applied sciences

MDPI

Article

A Novel Fault-Tolerant Aware Task Scheduler Using Deep Reinforcement Learning in Cloud Computing

Mallu Shiva Rama Krishna and Sudheer Mangalampalli *

School of Computer Science and Engineering, VIT-AP University, Amaravati 522237, India; mallu.22phd7084@vitap.ac.in
* Correspondence: sudheerkietmtech@gmail.com

Abstract: Task scheduling poses a wide variety of challenges in the cloud computing paradigm, as heterogeneous tasks from a variety of resources come onto cloud platforms. The most important challenge in this paradigm is to avoid single points of failure, as tasks of various users are running at the cloud provider, and it is very important to improve fault tolerance and maintain negligible downtime in order to render services to a wide range of customers around the world. In this paper, to tackle this challenge, we precisely calculated priorities of tasks for virtual machines (VMs) based on unit electricity cost and these priorities are fed to the scheduler. This scheduler is modeled using a deep reinforcement learning technique which is known as the DQN model to make decisions and generate schedules optimally for VMs based on priorities fed to the scheduler. This research is extensively conducted on Cloudsim. In this research, a real-time dataset known as Google Cloud Jobs is used and is given as input to the algorithm. This research is carried out in two phases by categorizing the dataset as a regular or large dataset with real-time tasks with fixed and varied VMs in both datasets. Our proposed DRFTSA is compared to existing state-of-the-art approaches, i.e., PSO, ACO, and GA algorithms, and results reveal that the proposed DRFTSA minimizes makespan compared to PSO, GA, and ACO by 30.97%, 35.1%, and 37.12%, rates of failure by 39.4%, 44.13%, and 46.19%, and energy consumption by 18.81%, 23.07%, and 28.8%, respectively, for both regular and large datasets for both fixed and varied VMs.

Keywords: DRFTSA; task scheduling; rate of failure; makespan; energy consumption

check for updates

Citation: Shiva Rama Krishna, M.; Mangalampalli, S. A Novel Fault-Tolerant Aware Task Scheduler Using Deep Reinforcement Learning in Cloud Computing. *Appl. Sci.* **2023**, *13*, 12015. https://doi.org/10.3390/app132112015

Academic Editor: Gang Lei

Received: 6 September 2023
Revised: 21 October 2023
Accepted: 31 October 2023
Published: 3 November 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Cloud computing is the on-demand delivery of IT resources to cloud users over the internet, with preconfigurable resources available from different datacenters around the world which can be run with a pay-as-you-go model. The resources in the cloud can also be easily scaled up and down based on a business's needs for cloud user applications [1]. Cloud computing services are rapidly provisioned to customers based on their service-level agreement needs which they make through an agreement with a cloud provider. Diversified types of cloud services are available in cloud environments and they need to be provisioned to customers qualitatively with respect to SLA guarantees provided by the cloud provider [2]. To provide these services demanded by users adequately to customers without having any issues, i.e., execution of tasks, operational costs, failure rate of tasks, energy consumption, etc., a task scheduler is desperately needed to assign incoming heterogeneous tasks to virtual resources in a cloud environment while improving the rendering of services to various cloud users around the world. Therefore, task scheduling plays a major role in the cloud environment which directly impacts cloud providers and indirectly impacts cloud users in terms of quality of service. Quality of service is improved when the failure rate of tasks is reduced in a cloud paradigm. Therefore, failure rate is one of the factors which impacts the quality of a cloud provider. Many authors have proposed various task scheduling algorithms using nature-inspired metaheuristic approaches, i.e., whale [3],

PSO [4], GA [5], and ACO [6] algorithms, as task scheduling is an NP-Hard problem in cloud computing; these nature-inspired approaches generate schedules with respect to the given workload by considering different conditions, but tasks generated from various heterogeneous resources may still have different run-time capacities, and it is difficult to assign tasks to virtual resources in the cloud. Metaheuristic approaches which generate near-optimal solutions are not able to map tasks to precise VMs. Many tasks fail and single points of failure still occur in the cloud paradigm due to inappropriate assignment of tasks to VMs by schedulers. Therefore, in our research, we propose a novel fault-tolerant aware task scheduler using deep reinforcement learning which is integrated as a DQN model into the scheduler which should be able to assign tasks by itself, checking priorities of both tasks and VMs which are fed to the scheduler which generates schedules dynamically based on rewards generated from the evaluated reward function while minimizing the parameters of makespan, rate of failure, and energy consumption.

Motivation and Contributions

This subsection precisely discusses the motivation behind this research and different contributions presented in this research work. Generally, the cloud paradigm provides various IT resources virtually to users based on their requests raised to cloud providers with different SLA guarantees. To provision resources efficiently to cloud users around the world, a cloud provider needs to effectively deploy an efficient scheduler which should benefit both the cloud provider and user. The main motivation behind this research is to minimize the rate of failure of tasks which are to be computed in the cloud paradigm. This can be carried out when the scheduler chooses an appropriate VM for the task that has arrived at the cloud application console. Ineffective scheduling mechanisms cause an increase in makespan, which increases execution time, thus consuming more energy, and also leads to failure of tasks if proper assignment of tasks to a suitable VM is not performed. From this perspective, our motivation behind this research is to develop a scheduler which should minimize the rate of failure and, as such, improve quality of service. In line with this research, we also tackle different parameters, i.e., energy consumption and makespan, which are primary objectives for any task scheduler. The main contributions of our research work are as follows.

1. A novel fault-tolerant aware task scheduling algorithm is developed by capturing priorities of tasks and VMs based on the unit electricity cost of the datacenter's location.
2. A reinforcement-learning-based deep Q-network (DQN) model is used as the methodology to design the scheduler.
3. A real-time dataset, i.e., Google Cloud Jobs, is fed as an input to the scheduler to check the efficacy of our proposed DRFTSA. An extensive set of simulations are conducted on Cloudsim using various scenarios by keeping a fixed number of VMs and varying the VMs in the simulation.
4. In this research, rate of failure, energy consumption, and makespan are evaluated by using the abovementioned dataset with different scenarios by changing the number of VMs.

The rest of the manuscript is organized as follows. Section 2 discusses related works, Section 3 discusses problem formulation and system architecture, Section 4 discusses the methodology used in DRFTSA, Section 5 discusses the simulation and result analysis, and Section 6 discusses the conclusion and future works.

2. Related Works

In [7], the authors proposed a self-learning methodology for energy-conscious work scheduling in a heterogeneous cloud computing environment. It presents a reinforcement learning-based energy-conscious task scheduling method which obtains results in terms of throughput and energy consumption. The authors of [8] proposed a novel task scheduling technique called QL-HEFT. It reduces makespan by using Q-learning HEFT. QL-HEFT is divided into two phases, i.e., task sorting using Q-learning to acquire order and the proces-

sor allocation phase. Both phases are essential to the overall functioning of the algorithm. Experiments demonstrate that QL-HEFT delivers better results in terms of average response time and makespan. In [9], a deep reinforcement learning combined with long short-term memory (DRL-LSTM) approach is developed with which experiments are conducted with real-world datasets retrieved from the Google Cloud Platform, which demonstrated the DRL-LSTM strategy. Compared with existing techniques, it is more effective in terms of considered parameters. DRL-LSTM reduces the cost of using the computer's processing power in comparison to SJF, RR, and IPSO. The authors of [10] designed a scheduler using a Q-learning-based agent which operates at VM level to schedule tasks using a pricing model of VM instances.

The authors of [11] formulate the online scheduling problem as EMDP to construct a deep risk-sensitive reinforcement learning algorithm that accounts for the task arrival dynamics. It evaluates risk and energy consumption as a constraint. For each phase, it calculates optimal parameters which minimizes delay and risk. The simulation results indicate that the proposed algorithm reduced processing time of tasks. The authors of [12] present an artificial bee colony with a Q-learning algorithm, which is a reinforcement learning technique. The MOABCQ method is an independent task scheduling approach for cloud computing. It is proposed to tackle throughput, load balancing, and resource utilization. Cloudsim was used to compare the efficiency of the proposed method with existing load balancing and scheduling techniques. Experimental results indicated that MOABCQ tackles cost, throughput, makespan, degree of imbalance, and average resource utilization.

The authors of [13] provide an effective approach for job scheduling in the specified cloud that makes use of the MapReduce framework [14], GA-WOA. Task features are first retrieved from the client's task. Enormous tasks are then divided into various tasks using the MapReduce framework. Finally, the GA-WOA algorithm is used to effectively allocate tasks. The authors of [15] developed a new deep-reinforcement-learning-based RP, TS system designed to cut down on energy costs on a large-scale basis for CSPs with a lot of servers which receive huge requests from users. A two-stage RP-TS processor based on deep Q-learning is developed to automatically make the best decisions for the future to adapt to the new environment by learning, e.g., user request patterns, the real price of electricity. The authors of [16] builds a scheduler by improving the DQN method and introducing a noise network to create the novel NoisyDQN algorithm, which both expands the scope of the reinforcement learning model's exploratory capabilities and raises the ceiling of its possible performance. Experimental results show that the proposed algorithm significantly decreases the average standard deviation of cluster CPU utilization, which is achieved by decreasing the cluster's power consumption, decreasing user waiting time, and increasing response speed and load balancing.

In [17], the authors focused on minimization of makespan, cost, and average delay when working under restrictions of VMs and sensitive deadlines. They developed a constraint-adaptive online task scheduling method using double deep Q-learning. They used Gaussian distribution of relevant parameters as part of the state space, which mitigates the effect of increasing the number of VMs on the input dimension of model. The model can be fine-tuned for a variety of objectives and workloads by carefully modeling the reward function. The authors tested the algorithm under varying workloads and compared its results to those of three heuristic algorithms (round robin, min–max, and FCFS).

The authors of [18] introduce TeNOR, which is an effective resource and network service mapping technique to build a microservice architecture for network function virtualization. Reinforcement learning is used in one approach to map available resources. Experiments show that the RL approach to large-scale NS has gained widespread popularity as a result of its iterative learning of optimal technique for resource mapping. In [19], a flexible framework for handling faults in a cloud computing model is developed, which is integrated into a Hadoop framework.. It uses Markov decision process and ML techniques to generated schedules. ATLAS+, an adaptive fault-aware scheduler for Hadoop, was used in the framework's deployment. Compared to Fair, FIFO, and Capacity Hadoop schedulers,

it greatly minimizes execution time and rate of failure and improves resource utilization. The authors of [20] proposed a task scheduling mechanism formulated for heterogeneous machine learning applications to run their tasks on edge computing systems to improve task completion rate and minimize energy consumption. They leveraged fairness among the different heterogeneous applications without allowing bias of the scheduler towards these tasks. In [21], the authors developed a fault-tolerant workflow scheduler which is based on a combination of a heuristic approach and a reinforcement learning mechanism. The authors used a Markov decision process to generate schedules while resubmitting and replication strategies are used to tackle fault tolerance in this mechanism. In [22], a hybridized approach formulated by the authors is used to address computation time of tasks by using whale optimization in the generation of scheduling decisions and convolutional neural networks for optimization of generated schedules by addressing computation time. The authors of [23] modeled a layered architecture which anticipates failures and developed a proactive mechanism by migrating the VM to another pool of VMs. This mechanism improved the makespan of the proposed approach over existing state-of-the-art algorithms. In [24], the authors developed a task assignment mechanism in fog computing nodes while achieving reliability in executing tasks, i.e., execution of tasks under deadline constraints. The main aim of this approach is to identify suitable fog nodes to compute the tasks and assign tasks to them by using an RL-based approach. This RELIEF approach dominates other existing algorithms by improving reliability.

The Section 2 and Table 1 represent various task scheduling algorithms modeled using various metaheuristic approaches as well as ML techniques. We carefully studied the existing literature and identified that some of the previous authors addressed parameters which influence the task scheduling such as makespan, computation time, resource utilization, and fault tolerance that give near-optimal solutions and generate schedules. However, in our research, the novelty is that we carefully evaluated the suitability of tasks to VMs based on their priorities. In this research, priorities mean both tasks and VMs based on electricity unit cost. Thus, a task with the highest priority should be mapped to a VM with the highest priority that means a VM with a low electricity unit cost. These priority calculations of both tasks and VMs based on electricity cost were not carried out by the previous authors. Moreover, we calculated task priorities based on the length of the task and processing capacity of VMs. When there is a better suitability of a VM for a task, then there is much less chance of failure of the task. These priorities are fed to the scheduler which is integrated with the DQN model to generate schedules while addressing parameters, i.e., makespan, rate of failure, and energy consumption.

Table 1. Analysis of existing task scheduling algorithms.

Author	Methodology	Parameters
[4]	IFWA-BSA	Execution time, total cost
[5]	HEFT	Average response time, scalability
[6]	DRL-LSTM approach	Makespan, total cost, and energy consumption
[7]	DRL-based agents	Cost, makespan
[8]	ECNDP	Resource utilization ratio, throughput, makespan
[9]	MOABCQ	Makespan, resource utilization
[10]	ABC with a Q-learning algorithm	Minimizing makespan, cost reduction, minimizing degree of imbalance
[11]	Q-learning HEFT	Total cost, response time
[12]	DRL-based RP and TS system	Processing cost, resource utilization, makespan
[13]	NoisyD3QN-algorithm-based cloud task scheduling model	Task transfer time, estimated completion time
[14]	Double deep Q-learning	Makespan, cost, execution time
[15]	RL approach to large-scale NS	Makespan, load balancing
[16]	MDP and ML techniques to create schedules. ATLAS+	Execution time, overall datacenter cost

Table 1. *Cont.*

Author	Methodology	Parameters
[17]	RL to make a scheduling model for cloud computing	Makespan, total power cost in datacenters, energy consumption, migration time
[18]	RL task scheduling with Q-learning	Makespan, total power cost, energy consumption
[19]	Task scheduling optimization algorithm called MPSO	Makespan, energy consumption, packet delivery ratio, trust value
[20]	FELARE	Ontime task completion rate, energy saving
[21]	RLFTWS	Makespan, resource usage
[22]	WBCNF	Computation time of tasks
[23]	BRCH-GWO	Makespan
[24]	RELIEF	Communication delay, reliability
[25]	PSO-based multipurpose algorithm	Turnaround time, makespan
[26]	Regressive WO algorithm	Processing cost, load balancing tasks
[27]	GO	Makespan, resource utilization
[28]	Dynamic task scheduling algorithm based on an improved GA	Total execution time and resource utilization ratio
[29]	PSO based on an AC algorithm	Task completion time, makespan
[30]	DRL-based task scheduling	Makespan, computation time
[31]	MRLCC is an approach for organizing tasks that are based on Meta RL	Energy consumption, total cost, makespan
[32]	A novel DRL-based framework	Cost and throughput, makespan
[33]	RLFTWS	Execution time, degree of imbalance
[15]	DRL model	Response time, makespan, CPU utilization
[34]	Deep reinforcement learning with PPSO	SLA violation, makespan
[35]	DRL Cloud is an NDR-learning-based RP and TS system	Estimated completion time, resource utilization
[36]	Deep Q-network model	Degree of imbalance, cost, makespan
[37]	SDM reinforcement learning	Energy consumption, resource utilization
[38]	DRLHCE	Response time, degree of imbalance
[39]	DQN	Makespan, total cost
[40]	Reinforcement learning	Makespan
[41]	DDDQN-TS	Task response time
[31]	Q-learning	Makespan

3. Problem Formulation and Proposed System Architecture

This section represents the formulation of the problem and proposed system architecture. Initially, the task scheduling problem is formulated in such a way that k tasks are indicated as $T^k = \left\{T^1, T^2, T^3, \ldots\ldots T^k\right\}$, n virtual machines are indicated as $V^n = \left\{V^1, V^2, V^3 \ldots..V^n\right\}$, s hosts are indicated as $PH^s = \left\{PH^1, PH^2, \ldots.PH^s\right\}$, and z datacenters are indicated as $D^z = \left\{D^1, D^2, \ldots D^z\right\}$. After consideration of the above entities, T^k tasks are mapped to V^n VMs which are placed in PH^s hosts and placed in D^t datacenters by considering priorities of the task and VMs while minimizing makespan, rate of failure, and energy consumption.

The Figure 1 represents the system architecture of the proposed deep reinforcement learning fault-tolerant task scheduling algorithm. In this architecture, initially a variety of users submit heterogeneous tasks to the cloud platform. At the cloud provider, these submitted tasks are captured by a broker which is an agent-based entity that takes all tasks and submits these tasks to the task manager. In our research, at the task manager level, we used logic to calculate task priorities and VM priorities. All tasks submitted by the broker to the task manager are to be prioritized based on task length and run-time capacities. VM priorities are calculated using the unit cost of electricity at the corresponding datacenter. These priorities are fed to the reinforcement-based scheduler. The scheduler is connected to the resource manager as it needs to track resources in the cloud. With consideration of these priorities, the scheduler will add these task sequences to the execution queue from which tasks are assigned to VMs based on the priorities. In this scheduler, we used a deep Q-learning model which works based on reinforcement learning, in which it takes input and analyzes tasks fed to it and schedules tasks to appropriate VMs. This is an agent-based

learning model; initially, it randomly schedules tasks based on priorities and looks for outcomes and if tasks are precisely distributed to VMs. Then, a positive reward will be raised by the DQN model and it learns, for a certain number of tasks generated previously, the best schedule and if there is a random distribution of task-generated schedules or if the outcome is not good then it considers it as a negative reward and it will learn from previous generated actions. In this research, we used a reward-based agent model which learns on its own based on the situation, i.e., type and number of tasks generated in this scenario. In our research, our main objective using this DRFTSA scheduler is to minimize the rate of failure which is a QoS parameter and it is very important from the point of view of the cloud provider. Failure in the cloud environment is often because many users execute their requests from all around the world and they are running their tasks from remote locations. Therefore, there is a chance to have a single point of failure in the cloud environment due to the unavailability of the VMs, network issues, etc. Our proposed DRFTSA scheduler keeps track of all the tasks submitted to it and, based on considered priorities, the scheduler needs to make a decision on its own and generate schedules effectively. Mathematical modeling of the proposed DRFTSA is shown below in Table 2.

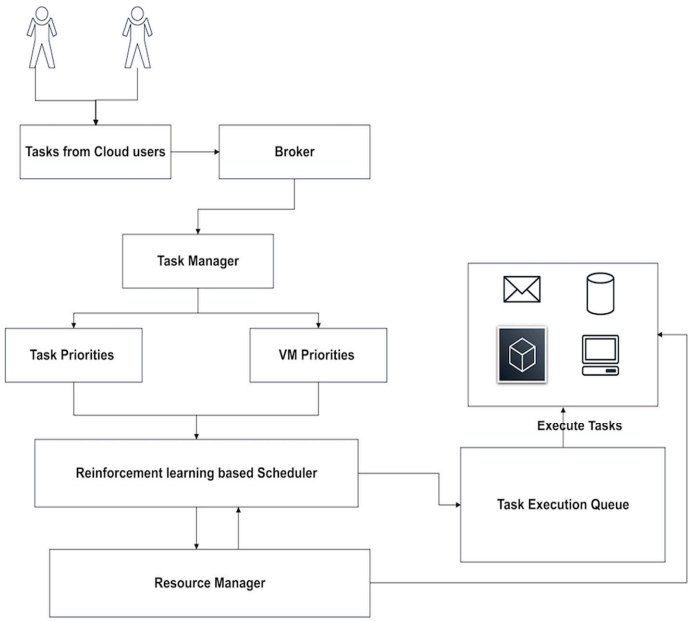


Figure 1. System architecture of DRFTSA.

Table 2. Mathematical notations used in proposed DRFTSA.

Name	Notation
$Crlo^{V}$	Current execution of tasks
$Crlo^{PH_s}$	Current execution of tasks on all physical machines/nodes
PR_{V}^{cap}	Processing capacity of VMs
T_{le}^{k}	Task length
T^{pi}	Task priorities
V_{n}^{pi}	Virtual machine priorities based on unit electricity cost
ms_k	Makespan of k tasks
$ene_{cons}(V^n)$	Energy consumption of n tasks
$Rate_{failures}$	Rate of failure
act^{T}	Action space for T tasks
$sta^{t^{T}}$	State space for T tasks
Rew_{fun}	Reward function
$M\partial$	Replay memory
μ	Learning rate of agent
ϑ	Time for agent learning

In the first step, after submission of various heterogeneous tasks by the users to the cloud platform, the broker captures those tasks and in turn submits tasks to the task manager. At this level, to calculate priorities of tasks, initially the task manager needs to know the currently running tasks on VMs and then it needs to know the current execution of tasks load on all physical machines/nodes.

The current execution of tasks running on VMs is calculated using below Equation (1).

$$Crlo^V = \sum Crlo_n^T \tag{1}$$

where $Crlo_n^T$ indicates current execution of tasks on n VMs.

Now, we calculate the current execution of tasks on all physical machines/nodes using Equation (2).

$$Crlo^{PH_s} = \frac{Crlo^V}{PH^s} \tag{2}$$

where $Crlo^{PH_s}$ indicates the current execution of tasks running on s physical machines/nodes.

In this research, an important aspect is to identify the appropriate VM which can accommodate tasks submitted to the cloud console. Therefore, the scheduler needs to make a decision whether to assign tasks to a specific VM or not. This decision is mainly based on the processing capacity of a VM. It is calculated in Equation (3) below.

$$PR_V^{cap} = PR_{no} \times PR_{MIPS} \tag{3}$$

where PR_V^{cap} indicates the processing capacity of a VM, the number of processing elements is indicated by PR_{no}, and the number of instructions processed per second is indicated by PR_{MIPS}.

After submitting the requests to the task manager, it need to identify two priorities, i.e., the task and VM priorities which prioritize the input task sequence which can be fed to the scheduler. Basically, task priority depends on its length and the run-time capacity of the VM. Therefore, task length is calculated in Equation (4) below.

$$T_{le}^k = T^{MIPS} \times T^{PR} \tag{4}$$

Now, task priorities are calculated using Equation (5) below. They are defined as the ratio of the length of the task to the processing capacity of a VM to which the task is submitted.

$$T^{pi} = \frac{T_{le}^k}{PR_V^{cap}} \tag{5}$$

Now, VM priorities are calculated with respect to unit cost of electricity as our aim is to identify a VM with a low unit cost of electricity but with high computing power to execute tasks as the provider needs to minimize energy consumption in the datacenter. Therefore, they are calculated using Equation (6) below.

$$V_n^{pi} = \frac{ele^h}{ele^{D^q}} \tag{6}$$

After careful identification of priorities of both tasks and VMs based on unit cost of electricity, we carefully modeled the parameters for evaluation of parameters, i.e., makespan, rate of failure, and energy consumption at datacenters. Makespan impacts the task scheduling process directly as, if execution time on a VM is reduced, then the VM is available for the next task which needs to be executed and, in reverse, if execution time increases on a VM, this impacts the QoS of the scheduler and execution cost can be increased. This is the reason why we chose makespan as a primary objective in this work. It is calculated using Equation (7) below.

$$ms_k = ava_n^V + ET_k \tag{7}$$

where ms_k indicates makespan for k tasks, ava_n^V is availability to run tasks with n VMs, ET_k is execution time for k tasks.

Minimizing energy consumption in datacenters is our next objective because, in a cloud computing paradigm, from the point of view of the cloud provider and from the point of view of the environment, to reduce carbon footprints to achieve green cloud computing it is important to minimize energy consumption at datacenters. It is calculated using Equation (8) below.

$$ene_{cons}(V^n) = \int_0^n ene_{cons}^{com}(V^n, T) + \int_0^n ene_{cons}^{idle}(V^n, T)dt \tag{8}$$

Total energy consumption at datacenters is calculated using Equation (9) below.

$$ToT_{cons}^{ene} = \sum ene_{cons}(V^n) \tag{9}$$

After Equations (8) and (9), we calculated rate of failure as our objective is to develop a fault-tolerant scheduler to minimize single points of failure in the cloud paradigm. It is calculated using Equation (10).

$$Rate_{failures} = \frac{\sum no\ of\ failures\ of\ DRFTSA}{\sum no\ of\ failures\ of\ existing\ algorithms} \tag{10}$$

After designing a detailed mathematical model for the task scheduler, we integrated a machine reinforcement learning mechanism into our proposed *DRFTSA* scheduler.

4. Methodology Used in DRFTSA

The Figure 2 represents the working of deep reinforcement learning for scheduling tasks in the cloud paradigm. It basically works based on agents and these agents will infer the knowledge by listening to consequences that occurred in the scheduling paradigm by giving rewards. The outcome of this model can be in the form of rewards which may be positive or negative and, if the reward is positive, the agent will update the schedule for the current state and update its state. Vice versa, if the outcome is negative, the agent will learn from the current schedule and it should not be generated for the current scenario next time. In this research, we used the DQN model which uses a Q-learning model [4] which does not need knowledge of previous history and it learns on its own by randomly spreading tasks based on two states available in the Q-learning table, i.e., state and action spaces. In this model, the Q-learning function automatically makes a decision by checking tuples available in the Q-learning table indicated as $Q(sta^T, act^T)$. This function updates for every iteration using Equation (11) below.

$$Q\left(sta^T, act^T\right) \leftarrow Q\left(sta^T, act^T\right) + \mu * \left[RE^T + \mu * \max_b Q\left(sta^{T+1},\ act\right) - Q\left(sta^T, act^T\right)\right] \tag{11}$$

where μ is learning rate of the agent, RE^T is the reward generated for an agent for a certain action that may be either positive or negative. The range of both the learning rate and reward generated for the agent always lies between 0 and 1. As discussed previously, in Q-learning there are different states which consist of certain actions that need to be carried out.

4.1. Action Space

In this research, we considered k tasks indicated as $T^k = \left\{T^1, T^2, T^3, \ldots\ldots T^k\right\}$, n virtual machines indicated as $V^n = \left\{V^1, V^2, V^3 \ldots..V^n\right\}$. These k tasks, which are submitted to the cloud platform, and then captured by the cloud broker and submitted to the task manager. Now, the task manager needs to calculate two types of priorities i.e., task and VM priorities. In this research, these two priorities are fed to the scheduler which is already integrated with DQN model to make decisions based on priorities fed to the

scheduler. After checking priorities, the scheduler needs to send these prioritized task sequences on to the execution queue which should be sent to execution with respective virtual resources in the cloud paradigm. It is calculated using Equation (12) below as action space.

$$act^T = \left[V^1, V^2, V^3 \ldots .. V^n\right] \tag{12}$$

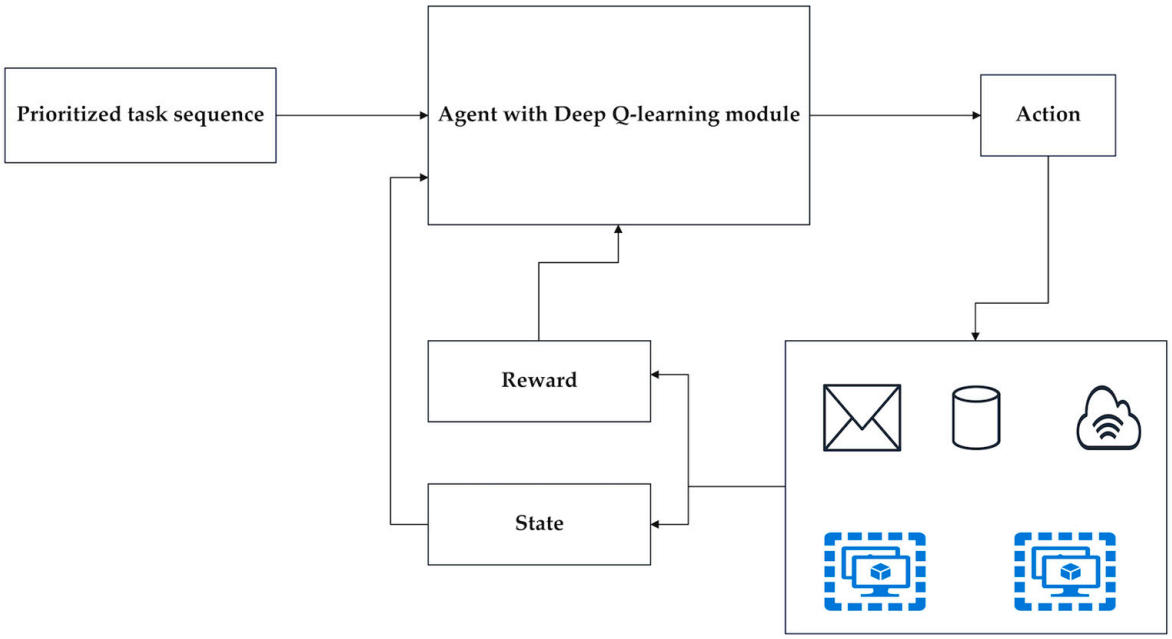


Figure 2. Working of Deep Reinforcement learning for DRFTSA.

4.2. State Space

This subsection represents the state space consisting of both states of a task and states of the VM. In our case, when a task T arrives at a time t, it is indicated as t^T. Now, the state of a task T is calculated using Equations (13) and (14).

$$sta^{t^T} = sta^T \cup sta_V^{t^T} \tag{13}$$

where sta^T is the state of task T at time instance t, $sta_V^{t^T}$ indicates the state of task T at time instance t.

$$sta^{t^T} = \left[T^{pi},\ V_n^{pi}, ms_k, ToT_{cons}^{ene}, Rate_{failures}\right] \tag{14}$$

After calculating the action and state spaces, we calculated the reward function which is either positive or negative and through which the agent can receive either a positive or negative reward. It is calculated using Equation (15).

$$Rew_{fun} = \min\left(ms_k,\ ToT_{cons}^{ene}, Rate_{failures}\right) \tag{15}$$

4.3. Training Agent in DQN Model

This subsection clearly presents how the agent in the DQN model is trained and the amount of replay memory considered in our work, the nature of state updates for every iteration, learning frequency considered, and learning rate identified are shown. When a task arrives at the scheduler in our research, the DQN integrated in the scheduler needs to make a decision. It works as follows. Initially, the scheduler looks for a task and, when it arrives, it randomly schedules that task to a VM with a probability *prob* and this probablility reduces to zero over a period of time. After the first decision of the generation of schedules, the scheduler looks at the Q-learning table for existing Q-values, i.e., schedules which were generated in its previous iteration. After generating the schedules with decisions in that

corresponding state, state and action values are updated accordingly in the Q-table. The decision generated, i.e., either positive or negative, is stored in the Q-table as the state values stored in replay memory, indicated as ∂. For every iteration in the scheduler, in ∂ values stored are $\left(act^t, s^{t^T}, Rew_{fun}, s^{t^{T+1}}\right)$. Replay memory capacity is indicated as $M\partial$. All iterations will run as batches and they are indicated as $G\partial$. In our research, time taken for an agent to make a decision and to generate a schedule is set to 10 ms, time for agent learning is indicated as ϑ, learning frequency is set to be 1.

4.4. Proposed Deep Reinforcement Fault-Tolerant Task Scheduling Algorithm (DRFTSA)

The Algorithm 1 and Figure 3 is the working flow of the proposed deep reinforcement fault-tolerant task scheduling algorithm. Initially, required parameters are probability with which a task is to be scheduled randomly, learning frequency, replay memory, and batch of replay memory values. Then, the Q-function is initialized and it is set to 0. After this step, task and VM priorities are calculated. After checking priorities of tasks and VMs, the scheduler needs to identify the resource manager for availability of resources and then, if they are available, schedule tasks according to priorities considered and check the reward in the next step. If it is positive, it is checked whether the parameters are minimized or not and then the state is updated. This is repeated for all batches of tasks until they are completed.

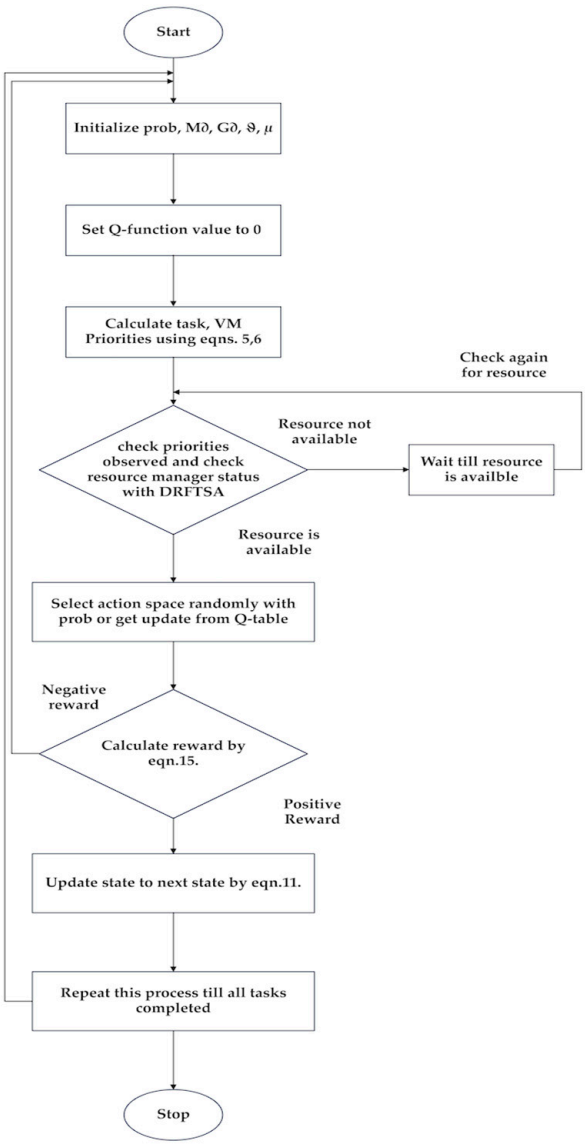


Figure 3. Working flow of proposed DRFTSA.

Algorithm 1: Proposed Deep Reinforcement Fault-Tolerant Task Scheduling Algorithm (DRFTSA).

Input: No. of tasks, i.e., $T^k = \left\{T^1, T^2, T^3, \ldots\ldots T^k\right\}$, no. of virtual machines, i.e., $V^n = \{V^1, V^2, V^3 \ldots..V^n\}$, physical hosts/nodes $PH^s = \{PH^1, PH^2, \ldots.PH^s\}$, datacenters $D^z = \left\{D^1, D^2, \ldots D^z\right\}$.
Output: Generation of schedules for T^k tasks mapped to V^n virtual machines by tackling makespan, rate of failure, energy consumption.
Start
Initialize parameters, i.e., *prob*, $M\partial$, $G\partial$, ϑ, μ.
Set $Q\left(sta^T, act^T\right)$ to 0.
    for every batch do
    select sta^{t^T}
    calculate task priorities using Equation (5).
    calculate VM priorities using Equation (6).
    for every batch do
        identify priorities of both tasks, VMs to schedule based on resource manager status
    select action space and choose a VM with *prob* or select arg max $Q(sta^{t^T}, act)$
    calculate reward function by Equation (15).
    check the reward for action done. (Identify parameters i.e., ms_k, ToT^{ene}_{cons}, $Rate_{failures}$)
    Update Equation (11). For next state
    Update sta^T to sta^{T+1}.
    Repeat this process until the last batch
End do
End do
Stop

5. Simulation and Results

In this section, we precisely discuss the simulation and results generated by our proposed approach, i.e., deep reinforcement fault-tolerant task scheduling algorithm (DRFTSA). This section is divided into different subsections. Section 5.1 discusses the simulation settings and dataset we have considered in our research, Section 5.2 discusses the presentation of the generated makespan with a regular dataset with fixed VMs, Section 5.3 discusses the presentation of the generated makespan with a regular dataset with varied VMs, Section 5.4 discusses the presentation of the generated makespan with a large dataset with fixed VMs, Section 5.5 discusses the presentation of the generated makespan with a large dataset with varied VMs, Section 5.6 discusses the presentation of the generated energy consumption with a regular dataset with fixed VMs, Section 5.7 discusses the presentation of the generated energy consumption with a regular dataset with varied VMs, Section 5.8 discusses the presentation of the generated energy consumption with a large dataset with fixed VMs, Section 5.9 discusses the presentation of the generated energy consumption with a large dataset with varied VMs, Section 5.10 discusses the presentation of the generated rate of failure with a regular dataset with fixed VMs, Section 5.11 discusses the presentation of the generated rate of failure with a regular dataset with varied VMs, Section 5.12 discusses the presentation of the generated rate of failure with a large dataset with fixed VMs, Section 5.13 discusses the presentation of the generated rate of failure with a large dataset with varied VMs.

5.1. Simulation Settings and Dataset Used for DRFTSA

This subsection discusses the configured simulation settings for DRFTSA as well as the dataset used in our research work. For our research work, Cloudsim [42] was used as a platform to simulate the cloud environment. After identifying the platform, we configured this simulation platform in our physical host. For simulation of Cloudsim, we need Java for backend support and the entire code for Cloudsim written in Java. Cloudsim installed in our physical host is a bear metal system with 16 GB RAM, 1 TB hard disk SSD, and i7 processor. The workload is generated in our research by giving an input of tasks from

the real-time Google Cloud Jobs dataset captured from a supercomputing cluster which consists of various tasks with different run-time capacities. It is available from [43]. It is a real-time workload trace and available in terms of million instructions per second (MIPS) It consists of parallel worklogs which are obtained from Google cluster traces. Initially, we identified the tasks available in the dataset as small, medium, large, and very large tasks and they account for 25%, 35%, 30%, 4%, and 6% respectively. In our case, a regular dataset consists of small and medium tasks and a large dataset consists of large and very large tasks. After identifying this, we categorized datasets as regular and large datasets by running an algorithm with fixed and changing VMs. In our research, with the proposed DRFTSA we calculated makespan, energy consumption, and rate of failure. The proposed DRFTSA is compared against existing state-of-the-art approaches, i.e., PSO, ACO, and GA. The standard configuration settings are taken from [44] to carry out simulations. Table 3 presents configuration settings used in our research for simulation.

Table 3. Configuration settings used for DRFTSA in simulation.

Name of the Entity	Value of the Entity
Tasks	1000
Length of Tasks	750,000
Memory of Physical Host	16 GB
Storage Capacity of Physical Host	1 TB
Network Bandwidth of Physical Host	500 MBPS
Number of VMs	50
Memory of VMs	2 GB
Network Bandwidth of VMs	20 MBPS
Operating System of VMs	Ubuntu
Type of Hypervisor	Monolithic
Name of Hypervisor	Citrix Xen
Number of Datacenters Used	10–80

5.2. Calculation of Makespan for DRFTSA Using Regular Dataset with Fixed Number of VMs

This subsection clearly presents the calculation of makespan for the proposed DRFTSA by giving input from a real-time dataset, i.e., Google Cloud Jobs. In Section 5.1, as mentioned previously, we categorize datasets as regular and large datasets and 100 to 600 tasks are used in the regular dataset and then the makespan of DRFTSA is calculated. Any task scheduler's performance depends on the makespan and here in our research when we choose a suitable VM for a task by considering the priorities of both tasks and VMs based on electricity cost effects and makespan. We choose a VM which can process the task according to its processing capacity and therefore there is no chance of an increase in execution time and failures. It directly impacts performance of the task scheduler as, if makespan is lowered or minimized for a task scheduler, then we can say that performance of the scheduler is good and it is defined as the time taken to execute a task on a virtual machine. If the execution time for a task on a VM is increased, it results in poor performance of the scheduler. Therefore, in our research work while developing DRFTSA, we calculated makespan by using a regular dataset with a fixed number of VMs and generated tasks. The simulation ran for 50 iterations to check the efficacy of our proposed DRFTSA against PSO, GA, and ACO approaches. Makespans generated for PSO with 100, 200, 300, 400, 500, 600 tasks are 684.3, 712.54, 732.57, 812.78, 836.9, 895.87, respectively. Makespans generated for GA with 100, 200, 300, 400, 500, 600 tasks are 725.62, 756.09, 791.36, 854.03, 912.36, 953.47, respectively. Makespans generated for ACO with 100, 200, 300, 400, 500, 600 tasks are 545.17, 759.97, 834.18, 871.22, 891.22, 932.18, as shown in Table 3, respectively.

Makespans generated for DRFTSA with 100, 200, 300, 400, 500, 600 tasks are 421.22, 503.12, 519.96, 485.78, 505.88, 456.27, respectively. From Table 4 and Figure 4 it is evident that DRFTSA minimized makespan compared to state-of-the-art approaches while running with a regular dataset and fixed number of VMs.

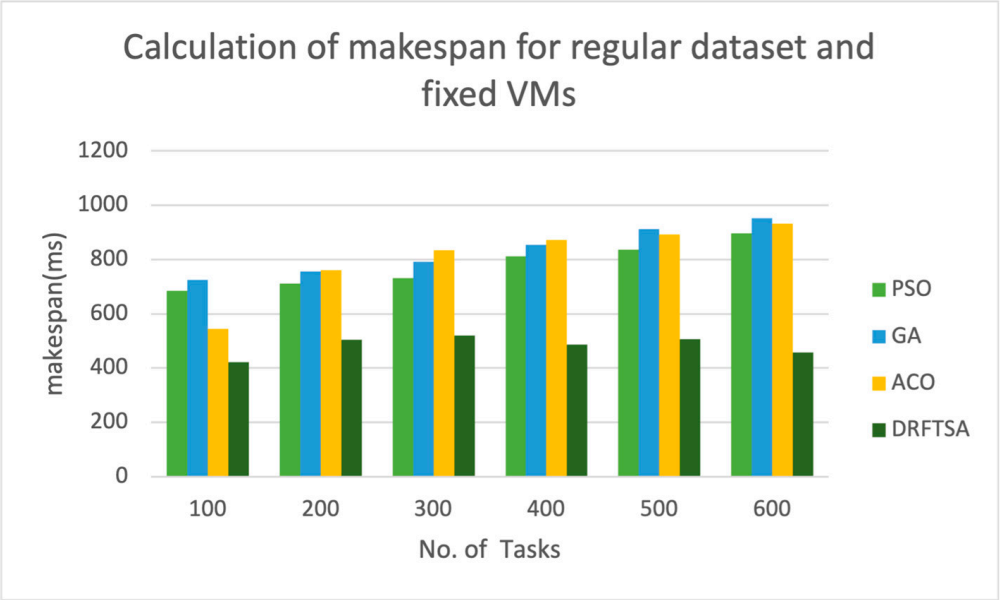


Figure 4. Calculation of makespan for regular dataset using fixed VMs.

Table 4. Makespan for DRFTSA using regular dataset with fixed number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
100	684.3	725.62	545.17	421.22
200	712.54	756.09	759.97	503.12
300	732.57	791.36	834.18	519.96
400	812.78	854.03	871.22	485.78
500	836.9	912.36	891.22	505.88
600	895.87	953.47	932.18	456.27

5.3. Calculation of Makespan for DRFTSA Using Regular Dataset by Varying Number of VMs

In this subsection, we present the generated makespan for DRFTSA using the regular dataset by varying the number of VMs and using 100 to 600 tasks. In this scenario, we have varied VMs by assigning 10 VMs to 100 tasks, 20 VMs to 200 tasks, 30 VMs to 300 tasks, 40 VMs to 400 tasks, 50 VMs to 500 tasks, and 60 VMs to 600 tasks to check efficacy of the scheduler in terms of minimization of makespan. The generated makespans for PSO with 100, 200, 300, 400, 500, 600 tasks are 586.71, 676.47, 687.87, 712.36, 757.86, 809.27, respectively. Generated makespans for GA with 100, 200, 300, 400, 500, 600 tasks are 603.39, 673.18, 706.88, 759.52, 784.18, 828.88, respectively. Generated makespans for ACO with 100, 200, 300, 400, 500, 600 tasks are 525.88, 712.74, 802.18, 769.17, 813.87, 798.54, respectively. Generated makespans for DRFTSA with 100, 200, 300, 400, 500, 600 tasks are 389.19, 432.17, 447.91, 478.23, 483.63, 587.41, as shown in Table 4, respectively.

From Table 5 and Figure 5, it is evident that DRFTSA minimized makespan compared to state-of-the-art approaches while running with a regular dataset and using variable numbers of VMs.

Table 5. Makespan for DRFTSA using regular dataset by varying number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
100	586.71	603.39	525.88	389.19
200	676.47	673.18	712.74	432.17
300	687.87	706.88	802.18	447.91
400	712.36	759.52	769.17	478.23
500	757.86	784.18	813.87	483.63
600	809.27	828.88	798.54	587.41

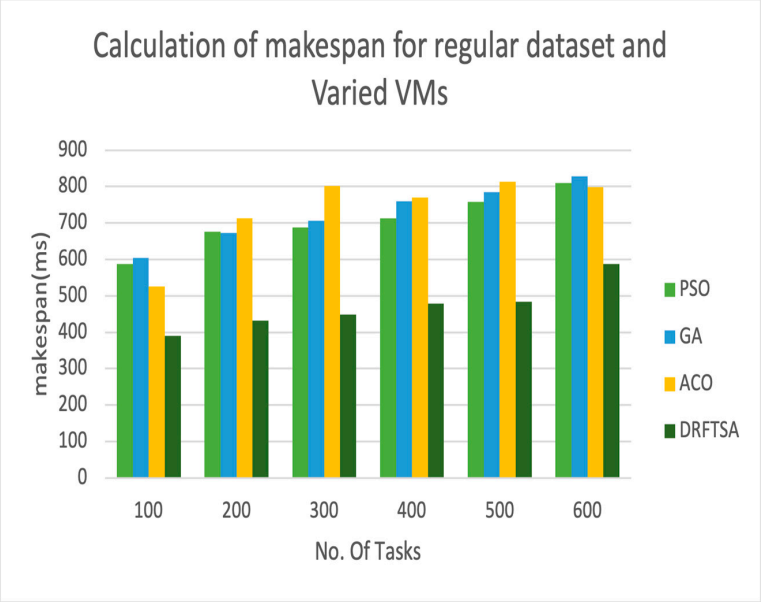


Figure 5. Calculation of makespan for regular dataset by varying number of VMs.

5.4. Calculation of Makespan for DRFTSA Using Large Dataset by Using Fixed Number of VMs

In this subsection, we present the generated makespan for DRFTSA using the large dataset by using fixed number of VMs and 700 to 1000 tasks. In this scenario, we have used a large dataset and a fixed number of VMs. We compared the proposed DRFTSA with state-of-the-art approaches, i.e., PSO, GA, and ACO. Generated makespan for PSO with 700, 800, 900, 1000 tasks are 784.27, 811.26, 943.78, 1048.86, respectively. Generated makespan for GA with 700, 800, 900, 1000 tasks are 767.96, 827.19, 958.14, 1127.36, respectively. Generated makespan for ACO with 700, 800, 900, 1000 tasks are 812.98, 853.36, 876.17, 1176.17, respectively. Generated makespan for DRFTSA with 700, 800, 900, 1000 tasks are 624.78, 645.87, 723.18, 767.99, respectively.

From Table 6 and Figure 6, it is evident that DRFTSA minimized makespan compared to state-of-the-art approaches while running with a large dataset and using fixed numbers of VMs.

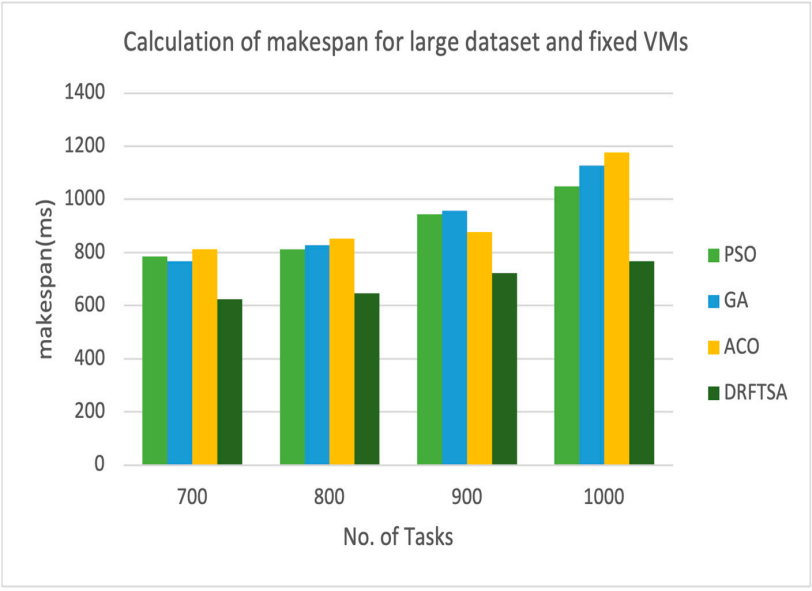


Figure 6. Calculation of makespan for large dataset using fixed number of VMs.

Table 6. Makespan for DRFTSA using large dataset by using fixed number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
700	784.27	767.96	812.98	624.78
800	811.26	827.19	853.36	645.87
900	943.78	958.14	876.17	723.18
1000	1048.86	1127.36	1176.17	767.99

5.5. Calculation of Makespan for DRFTSA Using Large Dataset by Using Variable Number of VMs

In this subsection, we present the generated makespan for DRFTSA using the large dataset by using fixed numbers of VMs and 700 to 1000 tasks. Variable numbers of VMs are used in this scenario: 50 VMs are assigned to 700 tasks, 60 VMs are assigned to 800 tasks, 70 VMs are assigned to 900 tasks, 80 VMs are assigned to 1000 tasks. We compared our proposed DRFTSA with PSO, GA, and ACO approaches. Generated makespan for PSO with 700, 800, 900, 1000 tasks are 634.87, 685.12, 709.18, 725.88, respectively. Generated makespan for GA with 700, 800, 900, 1000 tasks are 653.02, 674.11, 722.98, 763.46, respectively. Generated makespan for ACO with 700, 800, 900, 1000 tasks are 688.18, 724.37, 766.91, 843.99, respectively. Generated makespan for DRFTSA with 700, 800, 900, 1000 tasks are 421.57, 463.14, 521.66, 540.28, respectively.

The generated results from Table 7 and Figure 7 revealed that DRFTSA minimized makespan compared to state-of-the-art approaches when a large dataset is used while varying the number of VMs.

Table 7. Makespan for DRFTSA using large dataset by using variable number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
700	634.87	653.02	688.18	421.57
800	685.12	674.11	724.37	463.14
900	709.18	722.98	766.91	521.66
1000	725.88	763.46	843.99	540.28

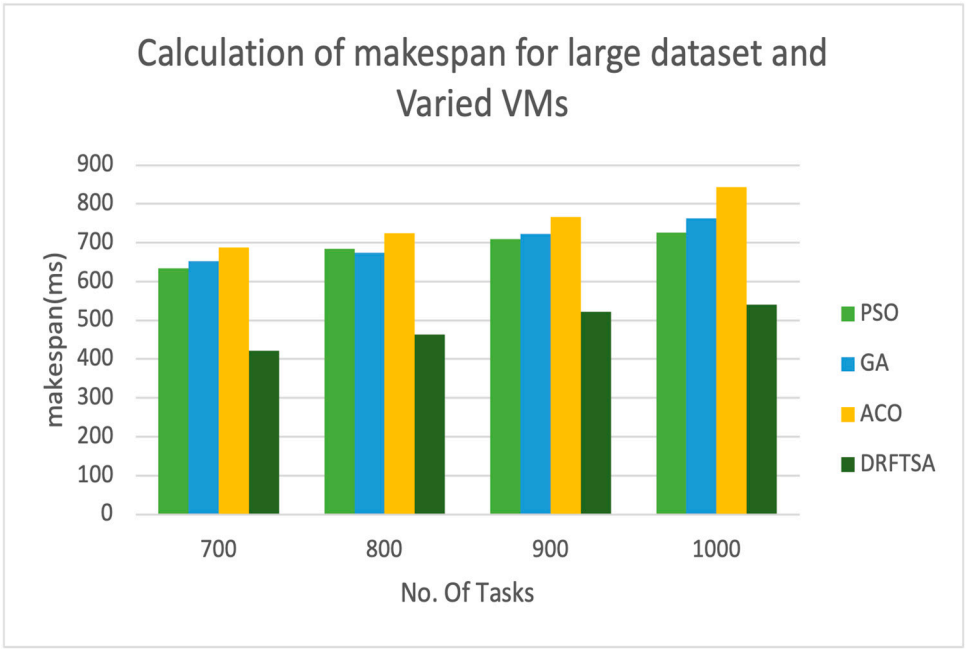


Figure 7. Calculation of makespan for large dataset using variable number of VMs.

5.6. Calculation of Energy Consumption for DRFTSA Using Regular Dataset by Using Fixed Number of VMs

In this subsection, we precisely present the energy consumption generated by the proposed DRFTSA by using the regular dataset and using a fixed number of VMs for all tasks. The main reason to calculate the energy consumption in this research is that, from the point of view of the cloud provider, it is an important parameter as, when more tasks are executed with less energy consumption, the cloud provider will have more profit in terms of money and can provide high-quality services to its customers for a lower price. Therefore, this parameter has more impact on both the cloud provider and user but, in order to minimize consumption of energy, a proper scheduling mechanism with dynamic decision-making capacity to map tasks to appropriate VMs is needed. Therefore, in this research our proposed DRFTSA carefully considered priorities of all task sequences and priorities of VMs and then scheduled tasks appropriately to VMs in respective datacenters. To evaluate the efficacy of DRFTSA in terms of energy consumption, we used a real-time dataset which consists of heterogeneous tasks and then used a regular dataset to be given as input to our algorithm to calculate energy consumption by using fixed number of VMs and various benchmark approaches to evaluate our proposed DRFTSA. The table below shows energy consumption of DRFTSA.

Table 8 presents generated energy consumption for various benchmark approaches and DRFTSA. Generated energy consumption for PSO with 100, 200, 300, 400, 500, 600 tasks is 77.68, 82.45, 91.38, 103.67, 125.72, 134.12, respectively. Generated energy consumption for GA with 100, 200, 300, 400, 500, 600 tasks is 81.79, 85.35, 94.17, 113.57, 130.93, 142.47, respectively. Generated energy consumption for ACO with 100, 200, 300, 400, 500, 600 tasks is 91.85, 89.32, 97.65, 120.32, 134.16, 152.82, respectively. Generated energy consumption for DRFTSA with 100, 200, 300, 400, 500, 600 tasks is 53.37, 60.53, 79.27, 98.88, 118.53, 124.32, respectively. From Table 8 and Figure 8, it is evident that energy consumption of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 8. Energy consumption for DRFTSA using regular dataset by using fixed number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
100	77.68	81.79	91.85	53.37
200	82.45	85.35	89.32	60.53
300	91.38	94.17	97.65	79.27
400	103.67	113.57	120.32	98.88
500	125.72	130.93	134.16	118.53
600	134.12	142.47	152.82	124.32

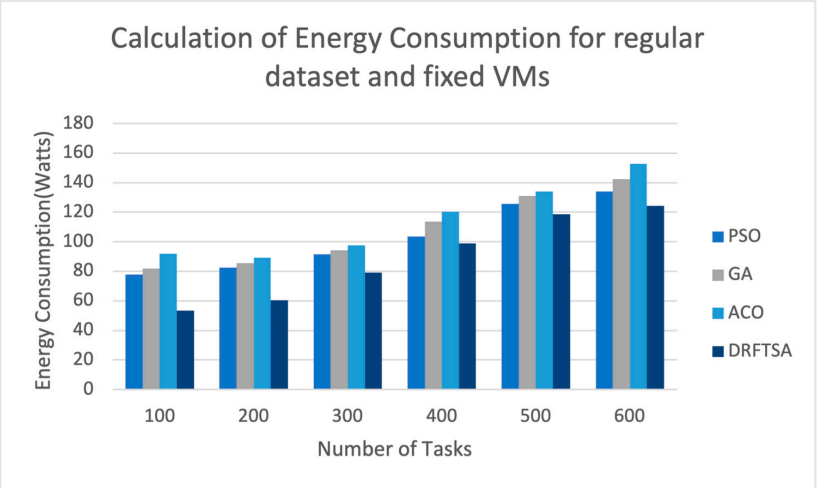


Figure 8. Calculation of energy consumption for regular dataset using fixed number of VMs.

5.7. Calculation of Energy Consumption for DRFTSA Using Regular Dataset by Varying Number of VMs

In this section, we present the next scenario of the generated energy consumption for DRFTSA with a regular dataset and by using various numbers of VMs. We considered VMs linearly by changing them and assigning them to various tasks, i.e., for 100, 200, 300, 400, 500, 600 tasks by 10, 20, 30, 40, 50, 60 VMs, respectively. The proposed DRFTSA was compared to existing PSO, GA, and ACO algorithms to check the efficacy of DRFTSA. Generated energy consumption for PSO with 100, 200, 300, 400, 500, 600 tasks is 63.25, 65.72, 68.16, 72.76, 77.95, 82.56, respectively. Generated energy consumption for GA with 100, 200, 300, 400, 500, 600 tasks is 67.03, 68.87, 71.46, 72.52, 76.88, 81.64, respectively.

Generated energy consumption for ACO with 100, 200, 300, 400, 500, 600 tasks is 69.47, 70.58, 73.84, 79.73, 82.81, 83.88, respectively. Generated energy consumption for DRFTSA with 100, 200, 300, 400, 500, 600 tasks is 54.99, 57.18, 62.18, 64.86, 66.71, 72.13, respectively. From Table 9 and Figure 9, it is evident that energy consumption of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 9. Energy consumption for DRFTSA using regular dataset by varying number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
100	63.25	67.03	69.47	54.99
200	65.72	68.87	70.58	57.18
300	68.16	71.46	73.84	62.18
400	72.76	72.52	79.73	64.86
500	77.95	76.88	82.81	66.71
600	82.56	81.64	83.88	72.13

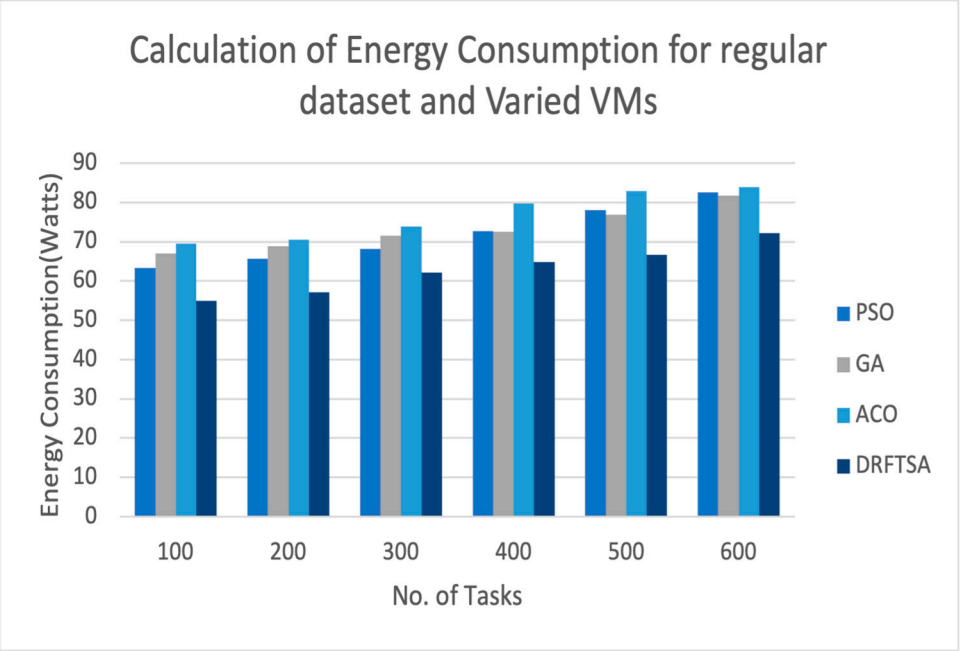


Figure 9. Calculation of energy consumption for regular dataset by varying number of VMs.

5.8. Calculation of Energy Consumption for DRFTSA Using Large Dataset by Fixed Number of VMs

In this section, we present the next scenario of generated energy consumption for DRFTSA with a large dataset and by using a fixed number of VMs. For the large dataset, we considered 700, 800, 900, 1000 tasks and a fixed number of VMs in this scenario. Generated energy consumption for PSO with 700, 800, 900, 1000 tasks is 82.56, 84.39, 93.57, 112.68, respectively. Generated energy consumption for GA with 700, 800, 900, 1000 tasks is 86.16,

87.99, 97.17, 108.99, respectively. Generated energy consumption for ACO with 700, 800, 900, 1000 tasks is 93.42, 97.19, 98.42, 134.77, respectively. Generated energy consumption for DRFTSA with 700, 800, 900, 1000 tasks is 48.89, 59.18, 68.74, 100.67, respectively.

From Table 10 and Figure 10, it is evident that energy consumption of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 10. Energy consumption for DRFTSA using large dataset using fixed number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
700	82.56	86.16	93.42	48.89
800	84.39	87.99	97.19	59.18
900	93.57	97.17	98.42	68.74
1000	112.68	108.99	134.77	100.67

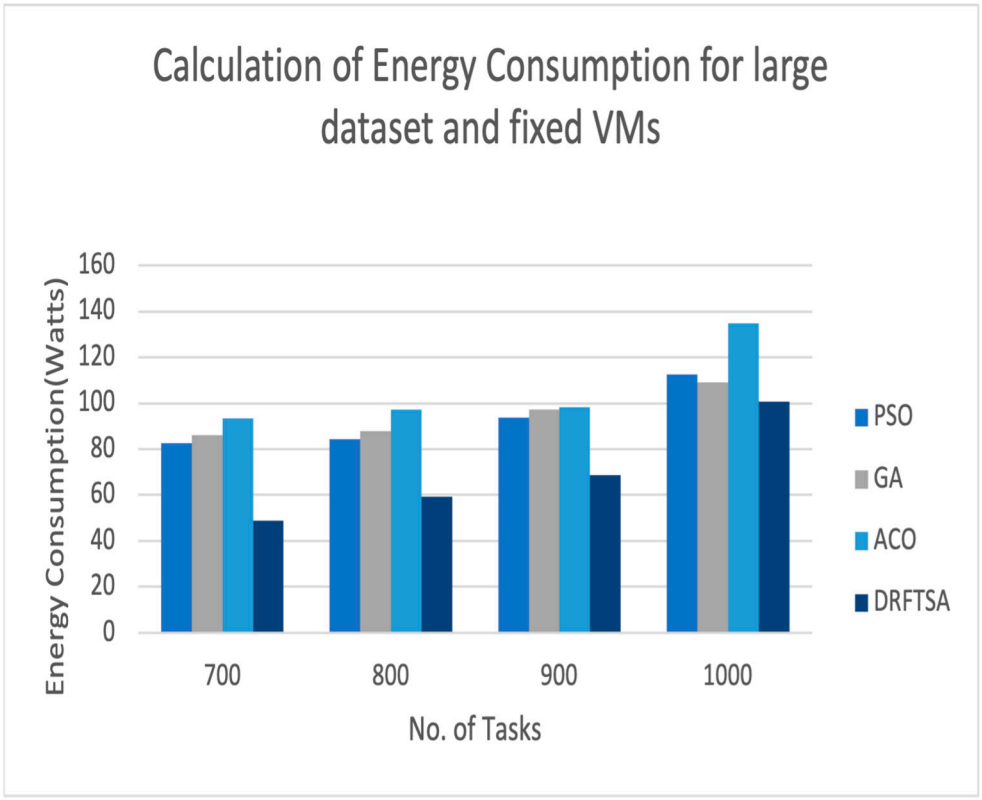


Figure 10. Calculation of energy consumption for large dataset by using fixed number of VMs.

5.9. Calculation of Energy Consumption for DRFTSA Using Large Dataset by Varying Number of VMs

In this section, we present the next scenario of generated energy consumption for DRFTSA with a large dataset and by varying the number of VMs. In this scenario, we linearly assigned different numbers of VMs, i.e., 50 VMs for 700 tasks, 60 VMs for 800 tasks, 70 VMs for 900 tasks, and 80 VMs for 1000 tasks. Generated energy consumption for PSO with 700, 800, 900, 1000 tasks is 82.17, 75.56, 84.26, 90.38, respectively. Generated energy consumption for GA with 700, 800, 900, 1000 tasks is 85.12, 87.03, 89.21, 92.77, respectively. Generated energy consumption for ACO with 700, 800, 900, 1000 tasks is 88.48, 82.66, 89.35, 94.17, respectively. Generated energy consumption for DRFTSA with 700, 800, 900, 1000 tasks is 56.18, 63.81, 71.14, 60.11, respectively.

From Table 11 and Figure 11, it is evident that energy consumption of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 11. Energy consumption for DRFTSA using large dataset by varying number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
700	82.17	85.12	88.48	56.18
800	75.56	87.03	82.66	63.81
900	84.26	89.21	89.35	71.14
1000	90.38	92.77	94.17	60.11

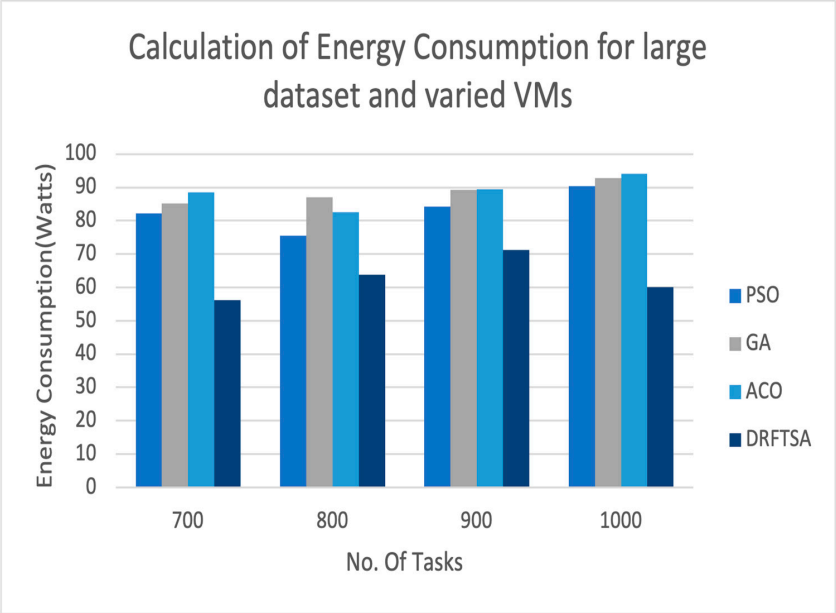


Figure 11. Calculation of energy consumption for large dataset by varying number of VMs.

5.10. Calculation of Rate of Failure for DRFTSA Using Regular Dataset Using Fixed Number of VMs

In this section, we precisely present the generated rate of failure for DRFTSA for a regular dataset using a fixed number of VMs. The reason to measure the rate of failure is that it is one of the parameters which is related to trust in the cloud provider and it is also used to identify the fault tolerance of the cloud provider as it impacts the quality of service. Therefore, we chose the rate of failure as one of the parameters and it is calculated by considering a regular dataset using a fixed number of VMs in this scenario. We considered 100, 200, 300, 400, 500, and 600 tasks. For all these tasks, VMs are considered as fixed. Generated rate of failure for PSO with 100, 200, 300, 400, 500, 600 tasks is 49.23, 53.57, 55.23, 57.17, 60.32, 63.98, respectively. Generated rate of failure for GA with 100, 200, 300, 400, 500, 600 tasks is 52.38, 56.18, 58.41, 62.59, 64.36, 67.16, respectively. Generated rate of failure for ACO with 100, 200, 300, 400, 500, 600 tasks is 58.31, 54.18, 49.53, 50.37, 63.18, 65.86, respectively. Generated rate of failure for DRFTSA with 100, 200, 300, 400, 500, 600 tasks is 32.18, 37.58, 34.21, 35.98, 33.27, 36.86, respectively.

From Table 12 and Figure 12 it is evident that rate of failures of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 12. Rate of failure for DRFTSA using regular dataset by using fixed number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
100	49.23	52.38	58.31	32.18
200	53.57	56.18	54.18	37.58
300	55.23	58.41	49.53	34.21
400	57.17	62.59	50.37	35.98
500	60.32	64.36	63.18	33.27
600	63.98	67.16	65.86	36.86

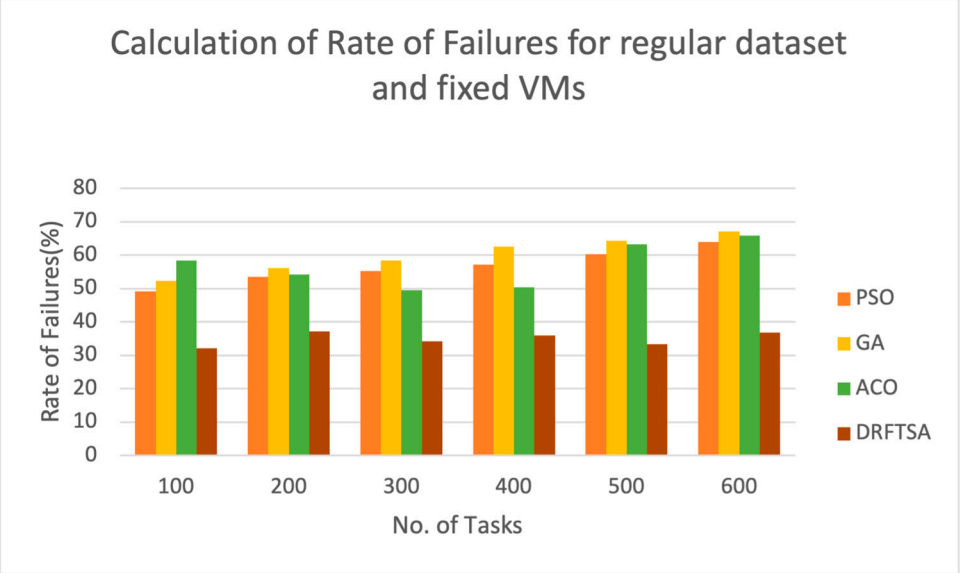


Figure 12. Calculation of rate of failure for regular dataset using fixed number of VMs.

5.11. Calculation of Rate of Failure for DRFTSA Using Regular Dataset by Varying Number of VMs

In this section, we precisely present the generated rate of failure for DRFTSA for a regular dataset by varying the number of VMs. VMs are varied over tasks linearly assigned to them, i.e., 100, 200, 300, 400, 500, 600 tasks assigned to 10, 20, 30, 40, 50, 60 VMs, respectively. Generated rate of failure for PSO for 100, 200, 300, 400, 500, 600 tasks is 45.17, 50.78, 56.97, 58.36, 59.45, 61.36, respectively. Generated rate of failure for GA for 100, 200, 300, 400, 500, 600 tasks is 47.36, 49.52, 54.57, 60.44, 62.74, 63.19, respectively. Generated rate of failure for ACO for 100, 200, 300, 400, 500, 600 tasks is 51.72, 52.47, 55.73, 59.18, 65.14, 67.74, respectively. Generated rate of failure for DRFTSA for 100, 200, 300, 400, 500, 600 tasks is 30.85, 28.78, 25.62, 24.64, 22.18, 20.04, respectively.

From Table 13 and Figure 13, it is evident that the rate of failure of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

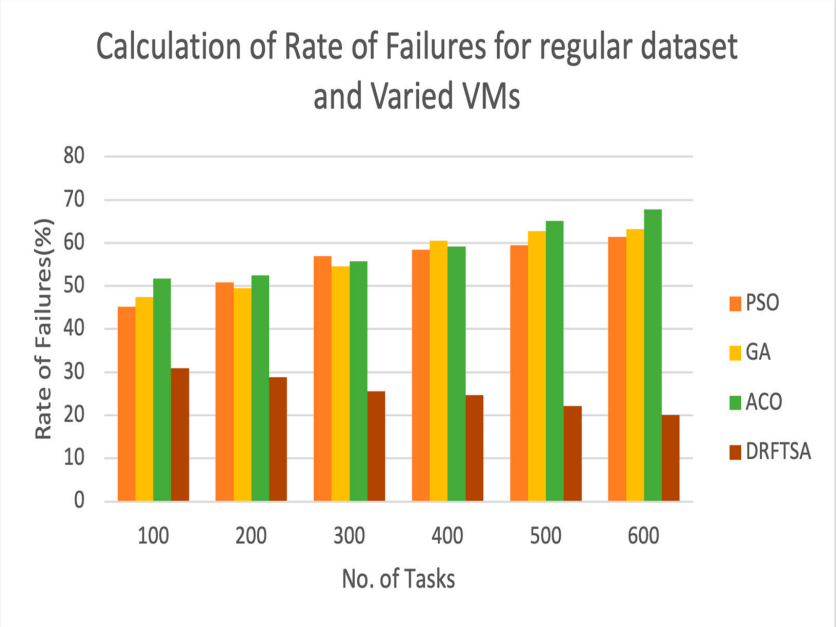


Figure 13. Calculation of rate of failure for regular dataset by varying number of VMs.

Table 13. Rate of failure for DRFTSA using regular dataset by varying number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
100	45.17	47.36	51.72	30.85
200	50.78	49.52	52.47	28.78
300	56.97	54.57	55.73	25.62
400	58.36	60.44	59.18	24.64
500	59.45	62.74	65.14	22.18
600	61.36	63.19	67.74	20.04

5.12. Calculation of Rate of Failure for DRFTSA Using Large Dataset by Using Fixed Number of VMs

In this section, we precisely present the generated rate of failure for DRFTSA for a large dataset by using a fixed number of VMs. The numbers of tasks considered in this dataset are 700, 800, 900, and 1000. Generated rate of failure for PSO with 700, 800, 900, 1000 tasks is 51.74, 56.47, 58.23, 62.16, respectively. Generated rate of failure for GA with 700, 800, 900, 1000 tasks is 53.02, 58.79, 63.19, 65.38, respectively. Generated rate of failure for ACO with 700, 800, 900, 1000 tasks is 54.28, 56.52, 62.04, 69.56, respectively. Generated rate of failure for DRFTSA with 700, 800, 900, 1000 tasks is 42.69, 35.82, 47.18, 52.18, respectively.

From Table 14 and Figure 14, it is evident that the rate of failure of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 14. Rate of failure for DRFTSA using large dataset by using fixed number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
700	51.74	53.02	54.28	42.69
800	56.47	58.79	56.52	35.82
900	58.23	63.19	62.04	47.18
1000	62.16	65.38	69.56	52.18

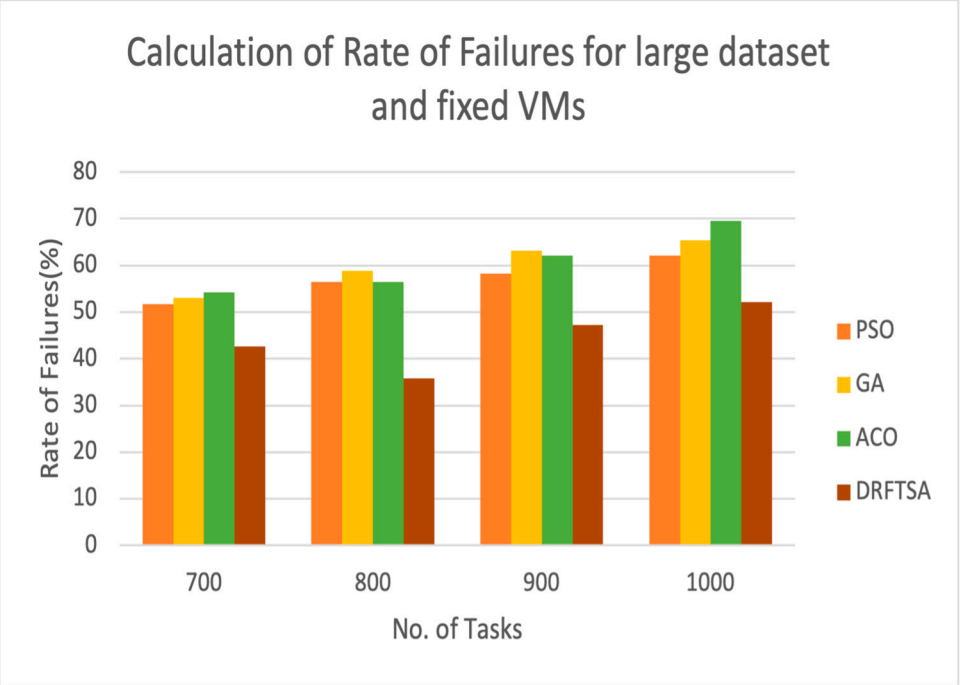


Figure 14. Calculation of rate of failure for large dataset by using fixed number of VMs.

5.13. Calculation of Rate of Failure for DRFTSA Using Large Dataset by Varying Number of VMs

In this section, we precisely present the generated rate of failure for DRFTSA for a large dataset by varying the number of VMs. The numbers of tasks considered in this dataset are 700, 800, 900, and 1000. To 700, 800, 900, and 1000 tasks are assigned 50, 60, 70, and 80 VMs, respectively. Generated rate of failure for PSO with 700, 800, 900, 1000 tasks is 56.24, 53.14, 58.19, 62.15, respectively. Generated rate of failure for GA with 700, 800, 900, 1000 tasks is 52.15, 58.06, 63.17, 65.87, respectively. Generated rate of failure for ACO with 700, 800, 900, 1000 tasks is 58.08, 61.43, 65.86, 71.36, respectively. Generated rate of failure for DRFTSA with 700, 800, 900, 1000 tasks is 42.14, 34.23, 26.87, 27.62, respectively.

From Table 15 and Figure 15, it is evident that the rate of failure of DRFTSA compared to various approaches is greatly minimized with the considered dataset and VMs.

Table 15. Rate of failure for DRFTSA using large dataset by varying number of VMs.

No. of Tasks	PSO	GA	ACO	DRFTSA
700	56.24	52.15	58.08	42.14
800	53.14	58.06	61.43	34.23
900	58.19	63.17	65.86	26.87
1000	62.15	65.87	71.36	27.62

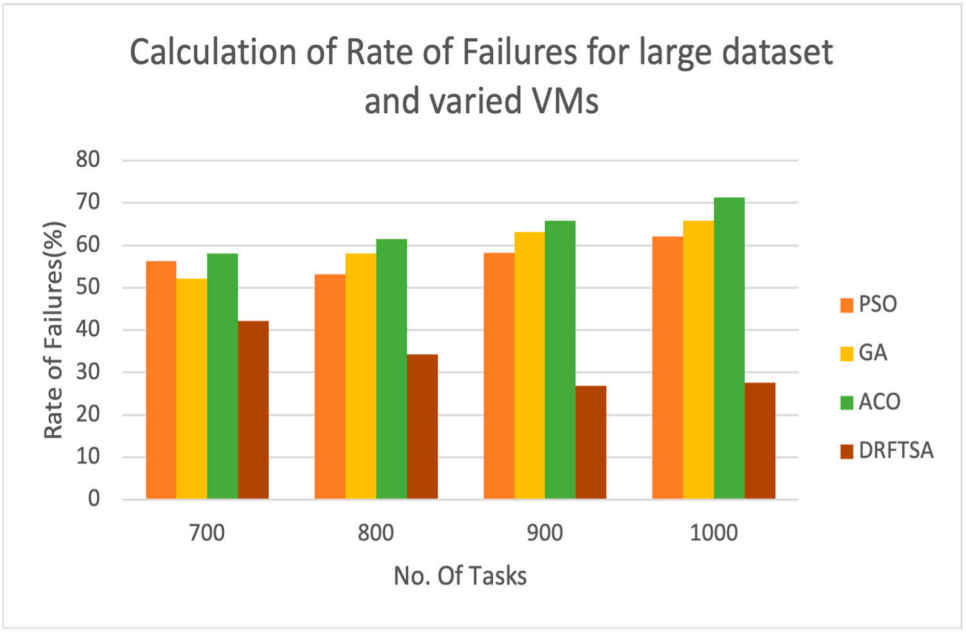


Figure 15. Calculation of rate of failure for large dataset by varying number of VMs.

5.14. Analysis and Discussion of Results

In this section, a detailed analysis and discussion of results which we presented earlier in Sections 5.2–5.13 are given. For the simulation of DRFTSA, we used Cloudsim [42] as the simulation platform. To evaluate our proposed DRFTSA, a real-time dataset was given as an input to the algorithm, i.e., the Google Cloud Jobs dataset from [43]. The entire dataset was categorized into two parts, i.e., regular and large datasets. After categorizing, different numbers of VMs for datasets are considered in different modes. For both regular and large datasets, we assigned VMs by fixing VMs in one scenario and varying VMs in another scenario. Extensive simulations were conducted by using the Google Cloud Jobs dataset, which is a real-time trace of a Google cluster. These VMs are varied linearly over the number of tasks in the dataset. The length of the tasks was 750,000 and when we used priorities for tasks and VMs and ran the simulation we observed that the proposed DRFTSA outperforms the existing approaches. DRFTSA was evaluated compared to existing algorithms, i.e.,

PSO, GA, and ACO approaches. When we compared the proposed DRFTSA against PSO, GA, and ACO, we observed that when VMs are fixed the makespan is slightly improved because there is fixed availability of VMs and even if we the increase workload on VMs that does not cause a significant change in makespan. However, when we vary the number of VMs, makespan improves compared with fixed VMs. Table 16 presents the improvement in average makespan of DRFTSA for regular and large datasets over existing algorithms. For the energy consumption, when we look at Table 17 it is clearly seen that for fixed number of VMs, energy consumption is not improved drastically but, when we varied VMs, energy consumption is improved compared to a fixed number of VMs. Table 17 presents the improvement in average energy consumption of DRFTSA for regular and large datasets over existing algorithms. Improvement of the rate of failure is observed in Table 18 as, if we fix VMs and keep on increasing load, a chance of failure occurs due to unavailable resources or unsuitable resources. However, when changing VMs, we observed that there is a clear improvement in the rate of failure, i.e., failures are greatly reduced. Table 18 presents the improvement in the average rate of failure of DRFTSA for regular and large datasets over existing algorithms.

Table 16. Improvement in average makespan of DRFTSA for regular and large datasets over state-of-the-art approaches.

Dataset	PSO	GA	ACO
Regular (fixed VMs)	38.13%	42.07%	40.16%
Regular (varied VMs)	33.37%	37.54%	43.87%
Large (fixed VMs)	23.02%	24.96%	25.73%
Large (varied VMs)	29.34%	35.81%	38.72%

Table 17. Improvement in average energy consumption of DRFTSA for regular and large datasets over state-of-the-art approaches.

Dataset	PSO	GA	ACO
Regular (fixed VMs)	13.02%	17.48%	22.03%
Regular (varied VMs)	12.16%	18.72%	25.87%
Large (fixed VMs)	25.64%	27.03%	30.52%
Large (varied VMs)	24.4%	29.05%	36.78%

Table 18. Improvement in average rate of failure of DRFTSA for regular and large datasets over state-of-the-art approaches.

Dataset	PSO	GA	ACO
Regular (fixed VMs)	38.23%	41.92%	38.58%
Regular (varied VMs)	54.19%	60.36%	66.43%
Large (fixed VMs)	22.19%	26%	26.62%
Large (varied VMs)	43.03%	48.24%	53.13%

6. Conclusions and Future Works

Cloud computing poses a challenge to cloud service providers in terms of task scheduling as a huge number of diversified and heterogeneous tasks are generated from different cloud users and all these tasks need to be scheduled on to the specified virtual machines while minimizing the rate of failure as single points of failure occur in the cloud paradigm. Therefore, to tackle the rate of failure we proposed a fault-tolerant aware task scheduling algorithm (DRFTSA) by considering the priorities of tasks and VMs based on unit electricity cost which precisely schedules tasks to VMs. A deep-reinforcement-learning-based model, i.e., DQN, is used to model our task scheduler. This model is integrated and fed to the scheduler to dynamically make decisions and generate schedules based on the input given to the scheduler. The workload used in this research is a real-time dataset, i.e., Google

Cloud Jobs. The entire dataset is divided into two categories, i.e., regular and large datasets. These two categories of workload are given as an input to the algorithm for checking the efficacy of the scheduler. Extensive simulations are conducted on Cloudsim by considering VMs in different scenarios of fixed and varied VMs. For evaluating the efficacy of the proposed DRFTSA scheduler, we compared our proposed approach with the existing state-of-the-art algorithms, i.e., PSO, GA, and ACO. From the results, it is evident that the proposed DRFTSA shows a huge impact compared to the existing approaches by improving average makespan, rate of failure, and energy consumption for both scenarios, i.e., for fixed VMs and varied VMs. The main limitation in this research is that the scheduler not able to predict the upcoming tasks on the cloud application console. In future, we need to examine our DQN model to evaluate parameters i.e., total power cost, migration time, and trust-based parameters, and to evaluate the relationship between fault tolerance and trust.

Author Contributions: M.S.R.K.—writing the draft, simulating results; S.M.—conceptualization, validating results. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All the data regarding research is to be provided based on the request. The data are not publicly available due to we have evaluated our metrics by using prioritized scheduler in our work and how it is programized and all can not be disclosed directlye.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, X. An IFWA-BSA Based Approach for Task Scheduling in Cloud Computing. *J. ICT Stand.* **2023**, *11*, 45–66. [CrossRef]
2. Wang, Y.; Tao, X.; Zhao, F.; Tian, B.; Sai, A.M.V.V. SLA-aware resource scheduling algorithm for cloud storage. *EURASIP J. Wirel. Commun. Netw.* **2020**, *2020*, 6. [CrossRef]
3. Mangalampalli, S.; Ganesh Reddy, K.; Utku, K. Multi Objective Trust aware task scheduling algorithm in cloud computing using Whale Optimization. *J. King Saud Univ. Comput. Inf. Sci.* **2023**, *35*, 791–809. [CrossRef]
4. Nabi, S.; Ahmad, M.; Ibrahim, M.; Hamam, H. AdPSO: Adaptive PSO-based task scheduling approach for cloud computing. *Sensors* **2022**, *22*, 920. [CrossRef]
5. Malathi, K.; Priyadarsini, K. Hybrid lion–GA optimization algorithm-based task scheduling approach in cloud computing. *Appl. Nanosci.* **2003**, *13*, 2601–2610. [CrossRef]
6. Elsedimy, E.; Algarni, F. MOTS-ACO: An improved ant colony optimiser for multi-objective task scheduling optimisation problem in cloud data centres. *IET Netw.* **2022**, *11*, 43–57. [CrossRef]
7. Siddesha, K.; Jayaramaiah, G.V.; Singh, C. A novel deep reinforcement learning scheme for task scheduling in cloud computing. *Clust. Comput.* **2022**, *25*, 4171–4188. [CrossRef]
8. Tong, Z.; Deng, X.; Chen, H.; Mei, J.; Liu, H. QL-HEFT: A novel machine learning scheduling scheme base on cloud computing environment. *Neural Comput. Appl.* **2020**, *32*, 5553–5570. [CrossRef]
9. Rjoub, G.; Bentahar, J.; Wahab, O.A.; Bataineh, A.S. Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e5919. [CrossRef]
10. Islam, M.T.; Karunasekera, S.; Buyya, R. Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 1695–1710. [CrossRef]
11. Zhou, C.; Wu, W.; He, H.; Yang, P.; Lyu, F.; Cheng, N.; Shen, X. Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN. *IEEE Trans. Wirel. Commun.* **2020**, *20*, 911–925. [CrossRef]
12. Kruekaew, B.; Kimpan, W. Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. *IEEE Access* **2022**, *10*, 17803–17818. [CrossRef]
13. Sanaj, M.; Prathap, P.J. An efficient approach to the map-reduce framework and genetic algorithm-based whale optimization algorithm for task scheduling in cloud computing environment. *Mater. Today Proc.* **2021**, *37*, 3199–3208. [CrossRef]
14. Mahmoud, H.; Thabet, M.; Khafagy, M.H.; Omara, F.A. Multiobjective task scheduling in cloud environment using decision tree algorithm. *IEEE Access* **2022**, *10*, 36140–36151. [CrossRef]
15. Cheng, M.; Ji, L.; Shahin, N. DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, Jeju, Republic of Korea, 22–25 January 2018.

16. Li, Z.; Guan, P.; Dong, Y.; Yang, Y. Multi-objective optimization of cloud task scheduling based on improved deep Q-learning algorithm. *arXiv* **2023**. [CrossRef]
17. Li, K.; Peng, Z.; Cui, D.; Li, Q. Sla-DQTS: SLA Constrained Adaptive Online task scheduling based on DDQN in cloud computing. *Appl. Sci.* **2021**, *11*, 9360. [CrossRef]
18. Riera, J.F.; Batallé, J.; Bonnet, J.; Dias, M.; McGrath, M.; Petralia, G.; Liberati, F.; Giuseppi, A.; Pietrabissa, A.; Ceselli, A.; et al. TeNOR: Steps towards an orchestration platform for multi-PoP NFV deployment. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Republic of Korea, 6–10 June 2016.
19. Soualhia, M.; Khomh, F.; Tahar, S. A dynamic and failure-aware task scheduling framework for hadoop. *IEEE Trans. Cloud Comput.* **2018**, *8*, 553–569. [CrossRef]
20. Peng, Z.; Cui, D.; Zuo, J.; Li, Q.; Xu, B.; Lin, W. Random task scheduling scheme based on reinforcement learning in cloud computing. *Clust. Comput.* **2015**, *18*, 1595–1607. [CrossRef]
21. Chen, J.; Han, P.; Liu, Y.; Du, X. Scheduling independent tasks in cloud environment based on modified differential evolution. *Concurr. Comput. Pract. Exp.* **2023**, *35*, e6256. [CrossRef]
22. Zhou, Z.; Chang, J.; Hu, Z.; Yu, J.; Li, F. A modified PSO algorithm for task scheduling optimization in cloud computing. *Concurr. Comput. Pract. Exp.* **2018**, *30*, e4970. [CrossRef]
23. Khodar, A.; Chernenkaya, L.V.; Alkhayat, I.; Al-Afare, H.A.F.; Desyatirikova, E.N. Design model to improve task scheduling in cloud computing based on particle swarm optimization. In Proceedings of the 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg and Moscow, Russia, 27–30 January 2020.
24. Narendrababu Reddy, G.; Phani Kumar, S. Regressive whale optimization for workflow scheduling in cloud computing. *Int. J. Comput. Intell. Appl.* **2019**, *18*, 1950024. [CrossRef]
25. Al-Zoubi, H. Efficient task scheduling for applications on clouds. In Proceedings of the 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Paris, France, 21–23 June 2019.
26. Hall, J.; Moessner, K.; Mackenzie, R.; Carrez, F.; Foh, C.H. Dynamic scheduler management using deep learning. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *6*, 575–585. [CrossRef]
27. Chen, X.; Dan, L. Task scheduling of cloud computing using integrated particle swarm algorithm and ant colony algorithm. *Clust. Comput.* **2019**, *22*, 2761–2769. [CrossRef]
28. Zhao, J.; Rodriguez, M.A.; Buyya, R. A deep reinforcement learning approach to resource management in hybrid clouds harnessing renewable energy and task scheduling. In Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 5–10 September 2021.
29. Xiu, X.; Li, J.; Long, Y.; Wu, W. MRLCC: An adaptive cloud task scheduling method based on meta reinforcement learning. *J. Cloud Comput.* **2023**, *12*, 75. [CrossRef]
30. Dong, T.; Xue, F.; Xiao, C.; Li, J. Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5654. [CrossRef]
31. Dong, T.; Xue, F.; Tang, H.; Xiao, C. Deep reinforcement learning for fault-tolerant workflow scheduling in cloud environment. *Appl. Intell.* **2023**, *53*, 9916–9932. [CrossRef]
32. Jin, H.; Fu, Y.; Yang, G.; Zhu, X. An intelligent scheduling algorithm for resource management of cloud platform. *Multimed. Tools Appl.* **2020**, *79*, 5335–5353. [CrossRef]
33. Pradhan, A.; Bisoy, S.K.; Kautish, S.; Jasser, M.B.; Mohamed, A.W. Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment. *IEEE Access* **2022**, *10*, 76939–76952. [CrossRef]
34. Wang, Y.; Liu, H.; Zheng, W.; Xia, Y.; Li, Y.; Chen, P.; Guo, K.; Xie, H. Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning. *IEEE Access* **2019**, *7*, 39974–39982. [CrossRef]
35. Wei, Y.; Kudenko, D.; Liu, S.; Pan, L.; Wu, L.; Meng, X. A Reinforcement Learning Based Workflow Application Scheduling Approach in Dynamic Cloud Environment. In *Collaborative Computing: Networking, Applications and Worksharing: 13th International Conference, CollaborateCom 2017, Edinburgh, UK, 11–13 December 2017, Proceedings 13*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018.
36. Sun, C.; Tan, Y.; Youxun, L. DDDQN-TS: A task scheduling and load balancing method based on optimized deep reinforcement learning in heterogeneous computing environment. *Int. J. Intell. Syst.* **2022**, *37*, 9138–9172. [CrossRef]
37. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [CrossRef]
38. Hussain, A.; Aleem, M. *Mendeley Data*, version 1. GoCJ: Google Cloud Jobs Dataset. Capital University of Science and Technology: Islamabad, Pakistan, 2018. [CrossRef]
39. Madni, S.H.H.; Abd Latiff, M.S.; Abdulhamid, S.I.M.; Ali, J. Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. *Clust. Comput.* **2019**, *22*, 301–334. [CrossRef]
40. Spano, S.; Cardarilli, G.C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Matta, M.; Nannarelli, A.; Re, M. An efficient hardware implementation of reinforcement learning: The q-learning algorithm. *IEEE Access* **2019**, *7*, 186340–186351. [CrossRef]
41. Mokhtari, A.; Pooyan, J.; Salehi, M.A. FELARE: Fair scheduling of machine learning applications on heterogeneous edge systems. *arXiv* **2022**, arXiv:2206.00065.

42. Neelakantan, P.; Yadav, N.S. Proficient job scheduling in cloud computation using an optimized machine learning strategy. *Int. J. Inf. Technol.* **2023**, *15*, 2409–2421. [CrossRef]
43. Pattanaik, B.C.; Sahoo, B.K.; Pati, B.; Laha, S.R. Dynamic Fault Tolerance Management Algorithm for VM Migration in Cloud Data Centers. *Int. J. Intell. Syst. Appl. Eng.* **2023**, *11*, 85–96.
44. Siyadatzadeh, R.; Mehrafrooz, F.; Ansari, M.; Safaei, B.; Shafique, M.; Henkel, J.; Ejlali, A. ReLIEF: A Reinforcement Learning-Based Real-Time Task Assignment Strategy in Emerging Fault-Tolerant Fog Computing. *IEEE Internet Things J.* **2023**, *10*, 10752–10763. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.