint 11c means that each virtual machine can only run on one server.

4. Proposed Multi-Agent reinforcement model

The proposed Intelligent Multi-Agent Reinforcement Model (IMARM) is structured by different types of agents, as shown in Fig. 3. One set of agents is named *Sensing agents*= {*ECCA, FDA, LCA*}. These agents are a group of autonomous, interacting entities that collect information about the cloud environment using sensors. Also, there is a set of agents named *VM agents*= {VMA_1, VMA_2,…,VMA_m}. These agents react according to a report from a Sensing agent. The proposed system relies on a central organization. Therefore, the system is monitored by a central Resource Allocation Agent (*RAA*) that acts with a Consumer Agent (*CA*).

4.1. Sensing agents (Environment)

Autonomous Sensing agents interact with VM agents to provide data about the cloud environment and make the system intelligent. Sensor agents provide real-time tracking of consumer requirements and monitoring of VMs to improve the quality of service. These agents communicate with RAA by messages.

- Energy consumption controller agent (*ECCA*): This agent considers the five different states of energy consumption mentioned above. In other words, they detect different energy

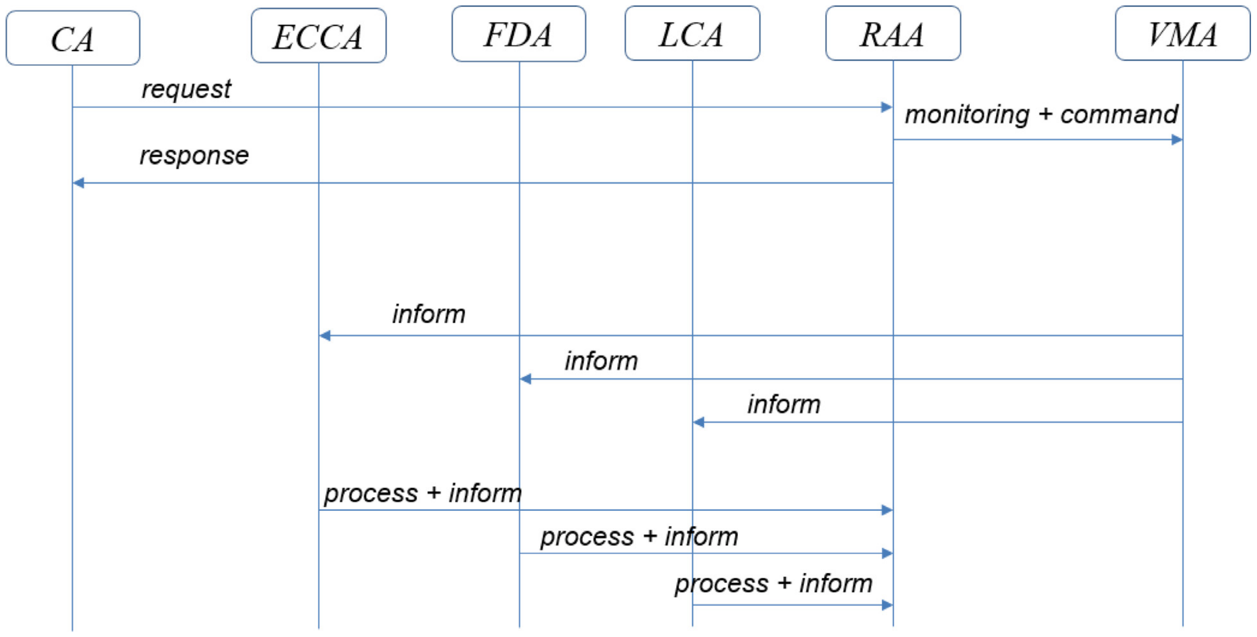


Fig. 3. Sequence diagram of proposed multi-agent system behavior.

consumption values of the environment and store these values in the registry.

- Fault tolerance detection agent (*FDA*): This agent collects failure, downtime, and recovery time at regular intervals.
- Load balancing agent (*LCA*): As mentioned before, virtual machines are distributed to various hosts while tasks are submitted to different virtual machines decentralized. *LCA* acts to collect precise state information about a load of virtual machines. As a result, it measures the VMs weight (σ) and load balancing factor of the system, thereby deducing the load balancing of energy consumption (θ_B).

This category of agents gives a global measure of the environment $E = (\varphi_E, \theta_B, \vartheta, \sigma_B)$, in each Δ time. The global measure can be calculated according to the following matrix:

$$E = \begin{pmatrix} \varphi_{1,1} & \theta_{1,2} & \vartheta_{1,3} & \sigma_{1,m} \\ \varphi_{2,1} & \theta_{2,2} & \vartheta_{2,2} & \sigma_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{n,1} & \theta_{n,2} & \vartheta_{n,m} & \sigma_{n,m} \end{pmatrix} \quad (12)$$

here, each row of the matrix presents the values sent by VMA to sensing agents and $\vartheta \in [0,1]$.

4.2. Service provider agents

The role of a VMA (Virtual Machine Agent) is to control the VMs, make them autonomous and react to various environmental changes. For this, the Q-learning method is employed to enhance the behavior of VMs. Q-Learning is a reinforcement learning method that allows for optimal allocation of resources according to environmental requirements and changes. VMA uses data acquired with the precedent category of agents to improve load balancing and energy consumption with fault tolerance. In the considered multi-agent system, each VM is regarded as an agent to determine its optimal computation offloading strategy by interacting with the environment. Sensing agents observe the local environment state of VM $s_i \in S$, and then RAA decides the correct action for each VM. For the Q-learning method, a stochastic formulation is given. The formulation is a generalization of a Markov decision process. It includes the tuples < *Cr,S,A,Pr,r* >.

- *Cr* is the set of *n* VMAs;
- *S* is the set of possible states of each VMA;
- *A* represents set of possible actions for each VM;
- *Pr* is the probability function of transitioning to state *S*, (Pr: $S \times A \in [0,1]$);
- *r* is the reward function for each transition.

Based on the definition above, the main items in the tuple will be as follows:

VMA state-space: At a time instant, the position and/or orientation of the VMA in the environment are: ready, off, failure, overload, balance.

VMA action-space: Each VMA takes action based on energy consumption, fault tolerance, and load balancing. We designate the actions downtime, migration, rollback, and switching (from ready to off and vice versa). The method will need a reward matrix and will output a quantity matrix.

Reward: At each quantum time, VMA receives an indication of the current state of the environment from the RAA agent. After that, VMA chooses an action to change the state of the environment. In each state transition, VMA receives a reward (Fig. 4).

Reward function:. This research aims to minimize the number of failures and energy consumption while maintaining load balancing. Thereby, the reward function should be defined as:

$$r = \begin{cases} \frac{1}{\omega_1 \theta_j + \omega_2 \varphi_j} & if \vartheta_j = 0, (random \omega_1, \omega_2 \in [0,1]) \\ 1 & if \vartheta_j = 1 (failure) \\ -1 & impossibletransition \\ 0 & keepthesamestate \\ 1 & if \sigma_j > \sigma_B and \vartheta_j = 0 \end{cases} \quad (13)$$

The Q-learning method aims to learn a policy that maximizes the total reward. The Q-learning process will have two main stages:

Q-table creation stage:. First, the Q-learning table (matrix) is built, where the number of columns and rows correspond to the

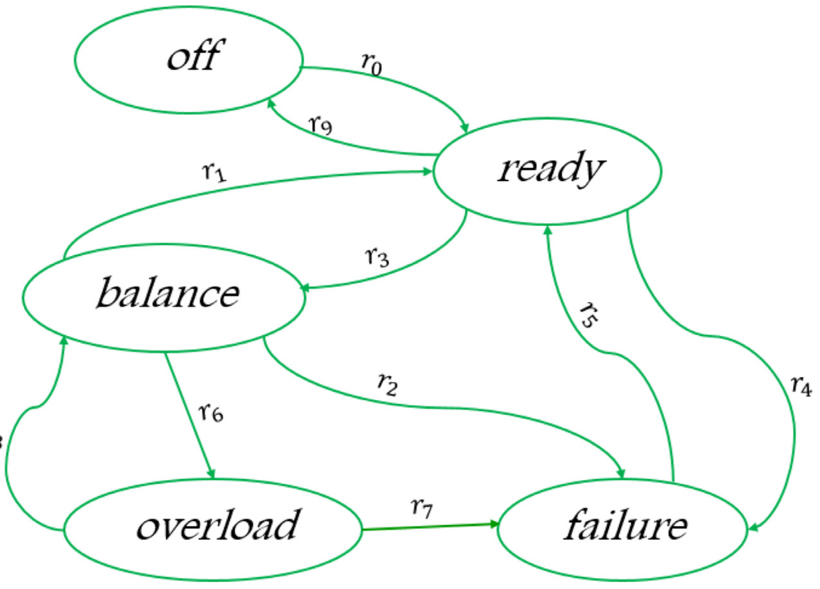


Fig. 4. The different changes of VMs state.

number of actions and states. Each state-action combination represents the quality value of an action taken from that state. The Q-value is initialized to zero; it will be updated and stored after each training. The Q-learning table is considered a reference to select the best action according to the Q-value.

Q-learning and making updates stage:. The VMA agent interacts with the cloud environment using one of two ways. The first way is exploitation, which uses the Q-table as a reference to select the action based on the maximum value of all possible actions for a given state. The second way is called exploring, which takes action to act randomly. This lets the agent explore and discover new states that otherwise may not be selected during the exploitation process. In addition, it is possible to balance exploration/exploitation by determining the value of exploring vs. exploiting. Briefly, Q-learning lets the VMA agent use the environment rewards to learn the best action decided for a given state. It is necessary to consult a Q-table to choose a sequence of actions that should maximize the reward. More precisely, the transition to state (s_i + 1), with an action (a_i + 1), is accompanied by receiving a reward (r_i + 1).

The Q-values are updated using the rule given by Sutton and Barto (1998) (Sutton and Barto, 2011) and Kaelbling et al. (Kaelbling et al., 1996) (Kaelbling et al., 1996):

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max'_a Q(s',a') - Q(s,a)) \tag{14}$$

here, *s'* is the state attained from state *s* when performing action a_i. In each iteration, the Q-values are adjusted according to equation (14). Here, two operators are used. The first operator is the learning rate (α), which gradually decreases for convergence ($0 < \alpha \leq 1$). The second operator is (γ), which is the discount factor ($0 \leq \gamma \leq 1$) (Sutton and Barto, 2011; Even-Dar et al., 2003). Moreover, the current action is updated with the possible future reward, and the maximum of the future reward is used and applied to the reward of the current state. This permits the VMA agent to select the highest return action at any given state (Fig. 5). Thus, multiple updates of every state-action pair lead Q-learning to achieve the optimal state-action decisions. The pseudocode of the Q-learning method is presented in algorithm 2, and algorithm 3 gives the source code of the proposed solution (IMARM).

4.3. Resource management agents

The cloud system is monitored by a central Resource Allocation Agent (*RAA*) that interacts with a Consumer Agent (*CA*).

- Consumer agent (*CA*): The CA agent represents the cloud consumer. It negotiates with the *RAA* to reach a service level agreement (SLA) about resources. The SLA should be taken into account in the resource allocation process. The result of the negotiation is a set of tasks directed to be executed on VMs.
- Resource allocation agent (*RAA*): This agent contains the resource allocation policy, responsible for mapping the tasks to VMs. In this research, we use our approach suggested in previous work (*MOSOS*) to assign tasks to virtual machines (Belgacem et al., 2020). *RAA* acts as a broker and receives inquiries from consumers to provide them with resources. It receives consumer requests and analyzes them to extract the resource requirements. Therefore, *RAA* allows allocating appropriate resources to every corresponding request. Consequently, it is an intermediary between sensing agents and service provider agents to facilitate resource allocation (Fig. 6).

Algorithm 1: States changing

```
Input: Q-table (si), Information given by sensor agents
  (VMinf), r, Action set (a).
Output: Best VMs adjusting state (a).
1:For Each (VM: n).
2:   For Each ($s_i$, $VM_{inf}$).
3:   Calculate the reward r and the state using the equation
     (13).
4:       IF(r < r').
5:           si ← (s'i).
6:        $a$ ← ($a'$).
7:        End If.
8:    End For.
9: End For.
```

```
Algorithm 2: Update Q-table:
Input: (s, a) the current state and action, s' is the next state, r
  is the immediate reward received.
Output: Best Q-table values.
1: Call Function1.
2: Q(s, a) ← Q(s, a) + α(r + γmaxa'Q(s', a') − Q(s, a)).
3: Construct Q-matrix for state-action pairs.
4:    For Each State.
5:       Choose an action a.
6:          Observe next state, s', select a action.
7:             Switch (Case).
8:                Case1: ($\vartheta_j$ = 0).
9:                   r = $\frac{1}{\theta_j+\omega_2\theta_j}$
10:                  Case2: (ϑj = 1) (failure) or σj > σB and ϑj = 0 σB
  and ϑj = 0).
11:                  $r \leftarrow 1$.
12:               Case3: (impossible transition).
13:            $r \leftarrow -1$.
14:               Case4: (a = a').
15:            $r \leftarrow 0$.
16:         End Switch.
17:   $Q - table \leftarrow (s, a')$ (case where r is minimun).
18: End For.
```

```
Function 1: Q-table creation stage
Input: Set of Action, set of State.
Output: Q-values table.
1:    For Each (state).
2:       Choose randomly action.
3:       $Q - table \leftarrow (s, a')$.
4:    End For.
```

```
Algorithm 3: IMARM
Input: Set VMs, set T.
Output: Best mapping T-VM,
Best Cloud environment management.
1: While t ← ∞.
2: For Each (t = Δ).
3:          RAA collects information from sensor agents.
4:       Call Algorithm 1.
5:       Call Algorithm 2.
6:       Call MOSOS Algorithm.
7:    End For.
8: End While.
```

5. Experimental evaluation

This section presents our experimental platform, evaluation, and results.

5.1. Implementation

The proposed solution is implemented with the Cloudsim 3.0.3 toolkit based on JAVA and JADE (Java Agent Development Environment). JADE is a software platform that provides basic middleware-layer functionalities. The functionalities are independent of the specific application, and they simplify the realization of distributed applications, which exploit the software agent abstraction (Wooldridge and Jennings, 1995). The experiments are performed on an Intel (R) Core (TM) i5 3320 M Processor 2.6 GHz, equipped with 4 GB RAM, Windows 7 platform, using the Eclipse IDE Luna release 4.4.0.

5.2. Performance metrics

Three series of experiments are conducted to evaluate the performance of our proposed intelligent multiagent model. The IMARM is compared with a failure-aware VM consolidation mechanism (FCM), the QMPSO and DRLM algorithms presented in the works (Sharma et al., 2019; Jena et al., 2020; Karthiban and Raj, 2020), respectively. FCM is selected since it can significantly improve energy consumption by considering the failure characteristics of physical resources and using the VM consolidation mechanism. QMPSO allows efficient execution time during load balancing. While the DRLM algorithm permits highlighting the effectiveness of combining MAS and Q-learning reinforcement methods. Therefore, our IMARM is assessed according to the following scenarios.

Fault tolerance evaluation scenario. This experiment was designed to assess and compare the effectiveness of the proposed multi-agent model for fault tolerance in heterogeneous cloud data centers. Therefore, the Grid5000 fault data set is used. The data set was collected for 1.5 years between 2005 and 2006, and it has been downloaded from Failure Trace Archive (FTA). The dataset includes traces containing information about failures and physical machine configurations for approximately 1300 nodes (Kondo et al., 2010). The mean time between failures (MTBF) and the meantime to return (MTTR) for each node in each cluster is determined based on the failure information given in the traces.

In this scenario, the cumulative distribution functions (CDF) of time between failures are evaluated. Another important forecasting technique that many organizations widely use is the value of the smoothing constant (Gelper et al., 2010). Therefore, we performed a statistical analysis of the failure accuracy using different smoothing constant values for failure prediction. The reason for choosing the average-based prediction method is the inconsistency and non-stationarity of the available data collected from the Failure Trace Archive (Kondo et al., 2010). For each constant smoothing value, a set of forecasts is generated. Then these forecasts are compared with the actual observations of the time series (A time series is a sequence of time intervals of observations). Finally, the value that offers the smallest sum of squared forecast errors is selected. This study was conducted for seven values of the smoothing constant, where the maximum checkpointing overhead was 20 s.

Energy consumption experiments scenario. This experiment instantiates a scenario taken from projections for the Exascale platforms (Ferreira et al., 2011; Dongarra et al., 2009; Herault and Robert, 2015). Therefore, real values for energy consumption and fault tolerance parameters are chosen. The energy consumption of a VM is capped to $\theta_s = 20mW$, and there are 10^6 VMs. This scenario assumes that the platform energy is about 50% of this power, hence $\theta_s = 10$. A key parameter for this experiment is the energy environment θ_E defined in equation (4a). According to (Zheng et al., 2012) and with one fault per day, v_{ind} equal to 45,208 365 ≈ 125 years. We take $\theta_c = \theta_d = 10$ min, $\Delta = 1$ min, and $\Omega = 1/2$. The MTBF (v) is varied from $v = 300$ min (5 h) down to $v = 30$ min.

Load balancing experiments scenario. The experiment is designed to assess the effectiveness of the proposed IMARM in terms of load balancing. Therefore, we investigated its behavior during VM migration. More precisely, the performance of IMARM load balanc-

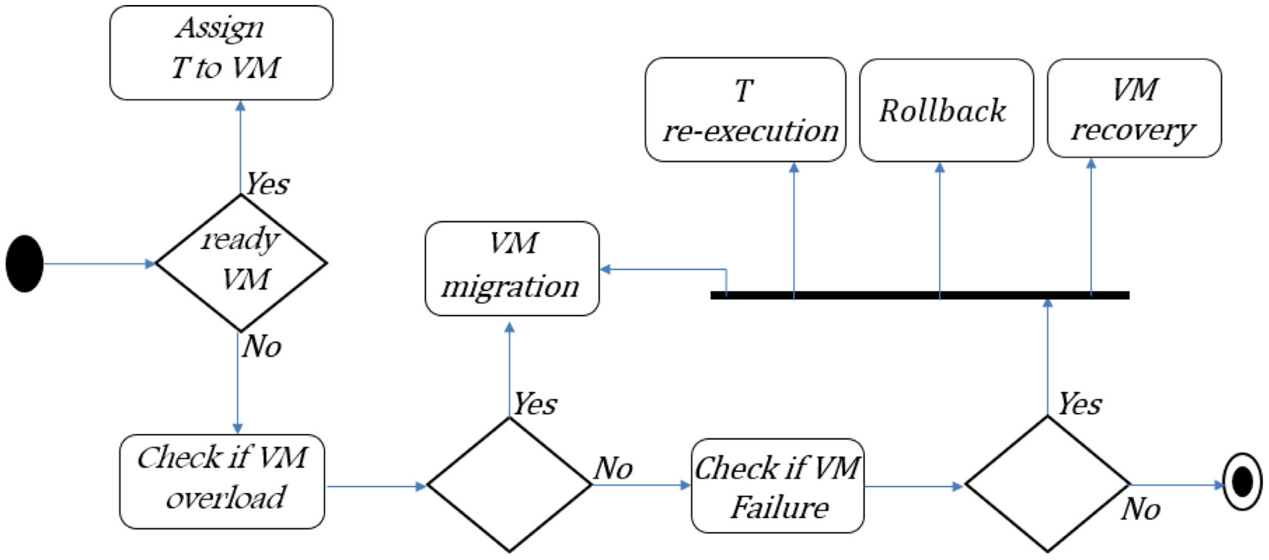


Fig. 5. The transition states in the cloud system.

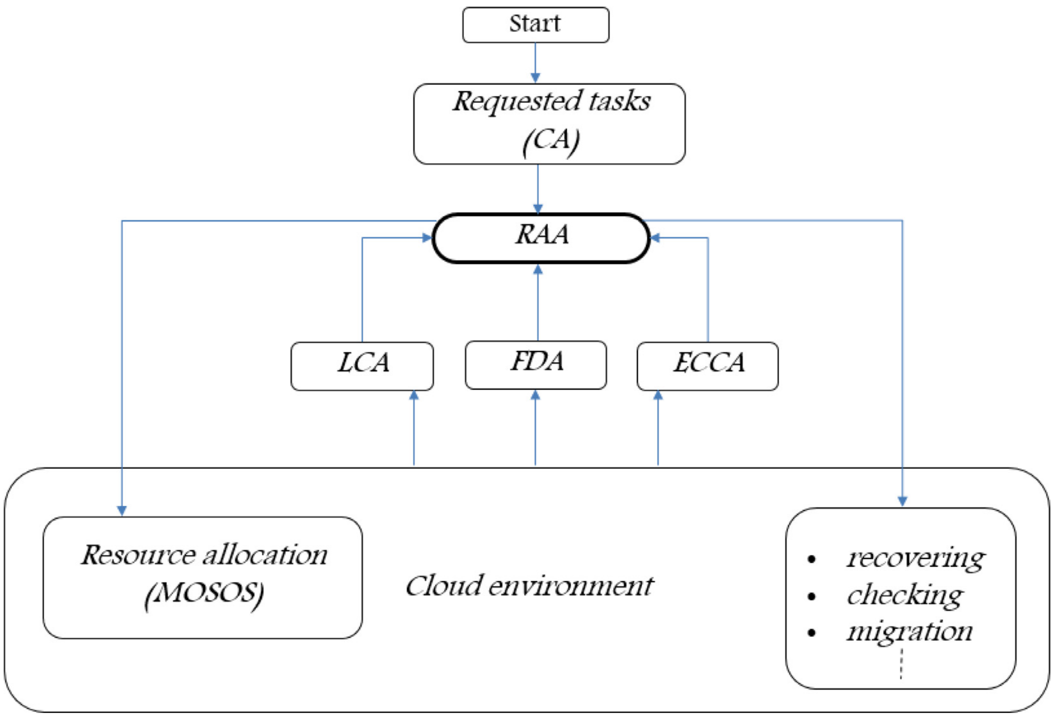


Fig. 6. Diagram of the IMARM solution.

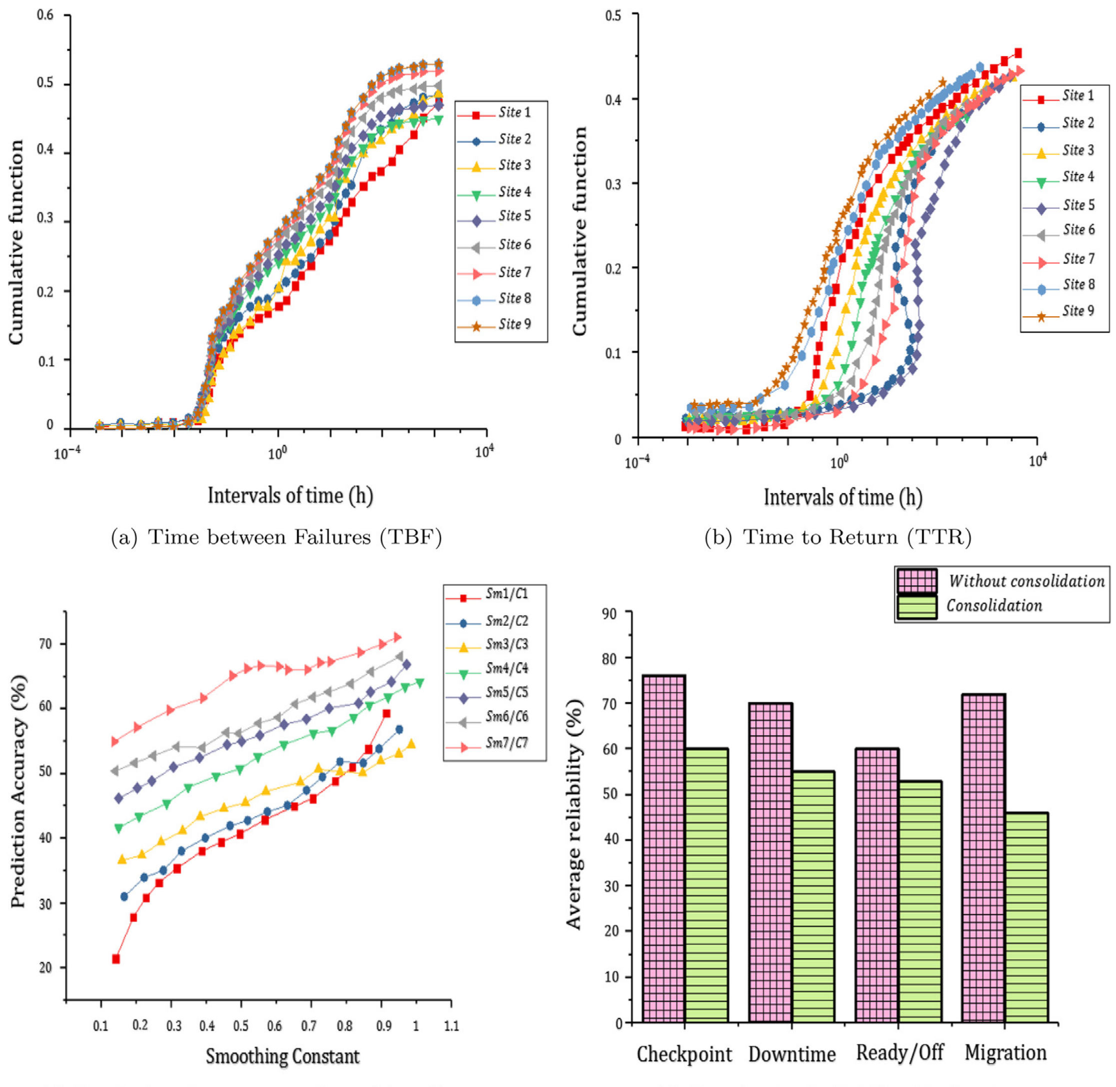
ing is analyzed, and the degree of imbalance is calculated according to the number of virtual machine migrations during the balancing process. The number of tasks is fixed to 1000, and VMs vary from 500 to 2500 in 50 s. We keep the same tasks and VMs configuration as in work (Jena et al., 2020).

5.3. Results and discussions

The performance evaluation of the proposed multi-agent solution was carried out in terms of fault tolerance, energy consumption, load balancing, and execution time. The results given are simulated 20 times then the average value is taken.

5.3.1. Fault tolerance evaluation

Fig. 7(a) shows the variation of the cumulative distribution function in different times between failures. The vertical axis is between 0 and 0.5, which gives the probability values of the CDF. The same behavior can be observed between the occurrence of failed and recovery events in (Fig. 7(b)). Both graphs fit a weibull and normal distribution with a slight deviation.



(a) Time between Failures (TBF) (b) Time to Return (TTR)

(c) Prediction Accuracy vs Smoothing Constant (d) Results for Reliability Evaluation

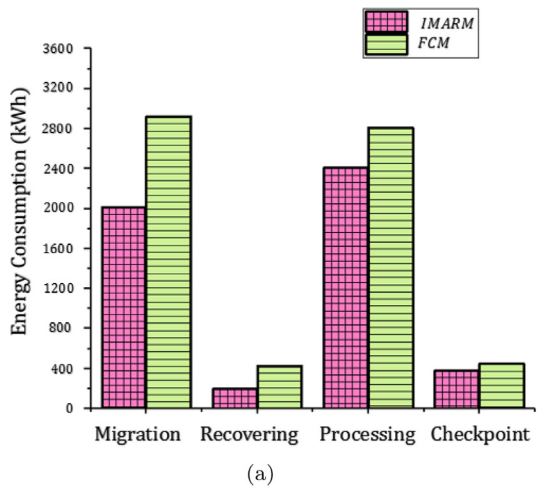
Fig. 7. Fault tolerance evaluation results.

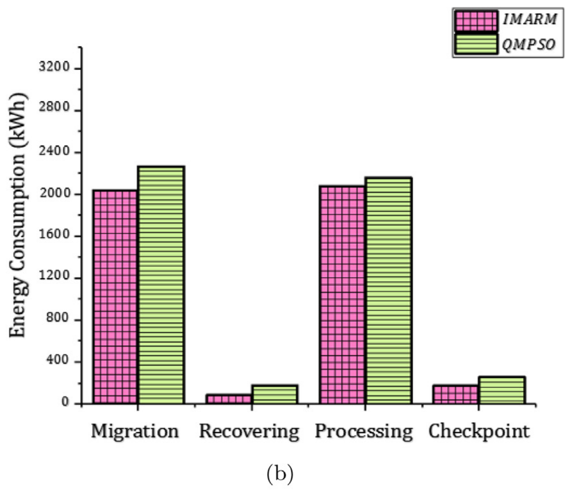
The smoothing constant indicates the level at which past observations impact the forecast. From examining Fig. 7(c), we observe that the value of the smoothing constant increases between 0.2 and 1, at the same time as the accuracy of the failure prediction also increases. Consequently, the smaller the values, the smoother the pattern in the smoothed values, thereby achieving better prediction results in the short term. Likewise, when using the moving average prediction method, IMARM showed better failure prediction accuracy. Because we generate a corresponding failure expectancy-value for each failure event value in failure traces (interpolation expectation). Generally, the prediction accuracy obtained by IMARM using a smoothing constant of 0.9 is significant.

Fig. 7(d) shows the average reliability behavior under different states with and without consolidation. Generally, it is noticed that the system with consolidation has higher reliability than the scenarios without consolidation. This can be due to the reduction in the occurrence of failures. However, when the overhead of virtual machines increased (increase in incoming tasks), the system reliability decreased. The results show that VM migration was mainly used for fault-tolerance reactions and system load balancing. In a failure, the system recovers the failed virtual machines, and the tasks are re-executed, which increases the system reliability.

5.3.2. Energy consumption evaluation

Fig. 8 shows the different types of average energy consumption of the system. Our proposed solution generally shows minimal energy consumption compared to the FCM, QMPSO, and DRLM algorithms. Our solution allows for a reduction of 24.25% of energy. Obviously, the system consumes maximum power in the case of processing due to a large number of incoming tasks. The system consumes minimal energy in recovery due to the low number of failures, reflecting IMARM fault tolerance efficiency. At the checkpoint, the system spends little energy. The energy consumed during VM migration is due to VM overload. The consolidation technique helps to reduce this type of energy consumed. The LCA and ECCA agents periodically inform the RAA of the load and energy consumption of the virtual machines, which leads to select-





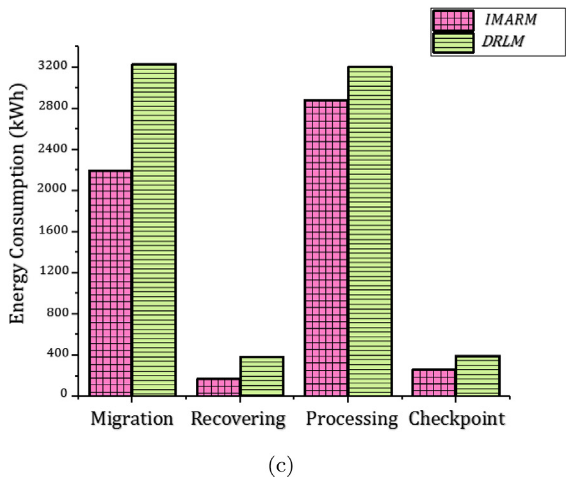


Fig. 8. Energy consumption.

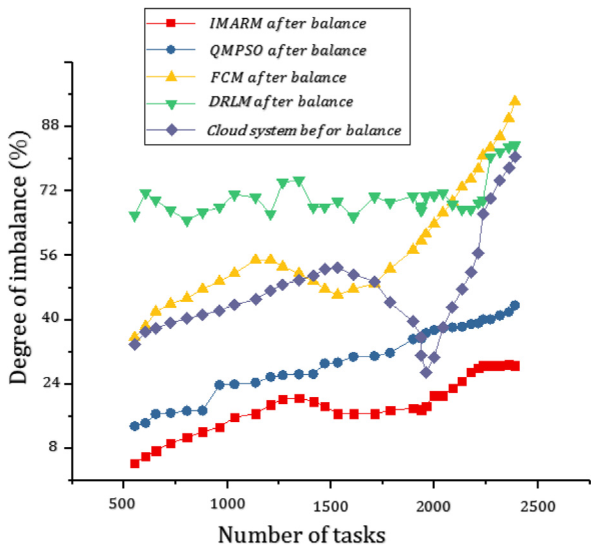


Fig. 9. Number of tasks with degree of imbalance before and after load balance.

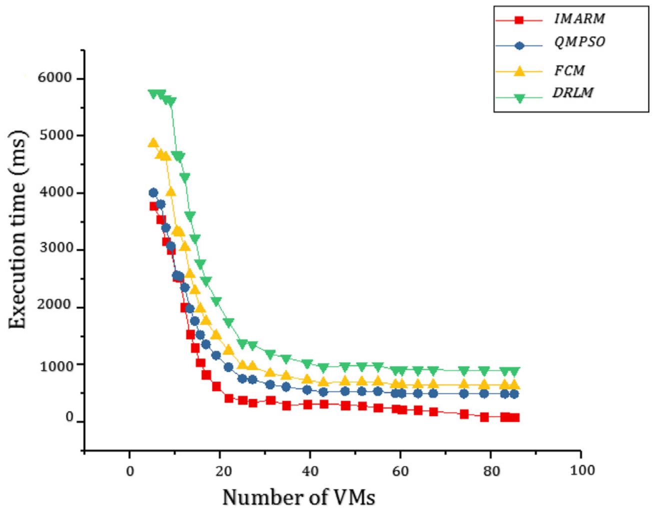


Fig. 10. Execution time results according to different numbers of virtual machines.

ing a set of suitable ready virtual machines in each allocation and shutdown others. Therefore, the overload and energy consumption of virtual machines are reduced.

5.3.3. Load balancing evaluation

The system load before and after balancing has been evaluated and is shown in Fig. 9. The degree of imbalance is calculated using equation (15). It is the number of VMs that have a load greater or

Table 2
Statistical analysis.

	SSDfMSF-statistic			P-value	F-critical
		Energy consumption			
Between groups	8170894.173	2	4085447.0863.631	0.081	3.16
Within groups	88508850.29	57	1552786.847		
Total	96679744.47	59			
		Execution time			
Between groups	235255.870	2	117627.9350.315	0.706	3.16
Within groups	19111018.19	57	352628.924		
Total	19346274.06	59			
		Load balancing			
Between groups	68.359	2	34.1800.052	0.950	3.16
Within groups	36720.559	57	660.010		
Total	37688.918	59			
		Comparison of statistical results			
	Energy consumption	Execution time	Load balancing		
IMARM	±0.065	±0.737	±0.945		
FCM	±0.765	0.999	±0.986		
QMPSO	±0.483 ± 1.02		±0.955		

less than the balance factor (σ_B) in the system. The imbalance degree is measured before and after the cloud system balancing, knowing that VMs are migrated due to overload. The figure illustrates that the degree of imbalance is the lowest after applying the IMARM algorithm compared to other algorithms. This is due to the regular task distribution that keeps the workload balanced on the virtual machines. Therefore, the proposed solution helps to maintain the balance of the cloud system when allocating resources.

$$Imbalance\ VM\ degree = \frac{Number\ of\ Imbalanced\ VMs}{n} * 100 \quad (15)$$

5.3.4. Execution time evaluation

Fig. 10 shows the execution time of tasks using different numbers of VMs. The execution time is calculated using equation (10). Initially, the standard deviation value is the same for IMARM and QMPSO. Then, gradually it decreases. However, when the number of VMs equals 30, it declines, and the degree of the standard deviation of QMPSO is greater than IMARM. This is due to the proposed sensor agent, which provides information regarding optimal VMs, quickly permits the selection of a set of high-performance VMs. In addition, as the number of VMs increases, the availability of optimal resources that can perform tasks in a short time increases. The same behavior is observed for FCM. Deviation starts when the number of VMs equals 36 because the number of VMs, in the beginning, was not very important compared to the upcoming tasks. After that, the number became sufficient, which justified the quasi stability of the curves. The same explanation we can say about the DRLM algorithm. Precisely, the IMARM curve falls below other curves, reflecting the minimum execution time compared to the others. Furthermore, using MOSOS that gives a better result in makespan has impacted the execution time.

5.4. Statistical analysis

Statistical analysis is performed to determine the accuracy of the classification by applying an ANOVA (Analyse of Variance) test to the experimental results. The results obtained from the test are presented in Table 2. The results show the experiences based on energy consumption, execution time, and load balancing. Here, SS is the sum of the square, Df is the degree of freedom, and MS is the average of the square. This statistical analysis is performed by obtaining the average test result from the data of different groups to determine whether the proposed algorithm has a statistical difference compared to other algorithms. This statistical test includes two main rules: the null hypothesis (H_0) and the alternative hypothesis (H_1). It is defined as, $H_{0:}\ z_0 = z_1 = \ldots = z_k$, $H_{1:}$ not all means are equal, and z_i is the mean of i^{th} level of the factor.

The ANOVA test examines the variance between the data groups and measures the F-statistical value and the P-probability value; to decide whether there is a significant difference between the groups. During the statistical test, if the F-statistic is less than the F-critical, then the test rejects the null hypothesis, and this condition indicates that the mean value of each group is not the same. If the F-statistic is more significant than F-critical, the null hypothesis is rejected, and the test accepts all the alternative hypotheses.

In this analysis, we perform the ANOVA test for each group of different values of the studied metrics, considering that alpha is 0.05. From the Table 2, the F-statistic > F-critical; so, null hypotheses are rejected. This establishes that the probability value of the F-statistic (P-value) < (0.05). The comparison between statistical results shows that the mean values of compared algorithms are not equal in terms of energy consumption and execution time. This means that the performance of IMARM is better than that of other algorithms (Miller, 1997). However, it shows a behavior close to the other compared algorithms in load balancing.

6. Discussion

To our knowledge, this paper is important, and it is a considerably extended version of the work reported in (Belgacem et al., 2020). The idea used in this work is based on agent and learning techniques to improve fault tolerance, energy consumption, and load balancing. The proposed model is intelligent since it automatically changes the VMs according to the state of the environment. The desired advantage when using the proposed solution is directly proportional to the creation and updating of the Q-table. More precisely, Q-learning is dedicated to training *RAA* and *VMA* agents to operate in a cloud environment. Our proposed solution is an excellent idea that combines agents and learning methods and completes the previous

MOSOS algorithm, using VMs migration and checkpoints techniques. It is an overall solution for the main resource allocation problems knowing that MOSOS improves the makespan and VMs cost usage.

7. Conclusion

This paper presents an Intelligent Multi-Agent Reinforcement Model (IMARM) for optimizing cloud resource allocation. It addresses the issue of resource allocation from several aspects, making it a comprehensive solution for cloud service providers. The proposed model provides fault tolerance and achieves load balancing using checkpointing and VM migration mechanisms. To determine the system reaction (recovery, downtime, migration, etc.), sensor agents periodically report system failure status, energy consumption, and workload. Correspondly, Q-Learning allows indicating the action to be performed in each system state. Knowing that the main parameters used to optimize the system are the weight of the virtual machines, the total energy consumption, and the quantum time. It is a compelling new strategy for finding the best response to changing cloud environments, with the ability to decide whether a task or virtual machine should be migrated, restored, shutdown, etc. To demonstrate the importance of IMARM, we evaluated it using FCM, QMPSO, and DRLM. The results and experimental analysis confirmed that our proposed algorithm outperforms others algorithms. It showed a good impact on the execution time by allowing a minimum execution time while using resources efficiently. Moreover, statistical analysis proves that IMARM has lower energy consumption and acceptable load balancing than its competitors. In the future, we plan to implement the IMARM solution on a real cloud computing platform.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Velte, A.T., Velte, T.J., Elsenpeter, R.C., Elsenpeter, R.C., 2010. Cloud Computing: A Practical Approach. McGraw-Hill New York.
Buyya, R., Broberg, J., Goscinski, A.M., 2010. Cloud Computing: Principles and Paradigms, Vol. 87. John Wiley & Sons.
A. Sunyaev, Cloud computing, in: Internet computing, Springer, 2020, pp. 195–236.
Tuli, S., Tuli, S., Tuli, R., Gill, S.S., 2020. Predicting the growth and trend of covid-19 pandemic using machine learning and cloud computing. Internet Things 11, 100222.
Dikaiakos, M.D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A., 2009. Cloud computing: Distributed internet computing for it and scientific research. IEEE Internet Comput. 13 (5), 10–13.
Belgacem, A., Beghdad-Bey, K., Nacer, H., 2020. Dynamic resource allocation method based on symbiotic organism search algorithm in cloud computing. IEEE Trans. Cloud Comput.
Belgacem, A., Beghdad-Bey, K., 2021. Multi-objective workflow scheduling in cloud computing: trade-off between makespan and cost. Cluster Comput., 1–17
Belgacem, A., Beghdad-Bey, K., Nacer, H., 2018. Task scheduling optimization in cloud based on electromagnetism metaheuristic algorithm. In: 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS), pp. 1–7.
Bellifemine, F., Caire, G., Greenwood, D., 2007. Developing Multi-Agent Systems with JADE. John Wiley & Sons.
Wooldridge, M.J., Jennings, N.R., 1995. Intelligent agents: Theory and practice. Knowl. Eng. Rev. 10 (2), 115–152.
De la Prieta, F., Rodríguez-González, S., Chamoso, P., Corchado, J.M., Bajo, J., 2019. Survey of agent-based cloud computing applications. Fut. Gen. Comput. Syst. 100, 223–236.
Mishra, S.K., Sahoo, B., Parida, P.P., 2020. Load balancing in cloud computing: A big picture. J. King Saud Univ.-Comput. Inform. Sci. 32 (2), 149–158.
Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.
Hasan, M., Goraya, M.S., 2018. Fault tolerance in cloud computing environment: A systematic survey. Comput. Ind. 99, 156–172.
Hameed, A., Khoshkbarforoushha, A., Ranjan, R., Jayaraman, P.P., Kolodziej, J., Balaji, P., Zeadally, S., Malluhi, Q.M., Tziritas, N., Vishnu, A., Khan, S.U., Zomaya, A., 2016. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. Computing 98 (7), 751–774.
Kaur, T., Chana, I., 2015. Energy efficiency techniques in cloud computing: A survey and taxonomy. ACM Comput. Surveys (CSUR) 48 (2), 1–46.
Belgacem, A., 2022. Dynamic resource allocation in cloud computing: analysis and taxonomies. Computing, 1–30.
Tamilvizhi, T., Parvathavarthini, B., 2019. A novel method for adaptive fault tolerance during load balancing in cloud computing. Cluster Comput. 22 (5), 10425–10438.
Sharma, Y., Si, W., Sun, D., Javadi, B., 2019. Failure-aware energy-efficient vm consolidation in cloud computing systems. Fut. Gener. Comput. Syst. 94, 620–633.
Marahatta, A., Wang, Y., Zhang, F., Sangaiah, A.K., Tyagi, S.K.S., Liu, Z., 2019. Energy-aware fault-tolerant dynamic task scheduling scheme for virtualized cloud data centers. Mobile Networ. Appl. 24 (3), 1063–1077.
Mustafa, S., Bilal, K., Malik, S.U.R., Madani, S.A., 2018. Sla-aware energy efficient resource management for cloud environments. IEEE Access 6, 15004–15020.
Adhikari, M., Amgoth, T., 2018. Heuristic-based load-balancing algorithm for iaas cloud. Fut. Gen. Comput. Syst. 81, 156–165.
Wang, W., Jiang, Y., Wu, W., 2016. Multiagent-based resource allocation for energy minimization in cloud computing systems. IEEE Trans. Syst. Man. Cybern. Syst. 47 (2), 205–220.
Bajo, J., De, F., la, Prieta, Corchado, J.M., S., Rodŕıguez,, 2016. A low-level resource allocation in an agent-based cloud computing platform. Appl. Soft Comput. 48, 716–728.
Gao, X., Liu, R., Kaushik, A., 2020. Hierarchical multi-agent optimization for resource allocation in cloud computing. IEEE Trans. Parallel Distrib. Syst. 32 (3), 692–707.
Singh, A., Juneja, D., Malhotra, M., 2017. A novel agent based autonomous and service composition framework for cost optimization of resource provisioning in cloud computing. J. King Saud Univ.-Comput. Inform. Sci. 29 (1), 19–28.
Gutierrez-Garcia, J.O., Ramirez-Nafarrate, A., 2015. Agent-based load balancing in cloud data centers. Cluster Comput. 18 (3), 1041–1062.
Kemchi, S., Zitouni, A., Djoudi, M., 2018. Amace: agent based multi-criterions adaptation in cloud environment. Human-centric Comput. Inform. Sci. 8 (1), 1–28.
Singh, A., Juneja, D., Malhotra, M., 2015. Autonomous agent based load balancing algorithm in cloud computing. Proc. Comput. Sci. 45, 832–841.
Jena, U.K., Das, P.K., Kabat, M.R., 2020. Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. J. King Saud Univ. – Comput. Inform. Sci.
Jyoti, A., Shrimali, M., 2020. Dynamic provisioning of resources based on load balancing and service broker policy in cloud computing. Cluster Comput. 23 (1), 377–395.
Xu, X., Fu, S., Li, W., Dai, F., Gao, H., Chang, V., 2020. Multi-objective data placement for workflow management in cloud infrastructure using nsga-ii. IEEE Trans. Emerg. Top. Comput. Intell. 4 (5), 605–615.
Chinnathambi, S., Santhanam, A., Rajarathinam, J., Senthilkumar, M., 2019. Scheduling and checkpointing optimization algorithm for byzantine fault tolerance in cloud clusters. Cluster Comput. 22 (6), 14637–14650.
Kurdi, H.A., Alismail, S.M., Hassan, M.M., 2018. Lace: a locust-inspired scheduling algorithm to reduce energy consumption in cloud datacenters. IEEE Access 6, 35435–35448.
Kong, L., Mapetu, J.P.B., Chen, Z., 2020. Heuristic load balancing based zero imbalance mechanism in cloud computing. J. Grid Comput. 18 (1), 123–148.
Devaraj, A.F.S., Elhoseny, M., Dhanasekaran, S., Lydia, E.L., Shankar, K., 2020. Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments. J. Parallel Distrib. Comput. 142, 36–45.
Singhal, R., Singhal, A., 2021. A feedback-based combinatorial fair economical double auction resource allocation model for cloud computing. Fut. Gener. Comput. Syst. 115, 780–797.
Thein, T., Myo, M.M., Parvin, S., Gawanmeh, A., 2020. Reinforcement learning based methodology for energyefficient resource allocation in cloud data centers. J. King Saud Univ.-Comput. Inform. Sci. 32 (10), 1127–1139.
Liang, H., Zhang, X., Zhang, J., Li, Q., Zhou, S., Zhao, L., 2019. A novel adaptive resource allocation model based on smdp and reinforcement learning algorithm in vehicular cloud system. IEEE Trans. Veh. Technol. 68 (10), 10018–10029.
Praveenchandar, J., Tamilarasi, A., 2021. Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing. J. Ambient Intell. Hum. Comput. 12 (3), 4147–4159.
Pradhan, A., Bisoy, S.K., 2020. A novel load balancing technique for cloud computing platform based on PSO. J. King Saud Univ.-Comput. Inform. Sci.
Karthiban, K., Raj, J.S., 2020. An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm. Soft. Comput. 24 (19), 14933–14942.
Thai, L., Varghese, B., Barker, A., 2018. A survey and taxonomy of resource optimisation for executing bag-of-task applications on public clouds. Fut. Gener. Comput. Syst. 82, 1–11.
Herault, T., Robert, Y., 2015. Fault-Tolerance Techniques For High-Performance Computing. Springer.
Young, J.W., 1974. A first order approximation to the optimum checkpoint interval. Commun. ACM 17 (9), 530–531.
R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction (2011).

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: A survey. J. Artif. Intell. Res. 4, 237–285.

Even-Dar, E., Mansour, Y., Bartlett, P., 2003. Learning rates for q-learning. J. Machine Learn. Res. 5 (1).

Kondo, D., Javadi, B., Iosup, A., Epema, D., 2010. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In: 2010 10th IEEE/ACM International Conference On Cluster, Cloud And Grid Computing. IEEE, pp. 398–407.

Gelper, S., Fried, R., Croux, C., 2010. Robust forecasting with exponential and holt–winters smoothing. J. Forecast. 29 (3), 285–300.

K. Ferreira, J. Stearley, J. H. Laros III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, D. Arnold, Evaluating the viability of process replication reliability for exascale systems, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011, pp. 1–12.

Dongarra, J., Beckman, P., Aerts, P., Cappello, F., Lippert, T., Matsuoka, S., Messina, P., Moore, T., Stevens, R., Trefethen, A., et al., 2009. The international exascale software project: a call to cooperative action by the global high-performance community. Internat. J. High Perform. Comput. Appl. 23 (4), 309–322.

Zheng, G., Ni, X., Kaĺe, L.V., 2012. A scalable double in-memory checkpoint and restart scheme towards exascale. In: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012), pp. 1–6.

Miller Jr, R.G., 1997. Beyond ANOVA: basics of applied statistics. CRC Press.