Indonesian Journal of Electrical Engineering and Computer Science
Vol. 33, No. 2, February 2024, pp. 1150~1168
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v33.i2.pp1150-1168

Enhancing fault tolerance: dual Q-learning with dynamic scheduling

Chetankumar Kalaskar, Thangam Somasundaram
Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetam, Bangalore, India

Article Info

Article history:

Received Oct 7, 2023
Revised Nov 4, 2023
Accepted Dec 8, 2023

Keywords:

Adaptive system
Cloud computing
Double deep Q-learning
Dynamic scheduling algorithm
Fault tolerance
Real-time systems

ABSTRACT

Cloud computing has revolutionized IT delivery by offering scalable on-demand internet services encompassing software, platforms, and infrastructure. However, cloud services face significant performance challenges due to their susceptibility to failures given their vast operational scale. Implementing fault tolerance in dynamic cloud services is a key challenge, with complex configurations and dependencies complicating deployment. This paper introduces an innovative approach that combines double deep Q-learning (DDQL) with a dynamic fault-tolerant real-time scheduling algorithm (DFTRTSA) to enhance fault tolerance in real-time systems. DDQL, an extension of deep Q-learning, optimizes the fault-tolerance decision-making process. The algorithm adjusts scheduling strategies dynamically based on system conditions and errors. The fusion of DDQL and DFTRTSA aims to create a resilient and adaptive fault-tolerant mechanism, ensuring uninterrupted operation while meeting real-time requirements. This adaptive approach efficiently manages resources, meets deadlines, and gracefully handles errors, as demonstrated through experiments. Our DDQL-DFTRTSA method outperforms conventional fault-tolerant mechanisms in defect tolerance, energy efficiency, downtime reduction, and system dependability. It proves to be an ideal solution for real-time systems in dynamic and unpredictable environments.

Corresponding Author:

Chetankumar Kalaskar
Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetam, Bangalore, India
Email: k_chetankumar@blr.amrita.edu

1. INTRODUCTION

Due to the popularity of cloud computing, there is an increasing demand for immediate allocation of computing resources to serve applications that are dynamic [1], [2]. Scalability and cost-effectiveness can be achieved by running programs on virtual resources, particularly virtual computers [3]. Cloud computing can efficiently address the needs of high-performance computing for specific scientific applications. The difficulties faced by cloud computing have been exacerbated by the large-scale systems' increasing complexity, with resource failure emerging as a significant worry. There will reportedly be one daily failure for a cloud of 10,000 servers with an extraordinarily long mean time between failures (MTBF) of 30 years. Furthermore, approximately 5% of disk drives fail yearly, and servers crash at least twice. Worse, data centres usually use low-cost commodity technology due to low production costs, increasing the likelihood of resource failure. Fault tolerance must, therefore, be offered in the cloud, especially for real-time applications. The scheduling aspect of fault tolerance is most important for real-time applications.

Fault tolerance scheduling links tasks to computing instances to ensure that activities are on time even when hardware and software fail. Replication and resubmission are two fundamental scheduling algorithms currently employed extensively in handling failures in distribution systems. The flexibility and complexity of scheduling are increased by the cloud's distinctive qualities, which set it apart from conventional distributed systems. There are generally three fundamental difficulties: (1) multiple computing instances (such as virtual machines (VMs)) will fail if the host crashes; (2) task operation times are very variable, and (3) the trade-off between replication and resubmission should be taken into account to reduce resource consumption while maintaining task reliability.

To increase the fault tolerance of real-time systems, this paper examines the combination of two complex techniques, double deep Q-learning (DDQL) and dynamic fault-tolerant real-time scheduling algorithm (DFTRTSA). DDQL is a reinforcement learning approach with extraordinary performance in tackling complicated decision-making problems. DFTRTSA, on the other hand, is a real-time scheduling algorithm that adapts dynamically to changing fault circumstances in the system. The fundamental goal of this research is to investigate the synergistic effects of integrating DDQL with DFTRTSA in real-time systems. This paper intends to design a novel approach that not only identifies and mitigates faults but also optimizes task scheduling to maintain system performance and reliability under fault situations by leveraging the learning capabilities of DDQL and the adaptability of DFTRTSA.

This study will delve into the theoretical foundations of DDQL and DFTRTSA, explain their respective contributions to fault tolerance, and show how their combination can lead to a more robust and trustworthy real-time system. Furthermore, we will offer experimental data and case studies to demonstrate our proposed approach's practical applicability and effectiveness. The ultimate goal is to give insights and tools to assist engineers and researchers in building and implementing highly fault-tolerant real-time systems, assuring the continuing operation of essential applications in the face of adversity. The essential contributions of this work are summarized as follows:

– To develop a novel approach for enhancing fault tolerance in real-time systems, we combine DDQL with a DFTRTSA. DDQL, an extension of DQL, serves to model and optimize the decision-making process of the fault-tolerance mechanism.
– Our approach dynamically adapts the scheduling strategy based on the operational state of the real-time system and the occurrence of errors. The fusion of DDQL and DFTRTSA aims to establish a robust and adaptive fault-tolerant mechanism that improves the system's resilience to failures while meeting real-time performance requirements.
– This adaptive approach empowers the system to efficiently manage resources, meet deadlines, and gracefully handle errors and disruptions, ensuring uninterrupted operation in the face of adversity.
– This research presents experimental evidence showcasing the superiority of our DDQL-DFTRTSA method over conventional fault-tolerant mechanisms. The results illustrate the proposed algorithm's exceptional performance in defect tolerance, energy efficiency, downtime reduction, and system dependability, making it an ideal solution for real-time systems operating in dynamic and unpredictable environments.
– Experimental results demonstrate that DDQP not only surpasses the state-of-the-art method by 95.66% and achieves a 97.44% higher acceptance ratio under the guarantee ratio and task acceptance ratio.

According to Jin and Zhang [4], the cost-benefit analysis support platform developed for cloud computing accounting services focuses primarily on analysing the intermediate data fault tolerance technique. The development of an organizational-specific cloud computing accounting service platform is the primary goal of this project. A thorough cost-benefit analysis of business transactions is carried out on this platform. Compared to already used application approaches, the research uses an enterprise's transaction cost within the accounting service platform to validate and evaluate its effectiveness. With a noteworthy data availability probability of 0.98, this approach significantly displays robust data availability even after fault-tolerant processing. This level of dependability guarantees that users will successfully engage with the platform.

Ahmad *et al.* [5] presented cluster-based, fault-tolerant, data-intensive (CFD) scheduling for cloud-based scientific applications. Due to the data-intensive nature of scientific operations, CFD leverages cluster-based, fault-tolerant technology to address this problem. The minimum completion time (MCT), max-min, and min-min heuristic scheduling policies were compared to the CFD approach. The scientific method used by montage is a simulation. Simulations demonstrate that compared to the other three strategies, CFD reduced make-span by 14.28%, 20.37%, and 11.77%. The CFD reduces execution costs by 1.27%, 5.3%, and 2.2% compared to the other three strategies. Current restrictions frequently violate the Service level agreement (SLA), but due to time and money limits, the CFD technique does not.

The proactively coordinated fault tolerance (PCFT) solution introduced by Liu *et al.* [6] takes into account VM coordination for concurrent application execution in the cloud while reducing network resource demand and energy consumption. PCFT, however, is only suitable for parallel applications and is ineffective for processes. To put it another way, the check pointing system can manage momentary resource failures but not high pass filter (HPF) or other long-term resource failures. Yao *et al.* [7] proposed the imbalance characteristic for fault-tolerant workflow scheduling (ICFWS) technique, which breaks the overall workflow deadline into several sub-deadlines at the sub-task level. Individual sub-tasks inside scientific workflows then choose appropriate fault-tolerant solutions from a pool of redundancy and rescheduling options driven by these previously specified sub-deadlines. Spot instances are the VMs supplied under the dynamic pricing mechanism cloud computing companies use. Spot instances are more affordable when compared to the VMs offered under the static pricing plan.

Yuan *et al.* [8] performed to optimize the goals. However, there are a few things that could be better. They would squander many resources and be unable to provide an entirely reliable service. Additionally, its performance could be better due to its inability to handle real-time demands. A practical solution is urgently needed for concurrent and flexible scheduling of active and standby instances. Long *et al.* [9] unique fault-tolerant scheduling method is explicitly designed for cooperative edge-IoT operations. This method starts with a thorough examination of work distribution based on dependencies. It then uses a primary/backup (PB) technique to deal with job failures at the edge nodes. The approach also uses a deep Q-learning algorithm to identify the ideal work scheduling strategy. This study uses thorough simulative case studies that include a variety of randomly generated workflows and real records describing the locations of edge-internet of things (IoT) servers. The outcomes show that our suggested solution outperforms even the most technologically sophisticated rivals regarding essential criteria like job completion ratio, server active time, and resource consumption.

Zhang *et al.* [10] proposed the online fault detection algorithm support vector machines (SVM) grid. This process is critical for cloud stability. The author suggests using different failure detection models to understand the cloud system's fundamental structure. Traditional SVM models are the most popular. However, they need higher accuracy. SVM-grid-based online fault detection has been presented to address this issue. SVM-grid predicts cloud difficulties. Grid technique fine-tuned the model's input parameter for better prediction accuracy. We have also created a refined prediction algorithm and an improved FT approach designed for sample databases to improve fault prediction accuracy while simultaneously lowering time-related costs. Google2 application cluster datasets were used for simulations. The proposed method was compared to back propagation, learning vector quantization (LVQ), and standard SVM. The experimental findings demonstrated that the innovative model performed better in accuracy and time than BP, LVQ, and standard SVM.

Rehman *et al.* [11] substantial foundation on cloud computing supports readers in obtaining a thorough understanding of the problem from beginner to expert level. Before developing cloud computing fault-tolerance requirements and applications, focus on fault-tolerance components and system-level measurements. Discuss modern proactive and reactive fault-tolerance strategies for cloud computing. This paper organizes and discusses fault-tolerance ideas and frameworks for cloud computing. This lecture discusses potential future directions for cloud computing fault-tolerance research.

Nirmala *et al.* [12] proposed employing replication strategies based on an unsupervised model in conjunction with weight-synchronized checkpointing to improve process completion effectiveness. Unfortunately, previous research fell short in addressing the fault tolerance dilemma because it needed to consider the unique aspects of the edge-IoT environment while orchestrating operations. One significant difference between the earlier studies and ours is that they should have been regarded as proximity restrictions in edge-IoT scenarios. In our study, we provide a novel fault-tolerant scheduling algorithm (FTAW) created exclusively for cooperative edge-IoT operations. With the DQN method, task overlap reduction, and PB-based replication, we aim to increase workflow scheduling's fault tolerance for potential edge server failures.

The fault-tolerant adaptive scheduling mechanism with dynamic QoS-awareness (FASDQ), developed by Wang *et al.* [13], is a fault-tolerant adaptive scheduling system with dynamic quality of service awareness that extends the PB paradigm. To reduce latency and ensure dependable service for tasks, the approach adjusts the execution periods of the task copies. A container resource-adaptive adjustment technique is also suggested by this work, which modifies resource scheduling when available resources are insufficient to fulfil task copy requirements. To assess the performance differences between FASDQ and other approaches, this study presents the findings of simulation tests performed on the EdgeCloudSim platform. The results show that this mechanism performs better than competing techniques in dependability and total resource consumption while also speeding up the execution of duplicated activities.

Zhang *et al.* [14] efficient priority and relative distance (EPRD) algorithm is proposed as a means to reduce task scheduling length without violating the end-to-end deadline limitation for precedence-constrained workflow applications. There are two processes in this algorithm. A task priority queue is created first. The suggested method optimizes resource allocation by assigning a VM to a job based on its relative distance, improving scheduling and VM utilization performance. The resource reduction rate and scheduling length of the EPRD method significantly outperform those of earlier algorithms, according to several extensive experiments utilizing both randomly generated and real-world workflow applications. Poola *et al.* [15] thorough overview of the fault-tolerant methods used in various workflow management systems (WFMS) is available online. They also comprehensively taxonomy the different fault tolerance approaches used in distributed contexts. The study also examines various measures for calculating fault tolerance.

Ding *et al.* [16] the authors propose a fault-tolerant elastic scheduling algorithm designed to efficiently manage workflows in cloud systems. Workflow scheduling is a crucial task in cloud computing, where multiple tasks or jobs need to be executed on a distributed set of resources. Ensuring fault tolerance is especially important in cloud environments to maintain the reliability and availability of services.
Yan *et al.* [17] "dynamic fault-tolerant elastic scheduling for tasks (DEFT) with uncertain runtime in cloud," focuses on addressing the challenges posed by tasks with uncertain runtime in cloud computing environments. The authors introduce a novel approach called DEFT, which stands for DEFT. This approach aims to optimize the allocation of cloud resources for tasks with varying and uncertain execution times.

Wang *et al.* [18] the authors introduce a novel approach called DDQP, which stands for DDQL to tackle the challenges associated with online SFC placement. This technique leverages reinforcement learning and deep learning principles to make intelligent decisions regarding the placement of service functions in a network while ensuring fault tolerance. By utilizing a DDQL framework, the approach aims to optimize SFC placement dynamically in response to changing network conditions and service requirements.

Thangam *et al.* [19] propose architecture for service selection based on consumer feedback in a service-oriented environment. Panwar and Supriya [20] introduce dynamic resource provisioning for cloud applications through bayesian learning. Prakash *et al.* [21] explore smart city video surveillance using fog computing, while Prakash *et al.* [22] delve into the issues and future directions of fog computing. Singh *et al.* [23] present a hardware setup for vehicle-to-vehicle communication under foggy conditions. Deepika and Prakash [24] focus on power consumption prediction in cloud data centers using machine learning techniques. Sandeep and Thangam [25] propose a hybrid cloud approach for efficient data storage and security. Iyer [26] provides a comprehensive study on evolutionary games in cloud, fog, and edge computing. These studies collectively contribute to a deeper understanding of modern computing paradigms and their applications.

– Real-time systems frequently function in resource-constrained contexts, such as entrenched systems or IoT devices. Improving fault tolerance can be complex when processor power, memory, and energy usage are limited.
– Fault tolerance methods frequently include expectations about the types of failures that must be addressed. If these assumptions do not match the actual fault characteristics of the system, the fault tolerance approach's practicality can be at risk.
– As the system evolves or new defects are detected, fault tolerance methods may need to be updated or modified. It can be difficult to manage these updates without interfering with system operation.
– In embedded systems, achieving fault tolerance frequently necessitates trade-offs with other system attributes such as power consumption, size, and weight. Selecting the best compromises for an exact application can be a challenging problem.
– Real-time systems must be deterministic to satisfy rigorous timing deadlines. Using fault tolerance events may result in non-deterministic behaviour, making it difficult to ensure timely responses.

In addressing the identified challenges in fault tolerance within real-time systems, our work endeavors to present a comprehensive solution. Our proposed approach incorporates double deep Q-learning (DDQL) alongside a dynamic scheduling algorithm to fortify fault tolerance:

– Real-time systems have severe timing limitations, with tasks having to be completed within predefined timeframes. Using fault tolerance methods such as redundancy and error recovery should not violate these temporal requirements. Complementary fault tolerance with low-latency execution is a problematic task.
– Many real-time systems run on resource-limited platforms, such as embedded systems or IoT plans. Implementing fault tolerance frequently necessitates using extra resources, such as redundant components or specialized technology, which might be challenging given the limited resources available.
– The scale of real-time systems varies from minor embedded devices to large-scale distributed systems. Creating fault tolerance systems that are scalable and applicable across various system sizes and complexities is a severe challenge.
– Real-time systems frequently operate in dynamic and unpredictable environments, with the system's requirements and attributes changing over time. Fault tolerance techniques should be able to adapt to these changes while still providing adequate protection.
– Ensuring fault tolerance techniques are successful in real-time systems might be challenging. Complete testing and validation procedures are required to ensure fault tolerance mechanisms function correctly without risking system performance or safety.

Section 2 serves as an introduction to our proposed approach, providing a comprehensive overview of the methodologies and frameworks adopted to bolster fault tolerance. This section delves into the intricacies of the strategy, highlighting its core principles and how it aims to enhance system reliability. In Section 3, we detail the experimental setup, offering a thorough explanation of the parameters, scenarios, and data collection methods employed to validate our approach. Here, we present the findings, drawing connections between the experimental results and the efficacy of the proposed model. Lastly, Section 4 encapsulates the study, summarizing the key takeaways, implications, and potential avenues for future research. This conclusive section ties together the insights gleaned, reinforcing the significance of the proposed approach in the domain of fault-tolerant systems.

2. PROPOSED SYSTEM

This section explores the usage of DDQL and a DFTRTSA. DQL models and optimizes the fault-tolerance mechanism's decision-making process. The suggested technique dynamically adapts the scheduling strategy depending on real-time system operation and failures. DDQL and DFTRTSA produce a robust and adaptive fault-tolerant mechanism that improves system resilience and real-time performance. This adaptive method helps the system manage resources, fulfil deadlines, and gracefully handle mistakes and disruptions, assuring uninterrupted functioning in adversity. Additionally, we provide experimental proof that our DDQL-DFTRTSA technique outperforms standard fault-tolerant systems.

2.1. Architecture of the system

A schematic representation of our approach is presented in Figure 1. The data centre's hosts can each offer several VMs or computer instances. Users' task flows have been queued and are ready to be sent to the data centre's computing instances. A work scheduler plus a performance monitor makes up the system scheduler. On the monitor, the system's performance is currently displayed. The task scheduler DDQL assigns tasks depending on input from the presentation monitor. This system uses a star architecture for communication between the performance monitor, task scheduler and data centre. Without sacrificing generality, the data center connectivity delay can be ignored.

T_i

2.2. Model of the system

This paper assume that the data centre has n physical hosts available or $Host = \{H_1, H_2, \dots, H_n\}$. Cloud data canters are generally part of heterogeneous clusters, with the VM serving as the core computing entity. On a single host, several VMs can be created. For ease of use, let's call the j-th VM on this host H_k. In the meanwhile, all VMs can be categorized under a heading we'll name $VC_i = VC_{k,j}$. For instance, $VC = \{VC_1, VC_2, \dots, VC_m\}$ denotes that H_k on this host is the i-th VM in the collection. This paper focuses on computationally intensive jobs heavily influenced by processing capability. The processing power of the host and the VM VC_i is designated as $PH_i\ and\ PV_i$. To device processing power, millions of instructions per second are utilized. $Task = \{T_1, T_2, \dots, T_{||Task||}\}$ denotes a real-time task set made up of T non-preemptive and independent tasks. A job T_i is described as $\|Task\| = (A_i, D_i, S_i)$, A_i, D_i, and S_i are the arrival time, deadline, and task size quantified in millions of instructions. Both replication and resubmission are thoroughly discussed in this study. The PB model is used for replication. Each task T_i has two copies, a primary copy T_i^P and a backup copy T_i^B, each of which is assigned to a different VMS on another host. The assignment is then re-submitted to the system if a problem develops with the task's initially assigned host. The mathematic explanation is simplified by expressing the first submission task as T_i^P and the subsequent submission task as T_i^B. Resubmission delays sending to the system T_i^B until the initial task has failed T_i^P, in contrast to replication. $R_{i,k}$ denotes the moment when VM VC_k was prepared to start working.

2.3. Runtime estimation

Usually, the job size and the system's performance influence how long a task takes to complete. Estimating the execution time for a specific job is challenging due to the dynamic demand that affects both the cloud's hardware and software performance. Because of this, past work has heavily relied on static estimation of task runtime to lessen problem complexity [16]. The discrepancy between expected length and actual required time significantly impacts task scheduling accuracy. By taking into account the following factors, we provide an efficient runtime estimation method for job H_i executing on VM VC_k:

- Factor 1: static processing is possible using P_k. It is the average processing power of VC_k. Thus, we can calculate the static runtime of task T_i as $TS_{i,k} = \frac{S_i}{P_k}$, where S_i is the task size.

- Factor 2: $TV_{i,k}$stands for runtime execution variation. In fact, it is closely related to the static runtime $TS_{i,k}$. As a result, we define $-a_i TS_{i,k} \leq TV_{i,k} \leq a_i TS_{i,k} (0 \leq a_i < 1)$, where i indicate task runtime uncertainty.
- Factor 3: $TD_{i,k}$ is the domain of runtime estimate. The static runtime and the uncertainty lead to $TD_{i,k} = TS_{i,k} + TV_{i,k}$. Consequently, the following phrase provides $TD_{i,k}$:

$$TD_{i,k} \in [(1-a)TS_{i,k}, (1+a)TS_{i,k}] \tag{1}$$

$\lfloor TD_{i,k} \rfloor = (1-a_i)TS_{i,k}$ *and* $\lceil TD_{i,k} \rceil = (1+a_i)TS_{i,k}$ are the lower and higher bounds of the predicted job runtime, respectively.

- Factor 4: we assume $TD_{i,k}$is the sample space of all possible actual runtimes, and $TR_{i,k}$is one of the sample points. We assume that the job runtime is distributed uniformly. The probability density function for $TR_{i,k}$ is expressed as (2).

$$f(TR_{i,k}) = \begin{cases} \frac{1}{2a_i TS_{i,k}}, \lfloor TD_{i,k} \rfloor < TR_{i,k} > \lceil TD_{i,k} \rceil \\ 0, \quad otherwise \end{cases} \tag{2}$$

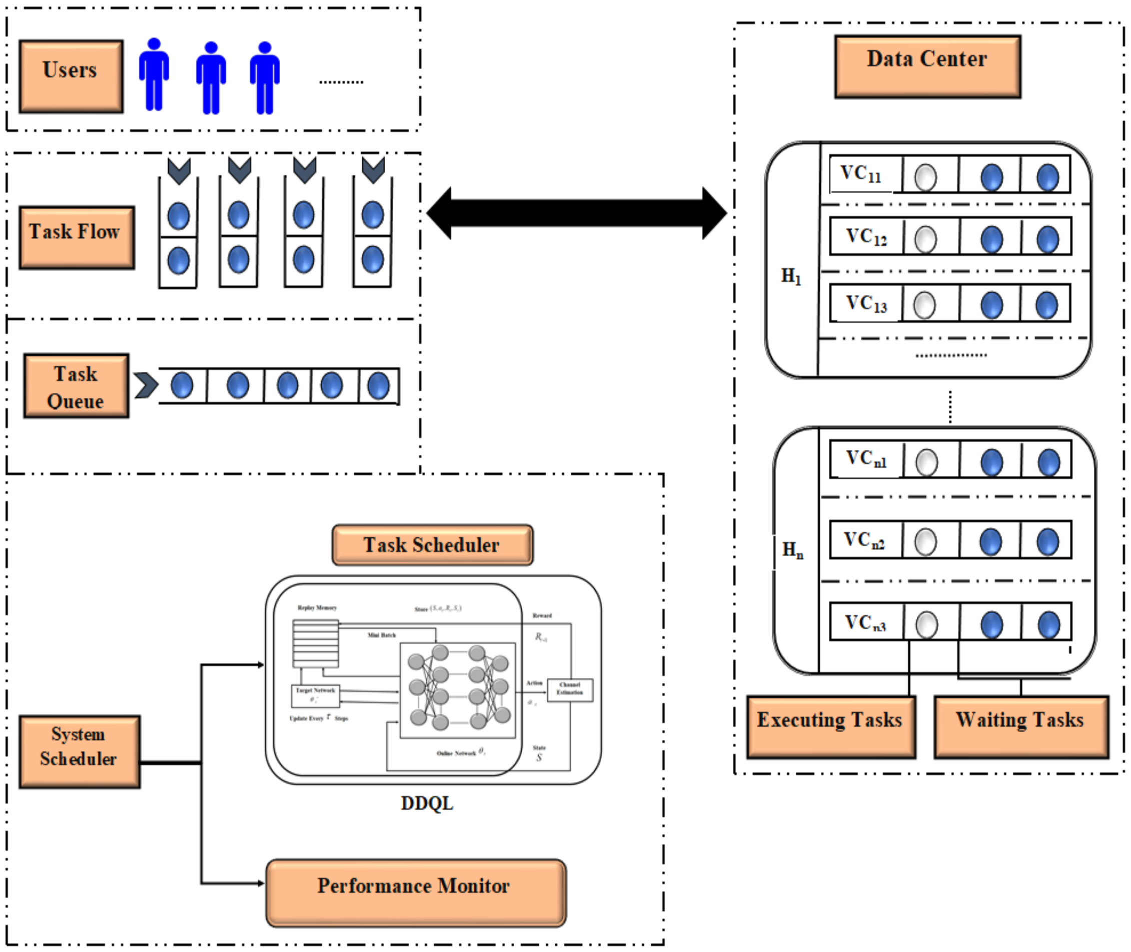

Figure 1. An overview of our proposed method

2.4. Fault model

A typical failure interval for computer systems is roughly 75 minutes. Nevertheless, the time needed to complete a cloud task is less than 3 minutes. So, in the case of a host failure, we anticipate that the tasks currently running on that host will be completed appropriately by their backup copies (or re-submitted copies) before another host encounters a problem. Additionally, we assume that the cloud infrastructure has methods for failure detection, such as acceptance tests and fail signals [17]. A host's failure is discovered right away. We presume that each host failure is distinct from the others. It is important to note that by classifying hosts into different groups, our fault-tolerant technique can be utilized to resolve various host problems.

2.5. Task scheduling algorithm

Ensuring resource distributions adhere to fault-tolerant requirements while allocating jobs to the proper VMs is the key problem with fault tolerance. The main problem and delay of a single job should, when using the replication strategy, be distributed among VMs on different hosts for fault tolerance since the failure of one host results in the failure of all VMs on it. A different host should always be chosen when a work is resubmitted after being abandoned due to host failure.

In this section, we consider scheduling alternatives for fitting new work inside time limitations, after which we examine task allocation ceilings when replication and resubmission are used. We assume N active VMs to maintain generality. We will first provide an overview of the three definitions below to help analyse scheduling strategies.

Definition 1. $ES_{i,k}^{p}$: earliest start time the $ES_{i,k}^{p}$ of a task T_i primary task on VC_k is defined the ES when T_i^p can primary execution if T_i^p assigned to the kth VM, as indicated by the equation:

$$ES_{i,k}^{p} = MAX\left\{R_{i,k}^{p}, A_i\right\}, k \in \{1, 2, \dots, N\} \tag{3}$$

initial time of finish $EF_{i,k}^{p}$ bottom and upper bounds can be determined by calculating task runtime estimates as follows:

$$\left\lfloor EF_{i,k}^{p} \right\rfloor = ES_{i,k}^{p} + \left\lfloor TD_{i,k}^{p} \right\rfloor, k \in \{1, 2, \dots, N\} \tag{4}$$

$$\left\lfloor EF_{i,k}^{p} \right\rfloor = S_{i,k}^{p} + \left\lfloor TD_{i,k}^{p} \right\rfloor, k \in \{1, 2, \dots, N\} \tag{5}$$

Definition 2. Latest start time $LS_{i,j}^{B}$: the backup (or resubmitted) task's $LS_{i,j}^{B}$ is defined as the LS at which $TR_{i,j}^{B}$ must backup execution if T_i^B is allotted to the jth VM if the real runtime $TR_{i,j}^{B}$ can be retrieved in development, which is resolute by the following expression:

$$LS_{i,j}^{B} = D_i - TR_{i,j}^{B}, j \in \{1, 2, \dots, N\} \tag{6}$$

the area of $TR_{i,j}^{B}$ is used to provide specified lower and upper boundaries that limit the latest start time;

$$\left\lfloor LS_{i,j}^{B} \right\rfloor$$

$$\left\lfloor LS_{i,j}^{B} \right\rfloor = D_i - \left\lfloor TR_{i,j}^{B} \right\rfloor, j \in \{1, 2, \dots, N\} \tag{7}$$

$$\left\lceil LS_{i,j}^{B} \right\rceil = D_i - \left\lceil TR_{i,j}^{B} \right\rceil, k \in \{1, 2, \dots, N\} \tag{8}$$

Definition 3. Task T_i dynamic time is defined by the dynamic time definition $DT_{i,k}$as follows:

$$DT_{i,k} = D_i - EF_{i,k}^{p}, k \in \{1, 2, \dots, N\} \tag{9}$$

the lower and higher constraints shown below confine the dynamic time $DT_{i,k}$:

$$\left\lfloor DT_{i,k} \right\rfloor = D_i - \left\lfloor EF_{i,k}^{p} \right\rfloor, k \in \{1, 2, \dots, N\} \tag{10}$$

$$\left\lceil DT_{i,k} \right\rceil = D_i - \left\lceil EF_{i,k}^{p} \right\rceil, k \in \{1, 2, \dots, N\} \tag{11}$$

three different scheduling scenarios are connected to fault tolerance.

Case 1. Non-fault tolerance (NFT): if the task's deadline T_i is less than its earliest EF time, and the task cannot be finished before the deadline on VC_k probably, the scheduling approach for the kth VM is NFT. Figure 2 depicts an example of NFT. If task T_i is assigned to VC_k, the lower certain of the earliest completion time $EF_{i,k}^{p}$ will exceed the deadline D_i, implying that the task can only be complete after the deadline. As a result, we refer to this as the NFT case.

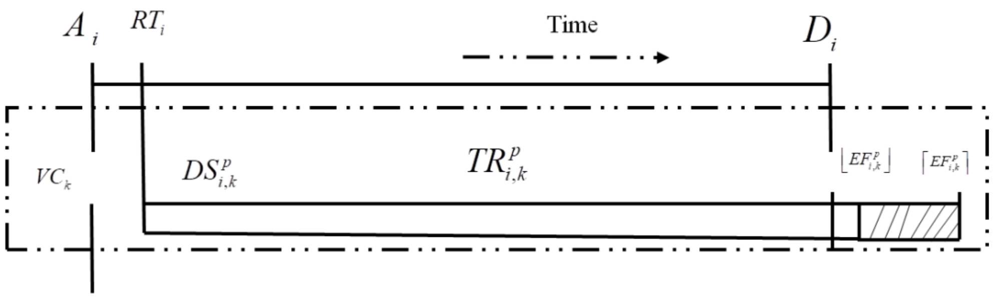

Figure 2. A sample of NFT

Case 2. Weak fault tolerance (WFT): if the task's deadline falls within the $EF_{i,k}^{p}$ domain and the task can or cannot be capable to appearance by its deadline on VC_k, the scheduling technique for the kth VM T_i is WFT. Figure 3 demonstrates an example of WFT. When a task T_i is assigned to VC_k, it is possible to determine that the deadline D_i is larger than but less than the $EF_{i,k}^{p}$. This means that there is a greater than 0% but less than 100% chance that the project will be completed on time. As a result, this is known as the WFT condition.

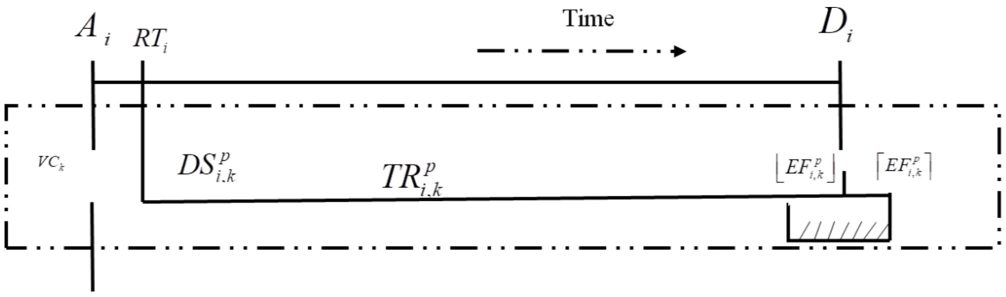

Figure 3. A sample of WFT

Case 3. Strong fault tolerance (SFT): if a task T_i can continuously be performed before its deadline on VC_k, the scheduling approach for the kth VM is SFT. Figure 4 depicts an example of SFT. While task T_i is assigned to the VC_k, the higher certain of the earliest completion time $EF_{i,k}^{p}$ is less than the deadline D_i, implying that the possibility of finishing the work before the deadline is 1. As a result, assign this scenario the SFT rating.

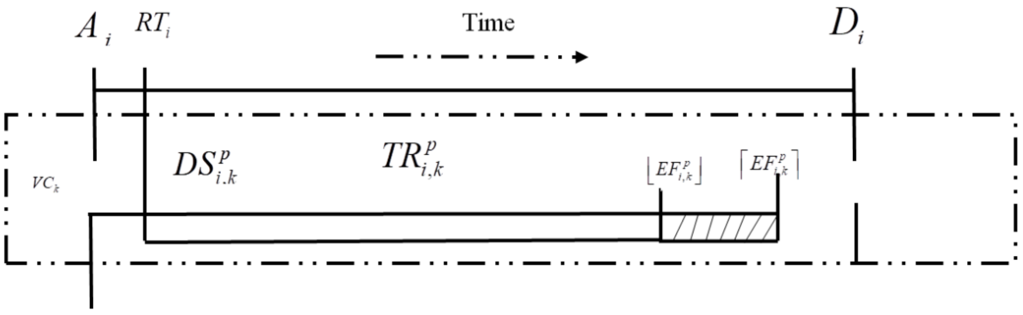

Figure 4. A sample of SFT

Corollary 1. $\forall T_i \in Task, if\lceil DT_{i,k}\rceil \leq 0$, then scheduling the task T_i to VC_k is NFT. Proof. According to (10) and (11), the dynamic time $DT_{i,k}$ reflects the comparative value amongst the earliest finish time $EF_{i,k}^{p}$ and the deadline D_i. If $DT_{i,k}$ is less than zero, $EF_{i,k}^{p}$ will exceed the D_i deadline. As a result, this scenario falls under NFT. Corollary 2. If $\forall T_i \in Task, if\lceil DT_{i,k}\rceil > 0\ and\lfloor DT_{i,k}\rfloor$, then scheduling task T_i to VM VC_k is WFT. Proof. The following deductions are proven by (10) and (11); (1) if $\lceil DT_{i,k}\rceil > 0$ is produced, then $\lceil EF_{i,k}^{p}\rceil \geq D_i$ is established; (2) if $\lceil DT_{i,k}\rceil \leq 0$ is formed, then $\lceil EF_{i,k}^{p}\rceil \leq D_i$ is deduced. As a result of WFT, the task can or cannot be completed by its deadline. Corollary 3. When the task $\forall T_i \in Task, if\lceil DT_{i,k}\rceil$ is scheduled, SFT is obtained if T_i to VM VC_k. Proof. We can deduce that $\lceil EF_{i,k}^{p}\rceil < D$ is produced from (10). SFT ensures the task is consistently finished ahead of schedule.

2.5.1. The fundamental scheduling approach for fault tolerance

Scheduling approaches should be varied because tasks and VMs are heterogeneous. Based on the activity's temporal characteristics, the scheduling method is examined. The task cannot be scheduled for the VM VC_k if task T_i on that VM is non-fault tolerant. If task T_i has non-fault tolerance across all VMs in the cloud, the scheduler would reject the job or create a new VM to complete it. The following sections investigate suitable scheduling options for weak and robust fault tolerance.

i. If task T_i is of WFT on VM VC_k

Weak fault tolerance D_i suggests that task T_i might be completed earlier than expected. The possibility of final the job $T_i$on VC_k before the deadline is what is meant by the likelihood $P_k(T_i)$.

$$P_k(T_i) = \frac{\lceil DT_{i,k}\rceil}{\lceil EF_{i,k}^{p}\rceil - \lceil EF_{i,k}^{p}\rceil} \tag{12}$$

Numerous VMs running on a cloud architecture could create a fault tolerance risk. T_i global optimum scheduling method's main goal is to determine the maximum $P_k(T_i)$ value for each VC_k while providing reliable fault tolerance. The formal definition of this value is $P_{max}(T_i) = MAX\{P_k(T_i)\}, \forall VC_k \in VMs$. However, it is incredibly unlikely that the work will be finished before the deadline if $P_k(T_i)$ is low. Scheduling the work on VC_k could waste resources and have an adverse effect on how the subsequent actions are carried out. We use a threshold $Task_i^{w}, 0 < Task_i^{w} \leq 1$ to regulate the scheduling strategy choice.

ii. If task T_i VM VC_kdisplays high fault tolerance

This situation allows task T_i to be routinely completed before its due date D_i. Determining when to use the resubmission method and when to employ the replication mechanism is a significant challenge regarding overall resource optimization. Corollary 4 is introduced here.

Corollary 4. $\forall T_i \in Task$, the replication mechanism must be used if $if\lceil EF_{i,k}^{p}\rceil \geq \lfloor LS_{i,j}^{B}\rfloor$. Proof. If $\lceil EF_{i,k}^{p}\rceil \geq \lfloor LS_{i,j}^{B}\rfloor$, then the initial job T_i^{p} and the resubmission task T_i^{B} will overlap in time. If the resubmission is used, there is not sufficient time to complete the work to be finished by its deadline $\lceil LS_{i,j}^{B}\rceil$when the first job fails after D_i. As a result, a replication technique must be used to achieve effective fault tolerance.

There can be no time-overlapping if $\lfloor EF_{i,k}^{p}\rfloor < \lceil LS_{i,j}^{B}\rceil$, therefore the resubmission procedure is possible. However, if $\lceil LS_{i,j}^{B}\rceil$ is just slightly greater than $\lfloor EF_{i,k}^{p}\rfloor$, as shown in Figure 5, there may need to be more time to complete the re-submitted assignment. If the initial job T_i^{p} fails at the $T_{fail} \geq \lceil LS_{i,j}^{B}\rceil$ donated like the arrow in this example, the resubmitted work cannot be completed.

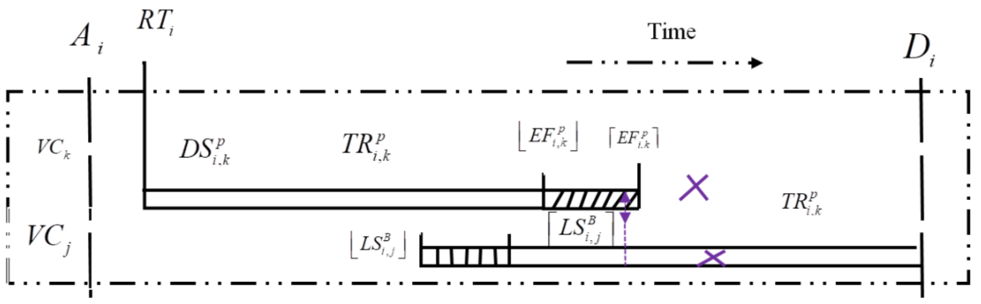
A_i
RT_i
Time
D_i
VC_k
$DS_{i,k}^{p}$
$TR_{i,k}^{p}$
$\lfloor EF_{i,k}^{p}\rfloor$
$\lceil EF_{i,k}^{p}\rceil$
$TR_{i,k}^{p}$
VC_j
$\lfloor LS_{i,j}^{B}\rfloor$
$\lceil LS_{i,j}^{B}\rceil$

Figure 5. An example of ineffective resubmission fault tolerance

2.6. Enhanced scheduling method for high fault tolerance

Resubmission and replication are covered in this section, along with their availability and effects on task scheduling. Due to oscillations in task runtime, task scheduling can adopt resubmission. Resubmission comes in a variety of forms that fall into two categories. Task scheduling is efficiently allocating and managing computational tasks across a system's resources to maximize throughput, minimize latency, and ensure the successful execution of tasks. In the context of SFT and replication, task scheduling has added significance as it must consider the redundancy and distribution of tasks to ensure uninterrupted operation in the face of failures.

Definition 4. Time of intersection $IST_{k,j}(T_i)$ the time window during which the implementation of T_i^p and T_i^B might coincide is known as the intersection time of a task T_i. $IST_{k,j}(T_i)$ mathematical expression is as (13).

$$IST_{k,j}(T_i) = \lceil EF_{i,k}^p \rceil - \lfloor LS_{i,j}^B \rfloor \quad (13)$$

2.7. DDQL

DDQL is an innovative method that can transform fault tolerance in real-time systems. DDQL uses deep reinforcement learning to improve systems' resilience in time-critical contexts. This sophisticated technique combines deep learning with reinforcement learning to enable systems to react to errors and proactively prevent possible failures, dramatically enhancing fault tolerance. This paper explores the fascinating topic of using DDQL to improve fault tolerance in real-time systems. We look into the fundamental principles of DDQL, practical uses, and its potential to alter how we see and handle fault tolerance problems in critical systems. DDQL presents a viable road ahead in reaching improved levels of dependability and resilience in real-time systems, ultimately leading to safer and more efficient operations across a wide range of disciplines. Join us as we investigate systems DDQL and its potential to revolutionize the field of fault tolerance.

2.7.1. DQN framework

The Q-learning (QL) parameter't' is modified following the action. If the reward 'Rt' has caused the state 'St' to change, then;

$$\theta_{t+1} = \theta + \beta \left(X_t^Q - Q(S, \alpha_1; \theta_t) \right) \nabla_{\theta t} Q(S, \alpha_t; \theta_t) \quad (14)$$

where β is the step size and X_t^Q is the aim, expressed as;

$$X_t^Q = R + \gamma \, max_a Q(S_t, a; \theta_t) \quad (15)$$

Where γ is the discount factor, stochastic gradient descent is used to update the current value $Q(S_t, A_t; \theta_t)$ in the direction of the goal value X_t^Q. A DQL is a multi-layered neural network that generates a network parameter and an action vector of $Q(S_t, .; \theta)$ for a state that is described by S_t. The neural network's ^{n} function is built for state spaces with n dimensions and action domains with m activities. Consequently, DQN's purpose is to.

$$X_t^{DQN} = R + \gamma max_a Q(S_t, a; \theta_t^-) \quad (16)$$

where τ the settings are reset after each step, so that $\theta_t^- = \theta_t$.

As previously mentioned, the max operator, as stated in (17) and (18), is used by both QL and DQN, which selects and evaluates actions using the same values. Because of this similarity, DQN has a problem that causes unduly optimistic value estimates. In the context of DDQL, experiences are allocated at random to update one of two different value functions, creating two sets of weights (selectors and evaluators). As a result, (7) goal value can be written as follows:

$$X_t^Q = R + \gamma Q(S_t, \arg max_a Q(S_t, a; \theta_t); \theta_t) \quad (17)$$

the QL mistake twice as (18).

$$X_t^{DoubleQ} = R + \gamma Q(S_t, argmax_a Q(S_t, a; \theta_t) \theta_t') \quad (18)$$

2.7.2. DDQN based training procedure

A prominent value-based DRL technique that can experience overestimation is DQN (deep Q-networks) [18]. Maximizing the Q-Learning technique leads to overestimation, which happens when the estimated value function exceeds the real value function? We use DDQN to compute the Q-network $Q_{\emptyset}$ (S, a) of the candidate's actions to directly maximize the quality of SFC deployment to solve this problem. Learning the parameter that will enable Q-network $Q_{\emptyset}$ (S, a) to approach Q-function Q_{π} (S, a) is the aim of DDQN. A training episode, a time slot, includes many MDP state transitions. These state changes are buffered and used for training from the start of each episode until the end. The following is an outline of the target function:

$$X = R + \gamma Q(S_t, argmax_a Q(S_t, a; \theta_t); \theta^-) \quad (19)$$

in this case, θ stands for the weights of the original Q-function Q and θ for the consequences of the intended Q-function Q. The rewards given as feedback are denoted by R. In contrast, the state at the following time step is represented by St. A fully connected neural network with five layers and 50 neurons per hidden layer makes up the proposed DQL architecture. Tanh is the hidden neurons' activation function, while a linear activation layer is the output. The DDQL algorithm's structure is shown in Figure 6.

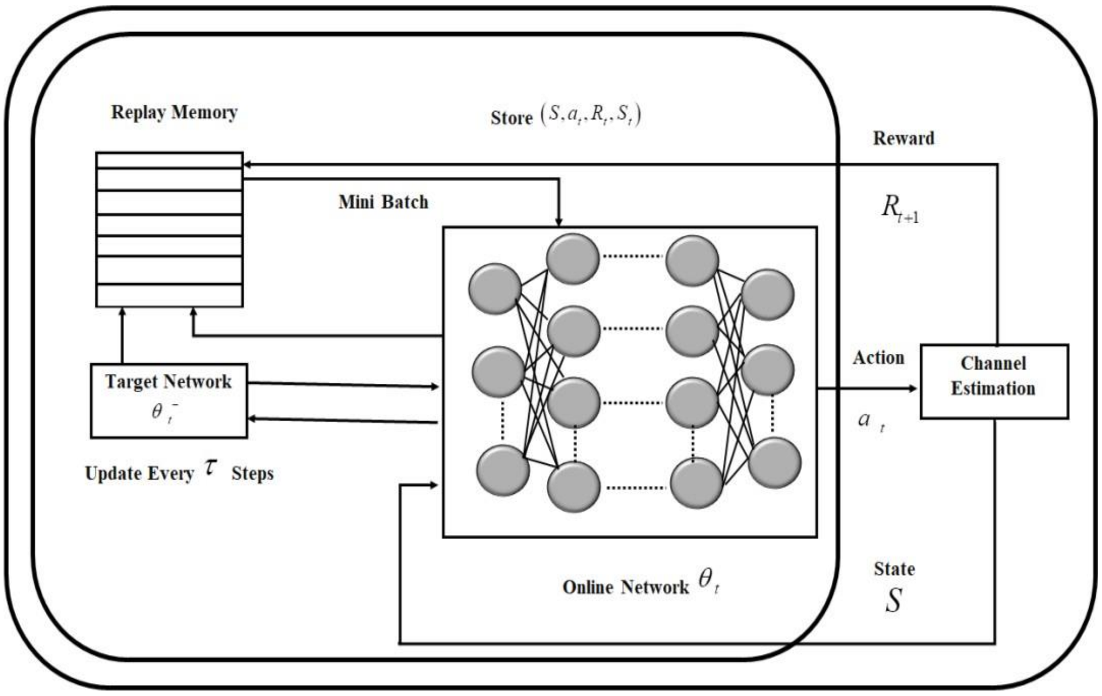
Replay Memory
Store (S, a_t, R_t, S_t)
Reward
R_{t+1}
Mini Batch
Action
a_t
Channel Estimation
Target Network θ_t^-
Update Every τ Steps
State S
Online Network θ_t

Figure 6. Structure of DDQL

Algorithm 1 describes the DDQN-based training technique. We set up the Q network infrastructure in each episode. With each MDP state transition, the agent improves fault tolerance in real-time systems and earns a reward Rt. To be more specific, to balance policy exploitation with environmental research, we use the -greedy technique for selecting actions. Additionally, we train functions Q and Q with different parameter values using the double method, which significantly decreases the problem of overestimation and yields outstanding results.

Algorithm 1. Double deep Q networkbased training procedure

```
Step 1: Begin: Initialize replay memory D to capacity N
Step 2: Initialize Q-function Q with random weights θ
Step 3: Initialize target Q-function $\hat{Q}$ with weights $\theta^{-=}$ θ
Step 4: for episode $\longleftarrow$ 1, M do
Step 5: Initialize the Q network environment and let state S $\longleftarrow$ $S_1$ and preprocessed sequence $\phi 1 = \phi(S_1)$
Step 6: for $t \longleftarrow 1, T$ do
Step 7: Select an action $a_t$ randomly with the probability €, otherwise select action $a_t = \max Q (S_t, a; \theta)$
Step 8: Execute action $a_t$ and observe reward $R_t$
Step 9: Transfer the state to S and preprocess $(\phi, a_t, R_t, \phi_{t+1})$
Step 10: Store transition (φt, at, rt, φt+1) in D
Step 11: Sample random minibatch of transitions $(\phi_j, a_j, R_j, \phi_{j+1})$ from D
Step 12: If episode terminates at step j + 1 then
Step 13: Set $x_j = R_j$
Step 14: else
Step 15: Set $a' = \text{argmax}_{a'}, Q (\phi_{j+1}, a, \theta)$
Step 16: Set $x_j = R_j, + \gamma \hat{Q}(\phi_{j+1,} a'; \theta^-)$
Step 17: end if
Step 18: Perform a gradient descent step on $(x_i, -Q (\phi_j, a_j, \theta))^2$ with respect to the network parameters $\theta$
Step 19: Every C steps reset $\hat{Q} = Q$
Step 20: end for
Step 21: end for
```

2.8. Advantages of proposed method

- DFTRTSA's capacity to tolerate defects in real-time systems is one of its key advantages. It can adapt and recover from unforeseen failures by incorporating fault tolerance methods into the scheduling algorithm, ensuring the system's stability.
- DFTRTSA is intended to maximize the use of available resources such as CPU time and memory. It guarantees that vital jobs are efficiently scheduled, eliminating resource waste.
- Because DFTRTSA is dynamic, it can adapt to changing situations in real-time systems. It can alter scheduling priorities and techniques based on the current condition and workload of the system to ensure optimal performance.
- When DDQL and DFTRTSA are combined, the system can make proactive scheduling decisions that minimize downtime and maximize the availability of key services or tasks.
- DFTRTSA can efficiently distribute resources to many tasks with varying priorities and requirements. This adaptability is critical in situations when duties vary in importance.

3. RESULTS AND DISCUSSION

This section primarily encompasses extensive simulations conducted for DDQL-DFTRTSA. We employed random workload generation; incorporated statistical task dispersion using various artificially generated datasets based on different distributions, and utilized real-time work logs from sources [27], [28]. To evaluate the performance of the DDQL-DFTRTSA scheduler, we conducted an extensive set of experiments utilizing the cloudsim [29] simulator. To begin our analysis, we initially employed various statistical data distributions to evaluate all relevant parameters. We generated datasets labeled as D01, D02, D03, and D04, which corresponded to uniform, normal, left-skewed, and right-skewed distributions, respectively. Following this, we calculated network lifetime using these datasets. Subsequently, we shifted our focus to real-time work logs, referred to as D05 and D06 throughout our research. The deliberate selection of these datasets for use in this simulation is noteworthy, as many previous authors have traditionally relied on randomly generated workloads. However, it is essential to recognize that the utilization of randomly generated workloads in addressing scheduling problems of this nature does not yield precise schedules. Therefore, we opted to create distinct distributions of datasets, specifically D01 through D04. In addition, to assess the energy consumption of the approach; we chose real-time work logs from both the HPC2N computing cluster and NASA work logs. This section encompasses various subsections, which cover configuration settings required for the simulation, the computation of throughput, energy consumption, end-to-end delay, network lifetime, guarantee ratio, and the task acceptance ratio of VMs. A thorough discussion of the produced results and analysis is presented in a separate subsection. The experimental design for this investigation is detailed in subsection 3.1, followed by descriptions of the comparative methodologies in subsection 3.2 and the performance metrics in subsection 3.3. Finally, a detailed discussion of the generated results and analysis is presented in subsection 3.4.

3.1. Experimental setup

This subsection provides a clear representation of the experimental setup and the standard configuration settings employed in our simulation. To assess the proposed DDQL-DFTRTSA, we conducted a comparative analysis with existing state-of-the-art approaches, namely ICFWS, PCFT, EPRD, and WFMS are among the current approaches that have been contrasted utilizing the configuration settings obtained from [28]. Table 1 displays the configuration settings applied to the simulation for the proposed DDQL-DFTRTSA.

Table 1. Configuration settings for simulation

Name	Quantity
No. tasks	1,000
Length of tasks	900,000
Processing elements	1,200 MIPS
Hypervisor type	Monolithic
Physical host hard disk capacity	2 TB
Processing elements	1,200 MIPS
OS of physical host	MAC
Bandwidth of virtual resources	15 Mbps
Bandwidth capacity of physical host	100 Mbps
Memory of virtual host	2,048 MB
Name of the hypervisor	Xen
Operating system of virtual host	Linux
Physical host memory	32 GB
No. of datacenters	10

3.2. Comparative methods

To validate the performance of our model, we compared our results with the following state-of-the-art. ICFWS, PCFT, EPRD, and WFMS are among the current approaches that have been contrasted. PCFT [6]: PCFT, on the other hand, is only appropriate for parallel applications and is ineffective for processes. To put it another way, the check-pointing system can handle short-term resource failures, not long-term ones like HPF. ICFWS [7]: the ICFWS method breaks down the overall workflow deadline into several smaller deadlines at the level of the individual subtasks. EPRD [14]: the EPRD algorithm is proposed to reduce task scheduling length without violating the end-to-end deadline limitation. WFMS [15]: online documentation for fault-tolerant approaches used in various WFMS is available. They also offer a comprehensive taxonomy of the different fault tolerance mechanisms utilized in distributed situations.

3.3. Performance evaluation

In assessing our comprehensive approach's effectiveness in fortifying fault tolerance within real-time systems, we've opted for various evaluation metrics. These metrics encompass throughput, energy consumption, end-to-end delay, network lifetime, guarantee ratio, and task acceptance ratio. Through the prism of these metrics, we aim to gauge the performance and resilience of our proposed approach. Each metric offers a distinct vantage point, enabling a multifaceted evaluation of the system's robustness. By leveraging these diverse metrics, we seek to provide a comprehensive analysis of how our approach impacts the various facets crucial to fault tolerance in real-time systems.

3.3.1. Throughput analysis

Throughput analysis refers to inspecting and assessing the system’s ability to maintain consistent and uninterrupted data processing and task execution in case of faults or failures. This study aids in the identification of potential bottlenecks and adjustments to ensure that the system can continue to satisfy its performance and timing requirements in the face of adversity. The throughput analysis of the DDQL-DFTRTSA approach in comparison to the current techniques is shown in Table 2 and Figure 7. The node demonstrates unequivocally how superior in every way the recommended strategy is to the alternatives. The DDQL-DFTRTSA method, for instance, has a throughput of 1334.19 kbps with 100 nodes, compared to the throughputs of the other existing methods, such as PCFT, ICFWS, DEFT, and EPRD, which are, respectively, 934.34 kbps, 1189.87 kbps, 1098.14 kbps, and 1034.19 kbps. The suggested technique has a throughput of 1455.67 kbps with 500 nodes, compared to 999.78 kbps, 1245.98 kbps, 1187.98 kbps, and 1156.87 kbps for the current methods, PCFT, ICFWS, DEFT, and EPRD respectively. This demonstrates the higher performance and higher throughput of the DDQL-DFTRTSA approach.

3.3.2. Energy consumption analysis

Energy consumption refers to the systematic examination and evaluation of power usage within real-time computing systems to improve their ability to withstand and recover from faults or failures while optimizing energy efficiency. Table 3 and Figure 8 display the energy consumption analysis of the DDQL-DFTRTSA method with existing methods. The proposed method consumes very little energy compared to the other techniques for any number of nodes. For example, with 100 nodes, the DDQL-DFTRTSA method consumes only 21.87J while the other methods like PCFT, ICFWS, DEFT, and EPRD consume 31.88J, 36.76J, 26.88J, and 40.19J respectively. Similarly, with 500 nodes, the proposed DDQL-DFTRTSA method consumes only 25.55J, whereas the other methods, like PCFT, ICFWS, DEFT, and EPRD, consume 35.19J, 39.12J, 30.13J, and 44.66J, respectively. The proposed method shows higher performance with less energy consumption.

Table 2. Throughput analysis for DDQL-DFTRTSA method with existing systems

Number of nodes	PCFT	ICFWS	DEFT	EPRD	DDQL-DFTRTSA
100	934.34	1189.87	1098.14	1034.19	1334.19
200	936.91	1191.54	1099.96	1067.78	1345.23
300	967.98	1195.87	1134.18	1112.87	1376.19
400	989.99	1221.17	1176.77	1145.18	1412.19
500	999.78	1245.98	1187.98	1156.87	1455.67

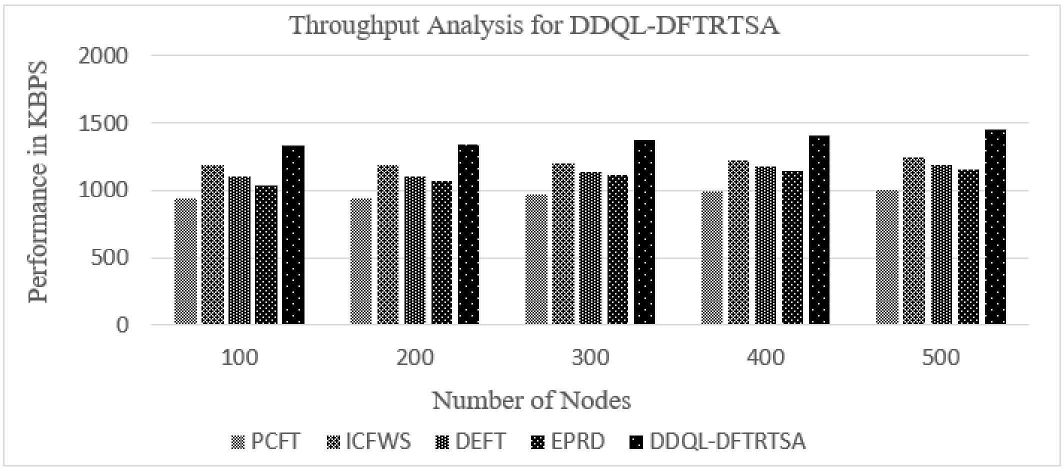

Figure 7. Throughput analysis for DDQL-DFTRTSA method with existing systems

Table 3. Energy consumption analysis for DDQL-DFTRTSA method with existing systems

Number of nodes	PCFT	ICFWS	DEFT	EPRD	DDQL-DFTRTSA
100	31.88	36.76	26.88	40.19	21.87
200	32.44	37.12	27.12	41.16	22.56
300	33.98	38.88	28.88	42.33	23.76
400	34.55	38.98	29.99	43.55	24.18
500	35.19	39.12	30.13	44.66	25.55

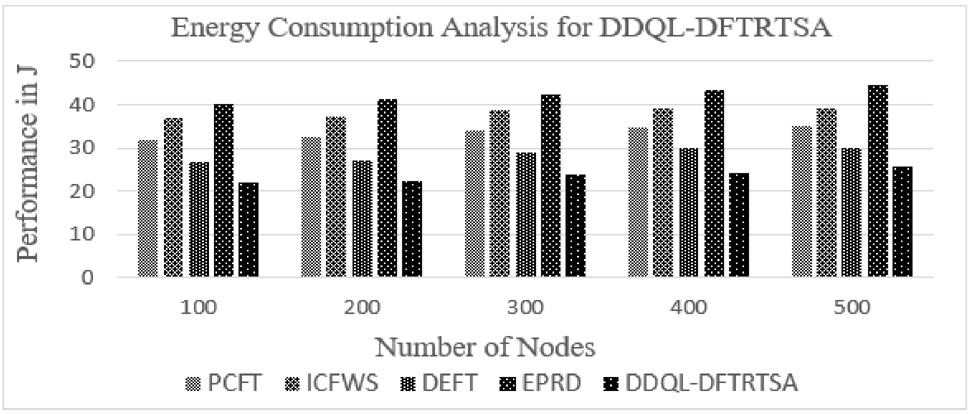

Figure 8. Energy consumption analysis for DDQL-DFTRTSA method with existing systems

3.3.3. Network lifetime analysis

Network lifetime refers to evaluating and optimizing the durability and reliability of interconnected systems, particularly in real-time settings, to ensure continued operation even in the presence of faults or failures. This analysis includes examining the network's lifespan and applying techniques to improve fault tolerance, thereby improving the system's capacity to satisfy its performance objectives over a prolonged period. The suggested DDQL-DFTRTSA methodology's network lifetime is contrasted with existing methods in Table 4 and Figure 9. The results determine unequivocally that the DDQL-DFTRTSA method outperformed all other techniques. The suggested DDQL-DFTRTSA approach, for example, took only 0.234 sec as its network lifetime for 100 nodes. In contrast, other current methods such as PCFT, ICFWS, DEFT, and EPRD have taken 8.115sec, 6.113sec, 4.567sec, and 2.675sec, respectively. Similarly, the suggested DDQL-DFTRTSA approach takes 1.998sec as its network lifetime of 500 nodes, while existing techniques like PCFT, ICFWS, DEFT, and EPRD have taken 10.334sec, 8.456sec, 5.887sec, and 4.187sec, respectively.

Table 4. Network lifetime analysis for DDQL-DFTRTSA method with existing systems

Number of nodes	PCFT	ICFWS	DEFT	EPRD	DDQL-DFTRTSA
100	8.115	6.113	4.567	2.675	0.234
200	9.567	6.456	4.987	2.776	0.156
300	9.178	7.543	5.112	3.443	0.786
400	10.651	7.771	5.554	3.678	1.556
500	10.334	8.456	5.887	4.187	1.998

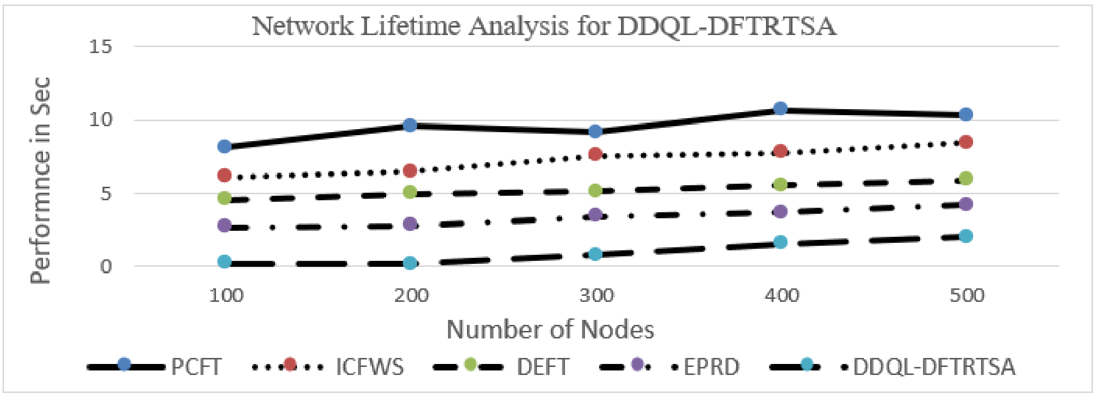

Figure 9. Network lifetime analysis for DDQL-DFTRTSA method with existing systems

3.3.4. End to end delay analysis

End-to-end delay analysis evaluates and optimizes the amount of time data or signals travel through a system to assure the system's ability to tolerate errors and sustain dependable operation in real-time or time-sensitive applications. The suggested DDQL-DFTRTSA methodology's end-to-end delay is contrasted with existing methods in Table 5 and Figure 10. The results determine unequivocally that the DDQL-DFTRTSA method outperformed all other techniques. The suggested DDQL-DFTRTSA approach, for example, took only 1.345sec as its end-to-end delay with 100 nodes. In contrast, other current methods such as PCFT, ICFWS, DEFT, and EPRD have taken 10.234sec, 8.123sec, 6.789sec, and 4.156sec, respectively. Similarly, the suggested DDQL-DFTRTSA approach takes 3.987sec as its end-to-end delay with 500 nodes, while existing techniques like PCFT, ICFWS, DEFT, and EPRD have taken 12.987sec, 9.567sec, 7.998sec, and 6.177sec, respectively.

Table 5. End to end delay analysis for DDQL-DFTRTSA method with existing systems

Number of nodes	PCFT	ICFWS	DEFT	EPRD	DDQL-DFTRTSA
100	10.234	8.123	6.789	4.156	1.345
200	10.987	8.456	6.998	4.987	1.987
300	11.234	8.998	7.134	5.556	2.567
400	12.765	9.145	7.897	5.987	2.667
500	12.987	9.567	7.998	6.177	3.987

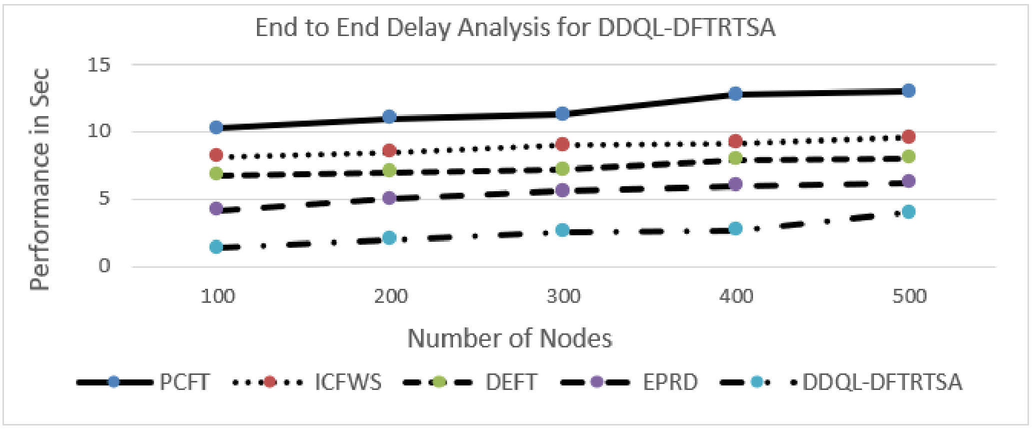

Figure 10. End to end delay analysis for DDQL-DFTRTSA method with existing systems

3.3.5. Guaranteed ratio analysis

Guaranteed ratio is a methodology for assessing and optimizing the reliability and performance of real-time computer systems by analysing the ratio of guaranteed task completions to total task executions, thereby improving the system’s ability to withstand faults and disruptions while meeting critical timing requirements. A comparison of the DDQL-DFTRTSA strategy’s guarantee ratio to various existing methods is shown in Figure 11 and Table 6 The graph illustrates how the deep learning approach has an improved efficiency with the guaranteed ratio. In contrast to the guaranteed ratio values of 75.12%, 82.66%, 79.12%, and 87.45% for the PCFT, ICFWS, DEFT, and EPRD models, respectively, the DDQL-DFTRTSA model has a guaranteed ratio of 92.76% for 100 nodes. However, the DDQL-DFTRTSA model has performed better with various nodes. The DDQL-DFTRTSA model has a guaranteed ratio of 95.66% under 500 nodes, compared to the PCFT, ICFWS, DEFT, and EPRD models, which have guaranteed ratio values of 78.94%, 86.77%, 81.87%, and 91.45%, respectively.

Table 6. Guaranteed ratio analysis for DDQL-DFTRTSA method with existing systems

Number of nodes	PCFT	ICFWS	DEFT	EPRD	DDQL-DFTRTSA
100	75.12	82.66	79.12	87.45	92.76
200	76.66	83.44	80.78	88.98	93.87
300	77.88	84.55	80.66	89.45	93.44
400	78.34	85.66	81.23	90.17	94.55
500	78.94	86.77	81.87	91.45	95.66

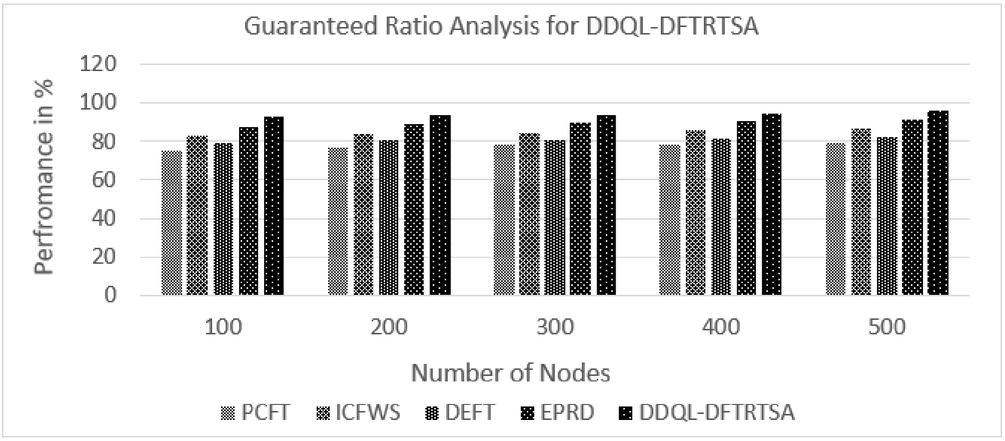

Figure 11. Guaranteed ratio analysis for DDQL-DFTRTSA method with existing systems

3.3.6. Ratio of task accepted analysis

The ratio of task accepted analysis is a metric that measures the percentage or proportion of tasks or requests that are successfully accepted or completed within a certain environment, it aids in assessing efficiency and performance by calculating the success rate in completing activities or assignments. A comparison of the DDQL-DFTRTSA strategy's ratio of tasks accepted to various existing methods is shown in Figure 12 and Table 7. The graph illustrates how the deep learning approach has an improved efficiency with increase in ratio of tasks accepted. In contrast to the ratio of 69.45%, 78.98%, 73.87%, and 87.88% for the SMANN, MS-LBP, FAGWO, OB-LBP, and DBN models, respectively, the DDQL-DFTRTSA model has a ratio of of 93.44% for 100 nodes. However, the DDQL-DFTRTSA model has performed better with various data sizes. The DDQL-DFTRTSA model has a ratio of task accepted with 97.44% under 500 nodes, compared to the SMANN, MS-LBP, FAGWO, OB-LBP, and DBN models, which have a ratio of 72.78%, 85.88%, 77.12%, and 92.67%, respectively.

Table 7. Ratio of task accepted analysis for DDQL-DFTRTSA method with existing systems

Number of nodes	PCFT	ICFWS	DEFT	EPRD	DDQL-DFTRTSA
100	69.45	78.98	73.87	87.88	93.44
200	69.66	80.77	74.66	88.12	94.67
300	70.45	82.77	75.89	89.67	95.55
400	71.23	83.21	76.98	91.23	96.89
500	72.78	85.88	77.12	92.67	97.44

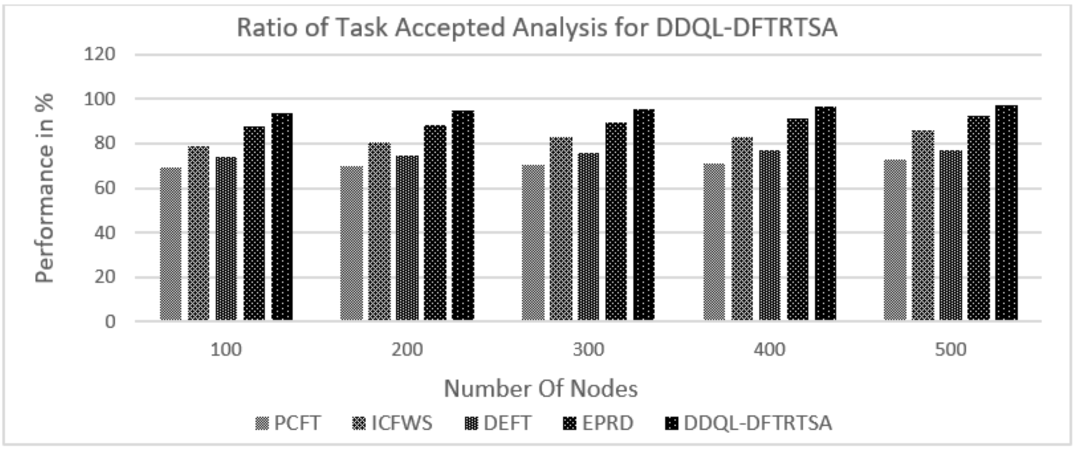

Figure 12. Ratio of task accepted analysis for DDQL-DFTRTSA method with existing systems

3.4. Analysis and result discussion

This section provides a clear presentation of result analysis and discusses how the proposed DDQL-DFTRTSA enhances scheduling while addressing parameters such as throughput, energy consumption, end-to-end delay, network lifetime, guarantee ratio, and task acceptance ratio. We performed an extensive series of simulations utilizing cloudsim [31]. Initially, we generated datasets with various statistical distributions, labeled as D01, D02, D03, and D04. Subsequently, we evaluated DDQL-DFTRTSA using real-time work logs, namely HPC2N [29] and NASA, represented as D05 and D06. The suggested DDQL-DFTRTSA approach was compared to existing advanced methods, ICFWS, PCFT, EPRD, and WFMS. Table 2 illustrates the improvement in throughput for DDQL-DFTRTSA compared to existing approaches, while Table 3 demonstrates the reduction in energy consumption. In Table 4, we observe the extension of network lifetime, and Table 5, the enhancement of end-to-end delay. Furthermore, Table 6 indicates an improvement in guarantee ratio, and Table 7 displays an increase in task acceptance ratio. Collectively, these results highlight the significant impact of the proposed DDQL-DFTRTSA in generating schedules and enhancing service level agreement-based trust metrics, ultimately improving the quality of service and fostering trust in the cloud provider. A key distinction between our approach and other methods lies in our consideration of task and VM priorities. DDQL-DFTRTSA outperforms existing approaches in terms of energy consumption, throughput, network lifetime, end-to-end delay, guarantee ratio, and task acceptance ratio.

4. CONCLUSION

In conclusion, combining DDQL and a DFTRTSA is a viable and new method for addressing real-time system difficulties. This study demonstrated the utility of DDQL in optimizing task scheduling and resource allocation while combining DFTRTSA's robustness and adaptability in the face of faults and uncertainty. The findings of this study reveal that DDQL, through its reinforcement learning capabilities, can effectively adapt to changing system conditions and optimize resource allocation, resulting in enhanced real-time scheduling performance. When combined with DFTRTSA, the system becomes efficient and resilient, with the ability to mitigate the impact of faults and disruptions. Additionally, DDQL and DFTRTSA's partnership has the potential to significantly increase the dependability and predictability of real-time systems across a range of industries, including aerospace, automotive, industrial automation, and more. The requirement for robust and adaptive real-time systems will only grow as technology advances, making integrating DDQL and DFTRTSA an appealing route for future study and practical implementation. According to the experimental results, the proposed work achieves a throughput of 1455.67kbps, a network lifetime of 1.998sec, an end delay of 3.987sec, energy consumption of 25.55J, a guaranteed ratio of 95.66%, a ratio of task accepted of 97.44%. It remains robust when compared to current methods. Developing the future of real-time systems provided increased performance and the resilience required to handle the challenges of a constantly evolving technological context.

REFERENCES

[1] A. Rista, J. Ajdari, and X. Zenuni, "Cloud computing virtualization: a comprehensive survey," in *2020 43rd International Convention on Information, Communication and Electronic Technology, MIPRO 2020 - Proceedings*, Sep. 2020, pp. 462–472, doi: 10.23919/MIPRO48935.2020.9245124.

[2] J. Kumar, A. K. Singh, and R. Buyya, "Self directed learning based workload forecasting model for cloud resource management," *Information Sciences*, vol. 543, pp. 345–366, Jan. 2021, doi: 10.1016/j.ins.2020.07.012.

[3] H. Raei and N. Yazdani, "Performability analysis of cloudlet in mobile cloud computing," *Information Sciences*, vol. 388–389, pp. 99–117, May 2017, doi: 10.1016/j.ins.2017.01.030.

[4] T. Jin and B. Zhang, "Intermediate data fault-tolerant method of cloud computing accounting service platform supporting cost-benefit analysis," *Journal of Cloud Computing*, vol. 12, no. 1, p. 2, Jan. 2023, doi: 10.1186/s13677-022-00385-4.

[5] Z. Ahmad, A. I. Jehangiri, M. A. Ala'anzy, M. Othman, and A. I. Umar, "Fault-tolerant and data-intensive resource scheduling and management for scientific applications in cloud computing," *Sensors*, vol. 21, no. 21, p. 7238, Oct. 2021, doi: 10.3390/s21217238.

[6] J. Liu, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, Oct. 2018, doi: 10.1109/TCC.2016.2567392.

[7] G. Yao, Y. Ding, and K. Hao, "Using imbalance characteristic for fault-tolerant workflow scheduling in cloud systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3671–3683, Dec. 2017, doi: 10.1109/TPDS.2017.2687923.

[8] G. Yuan *et al.*, "Fault tolerant placement of stateful VNFs and dynamic fault recovery in cloud networks," *Computer Networks*, vol. 166, p. 106953, Jan. 2020, doi: 10.1016/j.comnet.2019.106953.

[9] T. Long *et al.*, "A novel fault-tolerant scheduling approach for collaborative workflows in an edge-IoT environment," *Digital Communications and Networks*, vol. 8, no. 6, pp. 911–922, Dec. 2022, doi: 10.1016/j.dcan.2022.08.010.

[10] P. Zhang, S. Shu, and M. Zhou, "An online fault detection model and strategies based on SVM-grid in clouds," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, pp. 445–456, Mar. 2018, doi: 10.1109/JAS.2017.7510817.

[11] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Fault-tolerance in the scope of cloud computing," *IEEE Access*, vol. 10, pp. 63422–63441, 2022, doi: 10.1109/ACCESS.2022.3182211.

[12] S. J. Nirmala, A. R. Setlur, H. S. Singh, and S. Khoriya, "An efficient fault tolerant workflow scheduling approach using replication heuristics and checkpointing in the cloud," *Journal of Parallel and Distributed Computing*, vol. 136, pp. 14–28, Feb. 2020, doi: 10.1016/j.jpdc.2019.09.004.

[13] R. Wang, N. Chen, X. Yao, and L. Hu, "Fasdq: fault-tolerant adaptive scheduling with dynamic qos-awareness in edge containers for delay-sensitive tasks," *Sensors*, vol. 21, no. 9, p. 2973, Apr. 2021, doi: 10.3390/s21092973.

[14] L. Zhang, L. Zhou, and A. Salah, "Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments," *Information Sciences*, vol. 531, pp. 31–46, Aug. 2020, doi: 10.1016/j.ins.2020.04.039.

[15] D. Poola, M. A. Salehi, K. Ramamohanarao, and R. Buyya, "A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments," in *Software Architecture for Big Data and the Cloud*, Elsevier, 2017, pp. 285–320.

[16] Y. Ding, G. Yao, and K. Hao, "Fault-tolerant elastic scheduling algorithm for workflow in cloud systems," *Information Sciences*, vol. 393, pp. 47–65, Jul. 2017, doi: 10.1016/j.ins.2017.01.035.

[17] H. Yan, X. Zhu, H. Chen, H. Guo, W. Zhou, and W. Bao, "DEFT: dynamic fault-tolerant elastic scheduling for tasks with uncertain runtime in cloud," *Information Sciences*, vol. 477, pp. 30–46, Mar. 2019, doi: 10.1016/j.ins.2018.10.020.

[18] L. Wang, W. Mao, J. Zhao, and Y. Xu, "DDQP: a double deep q-learning approach to online fault-tolerant sfc placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 118–132, Mar. 2021, doi: 10.1109/TNSM.2021.3049298.

[19] S. Thangam, E. Kirubakaran, and J. William, "Architecture for service selection based on consumer feedback (FBSR) in service oriented architecture environment," *Asian Journal of Information Technology*, vol. 13, no. 5, pp. 282–286, 2014, doi: 10.3923/ajit.2014.282.286.

[20] R. Panwar and M. Supriya, "Dynamic resource provisioning for service-based cloud applications: a bayesian learning approach," *Journal of Parallel and Distributed Computing*, vol. 168, pp. 90–107, Oct. 2022, doi: 10.1016/j.jpdc.2022.06.001.

[21] P. Prakash, R. Suresh, and P. N. D. Kumar, "Smart city video surveillance using fog computing," *International Journal of Enterprise Network Management*, vol. 10, no. 3–4, pp. 389–399, 2019, doi: 10.1504/IJENM.2019.103165.

[22] P. Prakash, K. G. Darshaun, P. Yaazhlene, M. V. Ganesh, and B. Vasudha, “Fog computing: issues, challenges and future directions,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 6, pp. 3669–3673, Dec. 2017, doi: 10.11591/ijece.v7i6.pp3669-3673.
[23] M. P. Singh, K. Sangeeta, and B. Sreevidya, “Hardware setup for vlc based vehicle to vehicle communication under fog weather condition,” *International Journal of Advanced Science and Technology*, vol. 29, no. 3 Special Issue, pp. 145–152, 2020.
[24] T. Deepika and P. Prakash, “Power consumption prediction in cloud data center using machine learning,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 2, pp. 1524–1532, Apr. 2020, doi: 10.11591/ijece.v10i2.pp1524-1532.
[25] H. R. Sandeep and S. Thangam, “A hybrid cloud approach for efficient data storage and security,” *Proceedings of the 6th International Conference on Communication and Electronics Systems, ICCES 2021*, pp. 1072–1076, 2021, doi: 10.1109/ICCES51350.2021.9488938.
[26] G. N. Iyer, “Evolutionary games for cloud, fog and edge computing-a comprehensive study,” in *Advances in Intelligent Systems and Computing*, vol. 990, 2020, pp. 299–309.
[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software - Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011, doi: 10.1002/spe.995.
[28] C. Santoro, F. Messina, F. D’Urso, and F. F. Santoro, “Wale: a dockerfile-based approach to deduplicate shared libraries in docker containers,” in *Proceedings - IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, IEEE 16th International Conference on Pervasive Intelligence and Computing, IEEE 4th International Conference on Big Data Intelligence and Computing and IEEE 3rd Cyber Science and Technology Congress, DASC-PICom-DataCom-CyberSciTec 2018*, Aug. 2018, pp. 776–784, doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00135.
[29] F. D’Urso, C. Santoro, and F. F. Santoro, “Wale: a solution to share libraries in docker containers,” *Future Generation Computer Systems*, vol. 100, pp. 513–522, Nov. 2019, doi: 10.1016/j.future.2019.03.049.

BIOGRAPHIES OF AUTHORS

Chetankumar Kalaskar received the B.Eng. degree in Poojya Doddappa Appa College of engineering Karnataka India in 2009 and the Master of Technology Poojya Doddappa Appa College of Engineering Karnataka India 2012 Currently, he is an Assistant professor at the Department of Computer science Poojya Doddappa Appa College of Engineering. His research interests include deep learning, machine learning, and cloud computing. He can be contacted at email: k_chetankumar@blr.amrita.edu.

Dr. Thangam Somasundaram currently serves as an Assistant Professor (SG) in the department of computer science at the Amrita School of Computing Bengaluru Amrita Vishwa Vidyapeetham India. Her area of interest in research includes service-oriented architecture, networks, data structures and cloud computing. She completed her Ph.D. in Computer Science and Engineering from Anna University, Chennai. India. She has 22 years of teaching experience. She has published her research works in 8 international journals. She is a member of ISTE. She can be contacted at email: s_thangam@blr.amrita.edu.