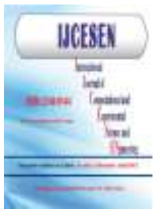
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 10-No.4 (2024) pp. 1094-1100
http://www.ijcesen.com

ISSN: 2149-9144

Research Article



Enhancing Fault Tolerance in Cloud Computing using Modified Deep Q-Network (M-DQN) for Optimal Load Balancing

Bikash Chandra Pattanaik[1*], Bidush Kumar Sahoo[2], Bibudhendu Pati[3], Arabinda Pradhan[4]

[1]Department of CSE, Biju Patnaik University of Technology, Rourkela, Odisha, India,
*** Corresponding Author Email:** bikashpatnaik73@gmail.com - **ORCID:** 0009-0000-6713-1492

[2]Department of CSE, GIET University, Gunupur, Odisha, India,
Email: bidush.sahoo@gmail.com-**ORCID**: 0000-0002-5044-0819

[3]Department of CS, Ramadevi Women's University, Bhubaneswar, Odisha, India,
Email: patibibudhendu@gmail.com-**ORCID:** 0000-0002-2544-5343

[4]Department of CSE, Gandhi Institute for Education and Technology, Khurdha, Odisha, India,
Email: arabindapradhan@giet.edu.in-**ORCID:** 0000-0002-3299-8990

Article Info:

DOI: 10.22399/ijcesen.601
Received : 09 November 2024
Accepted : 19 November 2024

Keywords :

Cloud computing,
Load balancing,
Fault tolerance,
DQN.

Abstract:

Due to popularity of cloud computing approach, excessive cloud user can send their request to cloud server for accessing their requirements. Servers are handling these incoming requests and allocate required resources to fulfill user demands. But in real scenario the numbers of servers are limited. Therefore, some servers are heavily loaded and some servers are in idle mode. This can result in a major fault tolerance issue that reduces system performance. To overcome this issue, this study presented an effective scheduling mechanism known as Modified Deep Q-Network (M-DQN). In this process the data centre controller performs appropriate actions on the environment in order to select a suitable virtual machine (VM) capable of optimizing different load balancing parameters. To get the desired outcome, a simulation is run using Google Colab with the TensorFlow environment, demonstrating the usefulness of the proposed scheduling technique. The experiment revealed that our suggested approach has a higher reward rate, reduces makespan but increases resource utilization and throughput when compared to the existing DQN algorithm. Simulation findings demonstrate that the M-DQN method works better in decreasing around 16% execution time and 10% makespan time, while it increases 8% resource utilization and 4% throughput value. Overall, it increases 18% reward value as compare with I-DQN and DQN algorithm.

1. Introduction

Cloud computing is becoming an increasingly common and comprehensive form of computer operations. It delivers the services that the user requests. These services are accessible at a cheap cost and on demand. The amount of requests for cloud computing services is rising due to the numerous benefits it provides. These services are typically provided by data centre, which contain a large number of servers. The data centre has inadequate servers to handle user requests. Using the virtualization approach servers are virtually separated into several virtual machines (VMs), having identical or distinct configuration from that of its host computer or computers [1]. Along with various positives, there are certain disadvantages to cloud computing, one of which is failure tolerance in load balancing.

Fault tolerance enables a system to continue to function even if one of its components fails. If a fault occurs then it reduces the system capacity rather than shutting down completely [2]. When a fault occurs in cloud computing, it should be identified first, followed by determining the nature of the fault and ultimately recovering the system without impacting the final output. In general, failures are occur in cloud computing to maintain load balancing in VMs. As a result, load balancing in cloud computing is a difficult topic [3]. The workload in the cloud may fluctuate on a regular basis due to user demand, making it difficult to assign these resources [4]. This difficulty occurs under a variety of settings. Examples include: (1)

task length and size fluctuations, (2) a lack of suitable VMs in the data centre, and (3) the status of available VMs. An effective scheduling technique may solve fault tolerance in load balancing by equitably distributing the whole job among available VMs, guaranteeing that all VMs are balanced while also minimizing processing time. Traditional scheduling techniques, such as Reinforcement Learning (RL) is not properly work well in this scenario. In RL algorithm, storage problem occur when number of state action value is generated. Also it takes more calculation time in high-dimensional scenarios [5]. Avoiding the drawbacks of RL approach, Deep Reinforcement Learning (DRL) techniques for example Deep Q-Network (DQN) is used. In DQN, neural networks are used for solving the load balancing issues [6]. However, the typical DQN method chooses a random action [7]. It yields an inaccurate result since there is no exact action structure built in a dynamic environment to optimize reward. To solve the above issues, this paper introduces an effective scheduling method called as Modified Deep Q-Network (M-DQN). The goal of this strategy is to apply an intelligent action that reduces execution and makespan time while improving resource utilization and throughput. The primary contribution of this study is mentioned as follows:

- Developed an effective M-DQN method which allows an agent to select the optimal action to discover suitable VMs from the data centre to achieve the aim.
- The suggested method optimizes outcomes by interacting with the environment to achieve high reward values.
- Our suggested algorithm's efficiency is demonstrated by simulation using the Google Colab and TensorFlow.

The rest of the paper is arranged as: Section 2 presents the relevant work. Section 3 explains the goal function. Section 4 represents the proposed scheduling algorithm. Section 5 depicts the experiment analysis. Section 6 finishes with the conclusion.

2. Related Work

Several academics have devised methodologies that are presently being utilized to maximise various fault tolerance parameters in load balancing. Such strategies are based on various machine learning-based scheduling approaches.

A Q-learning technique was introduced in [8] to spread the work burden across the VMs in order to improve QoS. To increase the system performance and handle load balance, [9] presented the adaptive fast reassignment approach. Cloud storage systems can be managed by Adaptive Resource Management technique which enhances cloud storage performance while preserving and balancing the load [10]. To reduce makespan time and increase incentive, [11] presented an efficient DRL scheduling. To handle task scheduling and resource allocation, [12] proposed a two-stage algorithm. A DRL algorithm was developed by both [13, 14] to optimize makespan and resource consumption. An Improved Deep Q-networking method was proposed in [15] that can increase the success rate while optimizing various parameters. Task completion time was reducing by [16] which is based on Deep Q-network algorithm.

From the study, we noticed that majority of researchers try to balance the load, limit makespan, and increase resource usage. But a fundamental drawback of most existing researches is that they consider VM migration instead of task migration. The VM migration concept has a high cost that might influence all services. However, our suggested approach can identify which VM is overloaded or under loaded by monitoring its status. It then selects the optimal VM to assign the load to base on its server selection. This approach aids in avoiding faults that may occur when balancing the load in a VM. Furthermore, the proposed technology is allowing users to receive services in lower costs and shortest amount of time.

3. Objective Function

Fig. 1 depicts the basic paradigm of cloud computing, which includes task and data centre layer. Both layers play a significant role in cloud computing and enable us to achieve our goals. The task layer contains the task queue, which saves all of the relevant information about the incoming task and locates the best VM on the server. Similarly, data center layer contains all information for both servers and VMs which help the data center controller to reach an objective.

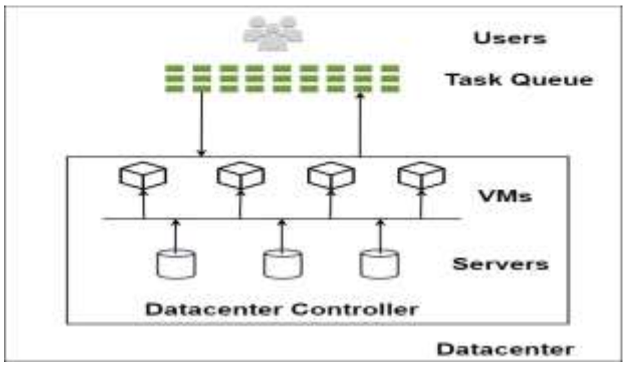


Figure 1. *Cloud computing model*

Our aim is to improve work completion time, resource usage, and throughput by taking necessary action. Based on the aim, we utilized the DQN method to determine the reward function, allowing us to improve the system.

To obtain the outcome, consider a scenario where a data centre includes v number of heterogeneous VMs and t numbers of tasks are entering into it. Initially, tasks are placed in a task queue and assigned to VMs on a first-come first-serve basis. VM selection is decided by the present load (VM_{ld}), capacity (VM_{cap}) and service rate (VM_{sr}) as shown in Eq. (1), (2) and (3). According to [17] each VM has three phases. After evaluating the VM phases, the best VM is chosen to handle the task and reduce the overall makespan time. As a result, calculates the probable execution time ($ET_{probable}$), task allocation time (T_{at}) and complete execution time (ET_{full}) as shown in Eq. (4), (5) and (6). Finally, makespan time which is shown in Eq. (7). Where VM_b, VM_{mips} and VM_{cpu} is represent as bandwidth, MIPS, CPU of VM. T_t, T_{leng} and T_{fs} represent number of task, its length and file size. The resource usage(Res_{util}) is obtained in Eq. (8).

$$VM_{ld} = \frac{T_t \times T_{leng}}{VM_{sr}} \quad \ldots (1)$$

$$VM_{cap} = VM_{sr} + VM_b \quad \ldots (2)$$

$$VM_{sr} = VM_{mips} \times VM_{cpu} \quad \ldots (3)$$

$$ET_{probable} = \frac{T_{leng}}{VM_{sr}} \quad \ldots (4)$$

$$T_{at} = T_{fs}/VM_b \quad \ldots (5)$$

$$ET_{full} = ET_{probable} + T_{at} \quad \ldots (6)$$

$$MS = max\{ET_{full}\} \quad ... (7)$$

$$R_{util} = \frac{ET_{full}}{MS} \quad ... (8)$$

4. M-DQN Algorithm

This section defines the core notion and reward function of M-DQN method. M-DQN follows DQN approach where an agent receives the reward by performing suitable actions on environment. In this procedure, the data centre controller functions as an agent, data centre serves as environment, VM is representing as state space and action is defined to allocate the task on a suitable VM for execution.

Due to the dynamic nature of the cloud computing, the load of each VM varies, causing some to be overloaded and others to be under loaded. To equalize the load among the VMs, the agent allocates additional jobs from overloaded to under loaded VMs. Hence, task allocation rate (T_{ar}) is used to select best action which is shown in Eq. (9) and the selected action a at time t denoted by a_t can then be determined using equation (10). Where, $EL_{p,q}$ represents the excess load between VM p to VM q. The needed bandwidth between VM p and VM q is represented as$B_{p,q}$. Finally, the reward function is represented in Eq. (11). Table 1 is representing the pseudo code for proposed algorithm.

$$T_{ar} = \frac{EL_{p,q}}{BW_{p,q}} \quad \ldots (9)$$

$$a_t = \max_a(T_{ar}) \quad \ldots (10)$$

$$r_f = \min\{MST/R_{util}\} \quad \ldots (11)$$

Table 1. *Pseudo code of M-DQN*

```
Input: Information about task, VM and server; Initialize all DQN parameters.
Output: Reward of data centre.

1. Start
2. For each cycle t, $s_1$ to be initialized with capacity and load
     For every task in the task-queue,
     If the probability is ∈, a random action $a_t$ to be chosen
     Otherwise $a_t = \max_a(T_{ar})$
     End if
     Using the action $a_t$, calculate $r_t$ (the total reward) using Eq. (11)
     Move to $s_{t+1}$ (the new state)
     Save the transition $(s_t, a_t, r_t, s_{t+1})$ in memory $E$
     Use $target_t = r_t + \gamma \max_{a_{t+1}} Q'(s_{t+1}, a_{t+1}; \theta')$
     Use gradient descent to evaluate error $L(\theta) = E[(target_t - Q(s_t, a_t; \theta))^2]$
     For every step $\theta' = $ □
     End For
3. End For
4. Return reward
5. Stop
```

5. Experimental Analysis

Our experiment is done in Google Colab using Python 3.9 and TensorFlow 1.4.0 on Windows 10 64-bit OS. In our simulation, we took 10,000 tasks with varying lengths and file sizes, such as 250 to 300. These tasks were spread over 100 VMs. The suggested M-DQN approach is tested and compared to various current techniques, including I-DQN and DQN. Fig. 2 to 9 displays all of the simulation findings. Table 2 displays all the features of the task, virtual machine and DQN method.

5.1 Execution time

The execution time of M-DQN is depicted in Table 3, Table 4, Fig. 2 and Fig. 3. Furthermore, the M-DQN method's performance is compared to that of other machine learning methods, including I-DQN

Table 2. *Features of tasks, VMs and DQN parameters.*

Task Properties		**VM Properties**		**DRL Properties**	
Task range	2000-10000	VM range	20-100	Maximum iteration	100
Length	500-3000	Processing speed	250-300	Learning rate	0.1
File Size	250	Memory	256-512	Discount factor	0.9
		CPU	1-5	Value of ∊	0.5-0.9
		Bandwidth	1000	Replay memory	1000
		VMM	XEN	Neurons	15

and DQN. The data set utilized in Fig. 2 is Table 3, which has multiple task sets ranging from 2000 to 10000 tasks and a fixed number of VMs of 100. Similarly, Table 4 and Fig. 3 include 20 to 100 VMs and a set amount of tasks, i.e., 10,000. Fig. 2 and Fig. 3 indicate that, while both I-DQN and DQN algorithms completed the tasks in longer time, the newly constructed M-DQN algorithm completed the entire job in the shortest amount of time because it better balances exploration and exploitation. The X- and Y-axes in Fig. 2 display the task number and execution time in seconds, whereas the X- and Y-axes in Fig. 3 indicate the VM number and execution time in seconds. Initially, in Fig. 3, we discovered that the suggested M-DQN scheduling method lowers execution time by around 3.13% to 7.1% when compared to the I-DQN algorithm and 8.63% to 16.52% when compared to the DQN algorithm when the number of jobs rises from 2000 to 10,000. Similarly, Fig. 3 shows that the proposed technique saves around 2.8% to 4.9% of execution time when compared to the I-DQN algorithm and 4.97% to 15.83% when compared to the DQN algorithm.

5.2 Makespan time

The makespan time of our proposed technique M-DQN is shown in Table 5, Table 6, Fig. 4 and Fig. 5. Table 5 contains the data for Fig. 4, in which

Table 3. *Execution time of 100 numbers of VMs and Task number is 2000 to 10000.*

No. of VM	**No. of Task**	**M-DQN**	**I-DQN**	**DQN**
100	2000	7421.82	7658.35	8091.17
100	4000	9745.64	10174.47	10487.94
100	6000	13221.17	14415.55	15218.18
100	8000	18719.75	20243.99	21834.61
100	10000	23089.64	24786.28	27249.84

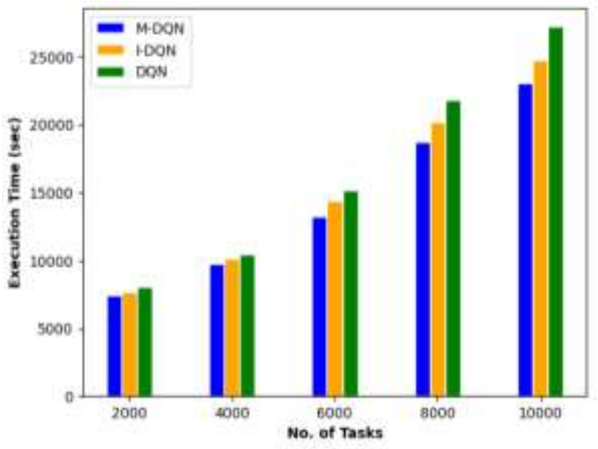


Figure 2. *Execution time of different tasks*

Table 4. *Execution time of 20 to 100 numbers of VMs and 10000 numbers of tasks.*

No. of VM	**No. of Task**	**M-DQN**	**I-DQN**	**DQN**
20	10000	33437.61	34374.26	35143.62
40	10000	30671.40	31154.12	32274.35
60	10000	26891.12	27705.48	29354.61
80	10000	23716.93	25624.98	27876.25
100	10000	22394.32	23518.34	26246.51

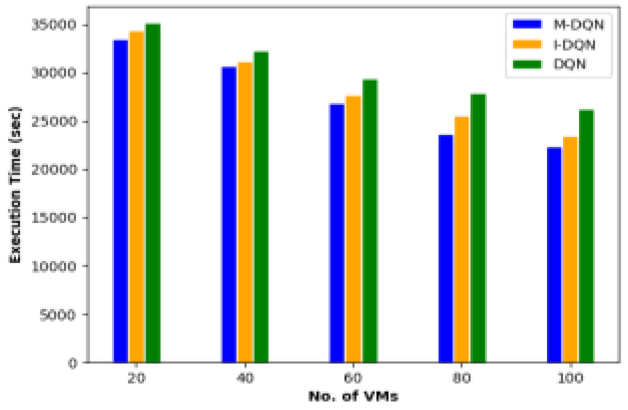


Figure 3. *Execution time of different VMs*

2000 to 10000 tasks are taken along with a fixed number of VMs. Table 6 contains the dataset for Fig. 5, in which the number of tasks is fixed at 10000 and 20 to 100 VMs. The proposed algorithm's simulation results are compared to those of other cutting-edge algorithms currently in use, such as I-DQN and DQN. The task number and makespan time in seconds are shown by the X-axes and Y-axes in Fig. 4, and the VM number and makespan time in seconds are represented by the X-axes and Y-axes in Fig. 5. Initially, in Fig. 4 we have to found that proposed M-DQN scheduling algorithm reduce approximately 3.35% to 6.66% makespan time as compared to I-DQN algorithm and 6.39% to 9.57% makespan time as compared to DQN algorithm when number of tasks is increases. Similarly, in Fig. 5 the proposed algorithm is reduce approximate 2.34% to 4.71%of makespan time as compare to I-DQN algorithm and 5.74% to 10.64% makespan time as compare to DQN algorithm.

Table 5. *Makespan time of 2000 to 10000 task and 100 numbers of VMs.*

No. of VM	**No. of Task**	**M-DQN**	**I-DQN**	**DQN**
100	2000	2254.31	2331.31	2403.18
100	4000	2531.52	2633.25	2737.27
100	6000	2724.64	2862.24	2911.25
100	8000	2887.18	2987.24	3025.63
100	10000	2978.91	3184.10	3278.48

\

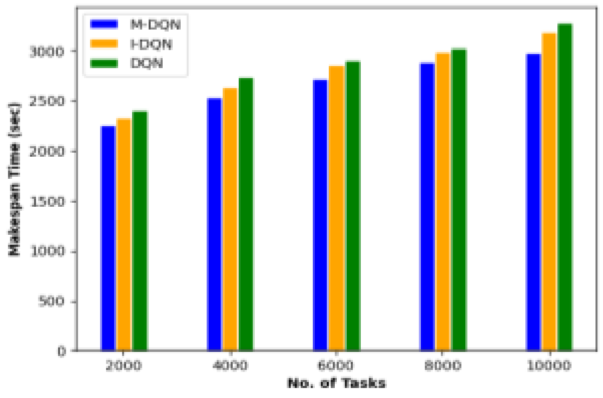


Figure 4. *Makespan time of different tasks*

Table 6. *Makespan time of 10000 number of tasks and VM is 20 to 100.*

No. of VM	**No. of Task**	**M-DQN**	**I-DQN**	**DQN**
20	10000	3557.74	3642.25	3768.25
40	10000	3412.63	3561.47	3624.34
60	10000	3366.45	3405.79	3463.61
80	10000	3197.21	3288.93	3376.84
100	10000	2947.17	3089.62	3123.44

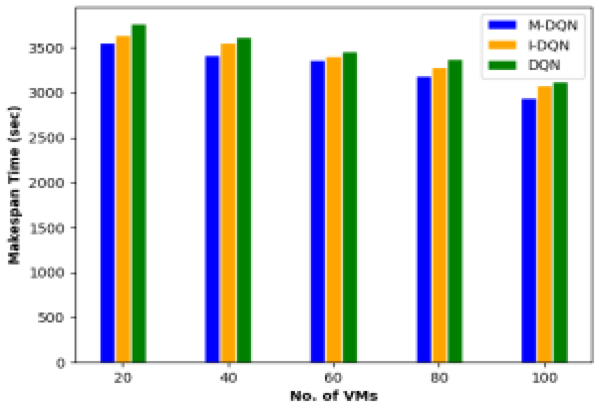


Figure 5. *Makespan time of different VMs*

5.3 Resource utilization

Fig. 6 and Fig. 7 shows the performance of average resource utilization of M-DQN, I-DQN and DQN algorithms where M-DQN algorithm shows the improvement of resource utilization as compare to other two algorithms. Fig. 6 depicts the average resource use for 2000 to 10,000 task sets, with 100 VMs picked at each iteration. This figure shows that the suggested M-DQN algorithm may boost average resource usage by 2.78% to 4.76% when compared to the I-DQN method and 4.19% to 8.48% when compared to the DQN algorithm. Table 7 displays the data for Fig. 6. Fig. 7 depicts the typical resource use of 20 to 100 virtual machines (VMs) with 10,000 jobs each iteration. According to this figure, the suggested M-DQN algorithm may boost average resource usage by around 4.31% to 5.29% when compared to the I-DQN method and 5.79% to 7.48% when compared to the DQN algorithm. Table 8 displays the data for Fig. 7.

Table 7. *Average resource utilization of 100 numbers of VMs and different number of tasks.*

No. of VM	**No. of Task**	**M-DQN**	**I-DQN**	**DQN**
100	2000	0.73	0.71	0.70
100	4000	0.79	0.76	0.75
100	6000	0.83	0.79	0.78
100	8000	0.85	0.81	0.79
100	10000	0.86	0.82	0.79

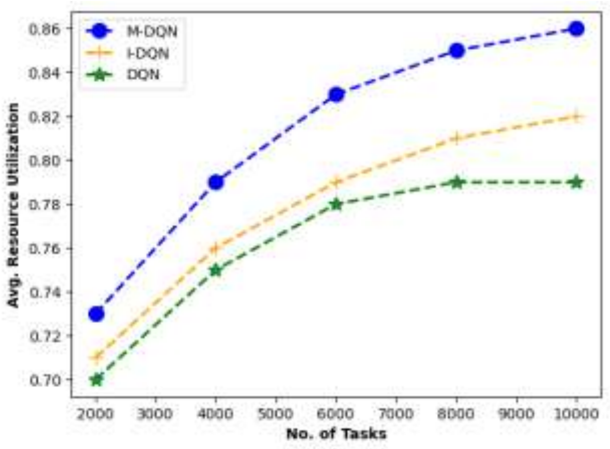


Figure 6. *Average resource utilization obtained for 2000 to 10000 task sets*

Table 8. *Average resource utilization of 10000 number of tasks and 20 to 100 number of VMs.*

No. of VM	**No. of Task**	**M-DQN**	**I-DQN**	**DQN**
20	10000	0.71	0.68	0.67
40	10000	0.85	0.81	0.77
60	10000	0.92	0.86	0.83
80	10000	0.96	0.91	0.89
100	10000	0.97	0.92	0.90

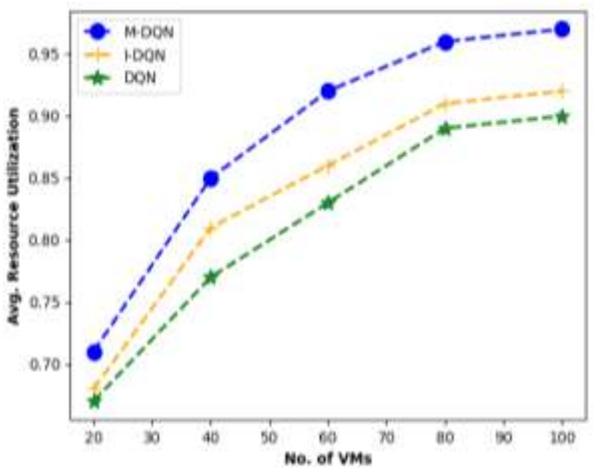


Figure 7. *Average resource utilization obtained for 20 to 100 VMs*

5.4 Throughput

The throughput measures the system's overall performance. Throughput refers to the number of tasks that may be accomplished in a particular period of time. Throughput of M-DQN is improved due to maintain of the balance between exploration-exploitation through which the suggested algorithm completes more jobs in a given amount of time, whereas other baseline methods struggle with overloading or under loading. Fig. 8 shows that the constructed M-DQN outperforms earlier baseline methods in terms of throughput. Fig. 8 shows the number of jobs on the X-axis and the system's throughput in seconds on the Y-axis. According to Fig. 8, the suggested scheduling approach improves by approximately 0.88% to 1.64% when compared to the I-DQN algorithm and 1.65% to 4.27% when compared with the DQN algorithm. Table 9 displays the data for Fig. 8.

5.5 Reward Value

Fig. 9 depicts the reward value of 100 number of iteration with 2000 task. From iteration 10 to iteration 25, the payout proportion rises from 55% to 80%. After iteration 25, the payout proportion remains constant, however it varies between iterations. Finally, it displays an estimated 80.7% at the conclusion of iteration. At the last iteration, the I-DQN and DQN algorithms showed reward values

Table 9. *Throughput value of different algorithms*

Schedule algorithm	2000	4000	6000	8000	10000
M-DQN	34.14	35.17	36.09	35.81	36.11
I-DQN	33.84	34.76	35.58	35.13	35.52
DQN	33.58	34.21	34.78	34.22	34.6

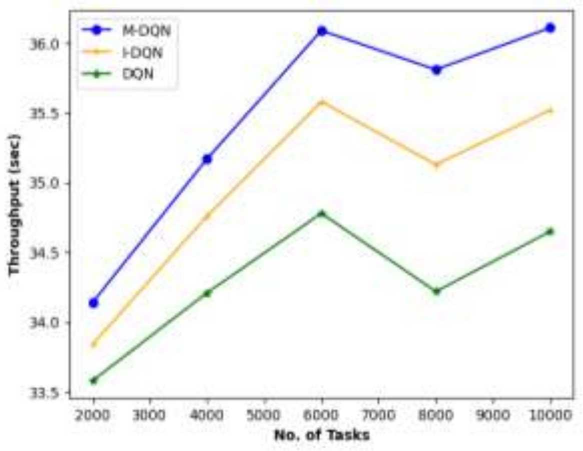


Figure 8. *Throughput value*

of 67.9% and 62.3%, respectively. The experiment revealed that our suggested method outperforms the I-DQN and DQN scheduling algorithms by 12.8% and 18.4%, respectively.

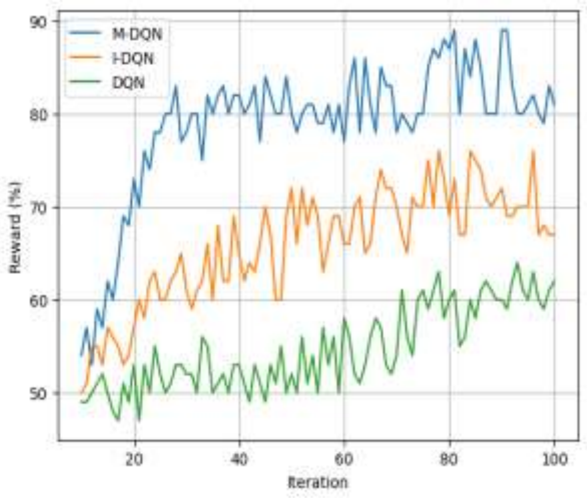


Figure 9. *Reward comparison of 2000 task set.*

6. Conclusion

One of the hardest problems in cloud computing is deciding which resources are best for end-user applications. While service providers want to maximize their profits by providing the best resources, end users want their apps to run in a certain amount of time and at the lowest possible cost. In this paper, we created cloud resource allocation architecture and all of its components to address the constraints of the present method. The resource monitoring node utilizes the M-DQN algorithm to continually monitor the state of VMs and fulfil the user demand. The performance of the M-DQN method is compared with I-DQN and DQN algorithms. The key findings of the experimental investigation, in contrast to I-DQN and DQN, are as follows:

- M-DQN technique reduces execution time by roughly 16%.
- The suggested approach decreases makespan time by approximately 10%.
- The suggested method improves resource consumption by around 8% on average.
- The suggested technique enhances throughput by around 4%.

The suggested approach boosts reward value by approximately 18%.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Pradhan, A., Bisoy, S. K., and Mallick, P. K. (2020). Load balancing in cloud computing: survey. In book: *Innovation in Electrical Power Engineering, Communication, and Computing Technology, Lecture Notes in Electrical Engineering, springer*, pp 99-111.

[2] Rehman, A. U., Aguiar, R. L., Barraca, J. P. (2022). Fault-Tolerance in the Scope of Cloud Computing. *IEEE Access*, 10;63422-63441.

[3] Pradhan, A., Bisoy, S. K., Das, A. (2022). A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment. *Journal of King Saud University –Computer and Information Sciences,* 34(8);4888-4901 https://doi.org/10.1016/j.jksuci.2021.01.003

[4] Pattnaik, B. C., Sahoo, B. K., Pradhan, A., Mishra, S. R., Tripathy, H. S., Agasti, P. (2024). Fault Tolerance Enhancement Through Load Balancing Optimization in Cloud Computing. *International Journal of Intelligent Systems and Applications in Engineering*, 12(4), 172–180. https://doi.org/10.1016/j.jksuci.2018.01.003

[5] Pradhan, A., Bisoy, S. K., Kautish, S., Jasser, M. B., Mohamed, A. W. *(2022*). Intelligent Decision-Making of Load Balancing Using Deep Reinforcement Learning and Parallel PSO in *Cloud Environment. IEEE Access*, 10;76939-76952.

[6] Pradhan, A., Bisoy, S. K., and Sain, M. *(2022*). Action-Based Load Balancing Technique in Cloud Network Using Actor-Critic-Swarm Optimization, *Wireless Communications and Mobile Computing, Wiley, Hindawi*, 2022;6456242, pp 1-17.

[7] Hatem, M. E., Rabie, A. R. (2019). Resource Scheduling for Offline Cloud Computing Using Deep Reinforcement Learning. *International Journal of Computer Science and Network Security (IJCSNS),* 19(4);54-60.

[8] Tennakoon, D., Chowdhury, M., Luan, T. H. (2018). Cloud-based load balancing using double Q-learning for improved Quality of Service. *Wireless Networks*, https://doi.org/10.1007/s11276-018-1888-8.

[9] Li, M., Zhang, J., Wan, J., Ren, Y., Zhou, L., Wu, B., Yang, R., Wang, j. (2019). Distributed machine learning load balancing strategy in cloud computing services. *Wireless Networks*. https://doi.org/10.1007/s11276-019-02042-2.

[10] Noel, R. R., Mehra, R., Lama, P. (2019). Towards Self-Managing Cloud Storage with Reinforcement Learning. *IEEE International Conference on Cloud Engineering (IC2E)*, pp 34-44.

[11] Tassel, P.,Gebser, M.,Schekotihin, K. (2021). A Reinforcement Learning Environment for Job-Shop Scheduling.*arXiv:2104.03760*[cs. LG].

[12] Lin, J., Cui, D., Peng, Z., Li, Q., He, J. (2020). A Two-Stage Framework for the Multi-User Multi-Data Center Job Scheduling and Resource Allocation. *IEEE Access*, 8;197863-74.

[13] Che, H., Bai, Z., Zuo, R., Li, H. (2020). A Deep Reinforcement Learning Approach to the Optimization of Data Center Task Scheduling. *Hindawi Complexity, Wiley*, 2020;3046769, pp 1-12. https://doi.org/10.1155/2020/3046769

[14] Dong, T., Xue, f., Xiao, C., Li, J. (2020). Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. *Concurrency and Computation: Practice and Experience*. Wiley, pp 1-12.

[15] Pradhan, A., and Bisoy, S. K. (2022). Intelligent Action Performed Load Balancing Decision Made in Cloud Data center Based on Improved DQN Algorithm. *2022 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pp. 1-6.

[16] Peng, Z., Lin, J., Cui, D., Li, Q., He, J. (2020). A multi-objective trade-off framework for cloud resource scheduling based on the Deep Q-network algorithm. *Cluster Computing*. https://doi.org/10.1007/s10586-019-03042-9.

[17] Pradhan A., Bisoy, S. K. (2022). A novel load balancing technique for cloud computing platform based on PSO. *Journal of King Saud University – Computer and Information Sciences*, 34(7);3988-3995. https://doi.org/10.1016/j.jksuci.2020.10.016