Enhanced Quality of Service (EQoS)-Enabled Load Balancing Approach for Cloud Environment

Minakshi Sharma, Rajneesh Kumar, and Anurag Jain

Abstract Cloud computing uses the "pay as you go model" to provide on-demand services to its users especially data storage, computing power, network, and others. These services are provided to users without their direct active participation in managing resources. Cloud computing relies upon resource sharing to acquire coherence and economies of scale. Task scheduling in such an environment is used for the task execution on a suitable resource by considering some parameters and constraints to achieve performance. During high demand for these virtualized resources, efficient task scheduling achieves the desired performance criteria by balancing the load in the system. This paper presents a balancing mechanism by practicing task scheduling to increase performance in the cloud environment. It perceives various load balancing approaches based on task scheduling and concludes that their optimization goals are multi-objective. The presented mechanism is an extension of the previous proposed work QoS-enabled JMLQ [1]. This proposed approach has been tested in the CloudSim simulator, and results show that the proposed approach achieves better results in comparison with QoS-enabled JMLQ and its other variants in the cloud environment.

Keywords Cloud computing · Load balancing · JMLQ · CloudSim · Task scheduling

M. Sharma (✉) · R. Kumar
Department of Computer Science and Engineering, MMEC, Maharishi Markandeshwar Deemed to be University, Mullana, Ambala, Haryana 134002, India

R. Kumar
e-mail: drrajneeshgujral@mmumullana.org

A. Jain
School of Computer Science, Univesity of Petroleum and Energy Studies, Dehradun, Uttarakhand 248007, India
e-mail: anurag.jain@ddn.upes.ac.in

N. Marriwala et al. (eds.), *Emergent Converging Technologies and Biomedical Systems*, Lecture Notes in Electrical Engineering 841,
https://doi.org/10.1007/978-981-16-8774-7_44

1 Introduction

The advent of various technologies such as distributed computing, utility computing, autonomic computing, and the unveiling of service-oriented architecture has driven the growth of cloud computing. It points toward the goal to serve the user at a low cost without any expertise or deep knowledge of the system. The services provided by it are software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service(IaaS).

Scheduling of user requests is an important concept in the cloud environment, which includes a mechanism that maps a user request to an appropriate resource for the execution of a task. The performance of the system is directly affected by the efficiency of a scheduling technique. These algorithms are based on the management of physical and virtualized resources in the environment. In this paper, a two-level load balancing mechanism based on task scheduling has been discussed. The research work presented here is a load balancing approach that efficiently schedules tasks to virtual resources to increase the quality of service requirements. It is an incremental approach to our previous proposed work QoS-enabled JMLQ [2, 3]. Enhanced QoS-enabled JMLQ is an improved version of QoS-enabled JMLQ. This proposed algorithm is an attempt to override the random behavior of the algorithm.

2 Related Work

Scheduling of tasks is an important concept in the field of cloud computing, efficient scheduling of tasks not only can meet user requirements but can also improve resource utilization, response time, and other performance parameters, thereby balancing the load on the system [4, 5]. Task allocation is the task assigned to an appropriate resource that suits user requirements while the task scheduling algorithm settles the execution order of each task to be executed on the server [6, 7]. There are different task scheduling algorithm exists in the literature that balances the load on the system and enhances the performance of the system by considering different performance parameter. The following is the research work studied for task scheduling-based load balancing in the cloud environment.

In 2011, Yi Lu et al. proposed a load balancing approach JIQ based on the scheduling of the tasks. The main objective of this proposed approach was to execute the tasks in minimum response time by avoiding extra communication overheads during the process of task assignment. It is a two-fold load balancing mechanism that balances the load for a large system. It uses a data structure, i.e., I-queue that acts as a communication medium during task assignment to the processor. The first level of load balancing works for task assignment by probing the I-queues for idle server, and the load balancing at the second level balances the load by idle processors placement in any random I-queue attached with the dispatchers. The prime consideration factor was response time for this approach [8].

In 2013, X. Wu et al. proposed a QoS-driven task scheduling approach in which tasks were sorted based on the special attribute of tasks to decide precedence among the tasks. Afterward, the completion time of tasks evaluated on different services and the task was scheduled for a service according to the sorted queue based on minimum completion time [9].

In 2015, Babu and Samuel devised a technique to balance the load based on the foraging nature of honey bees. In their proposed technique the task with the lowest priority has the higher chances of being migrated from an overwhelmed virtual machine (VM) to the underwhelm virtual machine. Their algorithm relies on the priority of tasks in a waiting queue for VM. Simulated results demonstrate that for tasks that are limited in number makespan reduces, and there is a reduction in the total number of migrations needed for operation, which shows that the proposed algorithm has low scalability [10].

In 2016, H.E.D. Ali et al. proposed a grouped task scheduling approach for the cloud environment based on QoS parameters. The incoming tasks from the user side were categorized into five groups that depend upon task attributes. The categories are user type, task type, task length, and task latency. The scheduling was carried out in two steps. The first step decides the category to be scheduled based on the high value of the attributes of tasks. The second step was based on tasks within the chosen category to be scheduled first based on the minimum execution time of the tasks. The latency of tasks and execution time considered as a performance parameter [11].

In 2017, Ashish Garg et al. proposed a metaheuristic search technique based on ant colony optimization for solving task scheduling problems. The objective of this algorithm was to balance the load across the system by optimizing the makespan performance parameter. The algorithm attains the local optimal solution by using an ant colony algorithm and at last, a Pareto set of solutions was attained by applying non-domination sorting [12].

In 2018, Chunpu wang et al. devised a load balancing approach for low latency in the cloud environment. The scheduler used a technique to deal with an empty I-queue. The task is allocated to the minimum loaded server after searching for d servers that are randomly chosen. Every dispatcher attached to I-queue implements the same strategy for task allocation when it is met with an empty I-queue. Low latency for service time is considered as a performance parameter. Moreover to avoid the delay in performance, a semiclosed-form expressions were also derived [13].

3 System Model for Enhanced QoS-Enabled JMLQ

The system model for "enhanced QoS-enabled JMLQ" comprises a two-fold load balancing mechanism that consists of n parallel VMs having homogeneous configuration VM (VM_0, VM_1, VM_2, ……., VM_{n-1}) interconnected with some networking components. These VMs join the I-queues of the dispatchers (D_0, D_1, D_2, ……., D_{m-1}) considered in a fixed ratio to the number of VMs, i.e., $r = n/m$ (here, n is the number of VMs, m represents an array of dispatchers, and r represents a ratio that is

fixed between n and m). The tasks processed in the system are considered independent, and the rate of task arrival is considered general [14] To represent the current scenario, the G/G/c like queuing model has been considered to represent the large cloud system as there is high variability in the arrival process and service process of the tasks [15, 16]. The system model consists of the following units.

Dispatcher: Every dispatcher is responsible for scheduling tasks and acts as an independent scheduler. Each dispatcher in this proposed approach has its unique id and a specified limit up to which it can possess the VMs in an I-queue.

I-Queue: This is attached to the dispatcher unit and acts as a communication medium for allocating tasks. When a VM completes its task, it joins the I-queue.

Task allocation to VMs: The VMs presented in I-queues are responsible for executing tasks that are removed from I-queues after task allocation. The incoming task is allocated to the first VM present within the I-queue of the first dispatcher with ID D_0 if it is nonempty otherwise, it probes for the other minimum loaded VM in adjacent dispatchers I-queue having dispatcher ID in sequence.

VMs allocation to I-queues using d-limit: This proposed approach place an upper bound on each I-queue set by d-limit. After the task completion, VM is drifted toward the first dispatcher's I-queue having ID D_0, it joins the I-queue if I-queue length is less than d-limit otherwise, it probes for the next dispatcher I-queue in a sequential manner. At a medium load, the I-queues remain well occupied with a large number of VMs and these VMs are equally distributed among all the I-queues using d-limit.

At high load, the VMs present with in these I-queues are very less, most of the I-queues remain empty and the probability of getting an empty I-queue is very high. At this stage, if a VM gets idle, it always joins the first dispatcher I-queue with ID D_0 and the next incoming task allocated to it and task allocation continues for this I-queue until and unless it gets empty. At high load, the algorithm behaves like a centralized policy as an idle VM always join the first dispatcher I-queue with ID D_0. This nature of the algorithm helps in a further reduction in response time and to utilize the resource more efficiently. If all the dispatcher I-queues are empty, then the task will be allocated to the minimum loaded VM which is chosen among d random VMs and the algorithm behaves like a randomized algorithm (Table 1).

Table 1 List of variables used in pseudocode

Variable notation	Significance
n	The number of available VMs in the system
m	Represents the number of dispatchers based on a fixed ratio b/w VMs and dispatchers
dl	Represents dispatcher limit, i.e., maximum number of VMs an I-queue can possess
D_0	Represents a dispatcher with dispatcher ID D_0
I-queue Length	Represents the length of dispatcher I-queue that consists of a subset of VMs that are minimum loaded in the system
Waiting task queue length	The no. of tasks in the queue waiting to be allocated to VM
d	Represents any random number from 1 to the total number of VMs (n)
F	Represents a flag value can be 0 or 1
Disipatcher_ID	Represents any dispatcher identity number form 0–999

3.1 Pseudocode for Enhanced Qos-Enabled Join Minimum Loaded Queue (JMLQ)

```
EQoS enabled JMLQ():
  {
      •   n= A set of of available VMs {VM0, VM2, VM3, …..VMn-1}
      •   m= an array of diapatchers {D0, D1, D2, ….Dm-1}
      •   d-limit (dl) = n/m
      •   Make a batch of cloudlets equal to 2n

For (I =0 to n-1)
  {
      •  Allocate 2 cloudlets to I-th VM at one clock_time()
  }
•  Make a procedure call named VM_to_dispatcher_mapping for assigning
   VM to an I-queue of the dispatcher Whenever any VM completes the
   execution of the cloudlet.

While (there is an incoming cloudlet from a consumer in a data center)
{
            For (I = 0 to m-1)
             {
      •   Obtain the Ith dispatcher_ID
      If ( dispatcher D0 I-queue is non-empty)
      {
```

- *Allocate cloudlet to the first VM in the I-queue of dispatcher D_0*
- *After cloudlet allocation remove VM from I-queue of dispatcher D_0*
- *break;*

}

Else

{

if dispatcher_ID == Total-no-of-dispatcher -1

{

If the last dispatcher's I-queue length is Nil then

{

- *Randomly choose d VMs*
- *Find the one VM having waiting task queue length minimum among the d random VMs.*
- *The task will be allocated to a VM having waiting task queue length is smallest.*

}

}

}

- *Make a procedure call named VM_to_dispatcher_mapping for assigning VM to an I-queue of the dispatcher Whenever any VM completes the execution of the cloudlet.*

}

}

VM_to_dispatcher_mapping()

{

- *dispatcher D_0 is selected based on its ID*

If (D_0's I-queue length is less than d-limit)

{

- *VM joins the I-queue of dispatcher D_0 which have completed the cloudlet execution*

}

Else

{

Repeat

{

- *Flag = 0*
- *Idle VM searches the adjacent dispatcher ID in a sequence having I-queue length is less than d-limit*
- *On the successful joining of VM in an I-queue set Flag=1*

} until(Flag = 1)

}

}

4 Experimental Setup and Result

To prove the better optimization effects of performance parameters by practicing "enhanced QoS-enabled JMLQ," the authors selected some load balancing approaches based on task scheduling including the previous versions of the proposed approach JMLQ and QoS-enabled JMLQ [17, 3]. All these selected approaches have been tested and analyzed in the simulated environment. CloudSim (version 3.0.3) has been used for demonstrating the cloud environment. Eclipse IDE is used to

Table 2 Simulation environment configuration

Configuration Details	
Cloudlets of variable length	In-between (600,000–800,000)
Number of hosts	2500, each with four processing elements (Pe)
Number of virtual machines (VMs)	10,000
Different sets of cloudlets	In-between (60,000 – 260,000)
Storage size for each VM	10,000 MB
RAM for each VM	512 MB
Million instructions per second (MIPS)	1000
Bandwidth	1000
Cloudlet scheduler	CloudletSchedulerSpaceShared()
VM scheduler	VmSchedulerTimeShared()

develop and implement the algorithm using JDK 1.8. The following table represents the configuration details used in CloudSim (Table 2).

4.1 Validation of Results for Response Time

Figure 1 represents the comparison of response time for different distributed approaches versus the proposed approach. This proposed approach average response

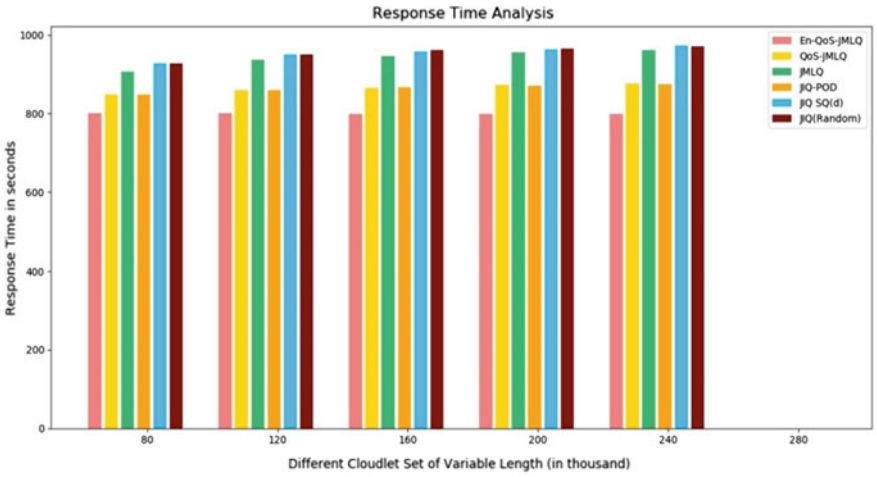

Fig. 1 Comparison of the response time for different distributed approaches with the proposed approach

time is 6.86% less than QoS-enabled JMLQ and JIQ-Pod, 14% less than JMLQ, 15.35% less than JIQ-SQ(d), and 15.53% less than JIQ-Random.

Figure 1 depicts that response time declines with some points as the number of cloudlets increases for the proposed approach, here, cloudlets represent the number of tasks. This proposed approach uses the dispatcher ID for joining a VM in an I-queue, therefore, all the VMs are placed in I-queues in a contiguous form without insertion of empty I-queues, also, these VMs are equally distributed among the I-queues based on d-limit. So an incoming task will always get a VM in an I-queue until these are present in the I-queues. This proposed approach overcomes the random behavior of the previous variants and eliminates the mapping of the incoming tasks with empty I-queues until and unless VMs are not present in the I-queues. After, a set of 160,000 cloudlets response time gets stable.

4.2 Validation of Results for Resource Utilization

Figure 2 represents the percentage of resource utilization for different distributed approaches in comparison with this proposed approach. It depicts from Fig. 2 that the proposed approach utilizes the resource more efficiently in comparison with its variants. The percentage of resource utilization is improved over QoS-enabled JMLQ

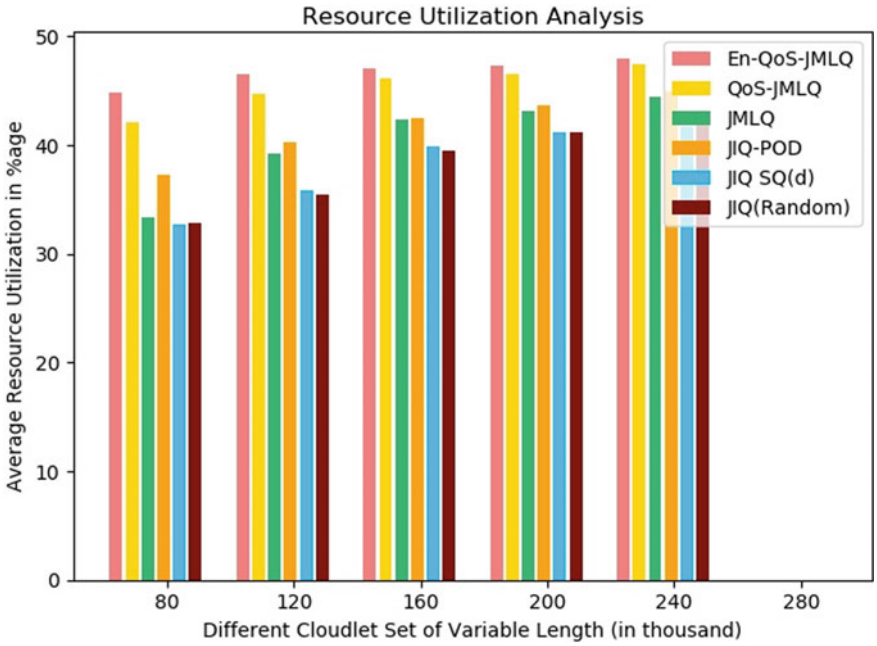

Fig. 2 Number of cloudlets versus percentage of resource utilization

by 2.55%, 16.51% better than JMLQ, 13.14% better than JIQ-Pod, 25.03% improved over JIQ-SQ(d), and 25.88% improved over JIQ-random. It is an important parameter to achieve an efficient load balancing policy that led to a decrease in cost for services from the customer side.

4.3 Validation of Results for Average Waiting Time

To determine the optimization of performance parameters for this proposed approach, next, parameter is considered as the average waiting time. The reduction in average waiting time represents a decrease in the waiting period to serve user requests. It can be analyzed from Fig. 3 that an increase in average waiting time with respect to increase in the number of cloudlets is not very significant in comparison with other distributed approaches. The decrease in waiting time observed 0.7% less than QoS-enabled JMLQ, 1.68% less than JMLQ, 15.02% less than JIQ-pod, 16.59% improved over JIQ-SQ(d), and 15.92% less than JIQ-random. These experimental results prove the efficacy of this proposed approach in comparison with other distributed approaches.

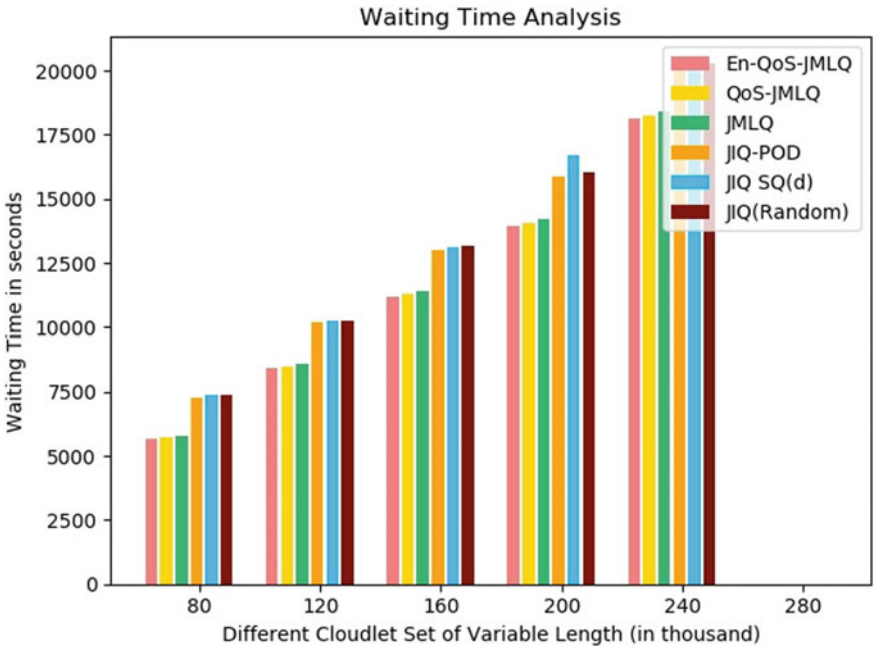

Fig. 3 Average waiting time versus no. of cloudlets

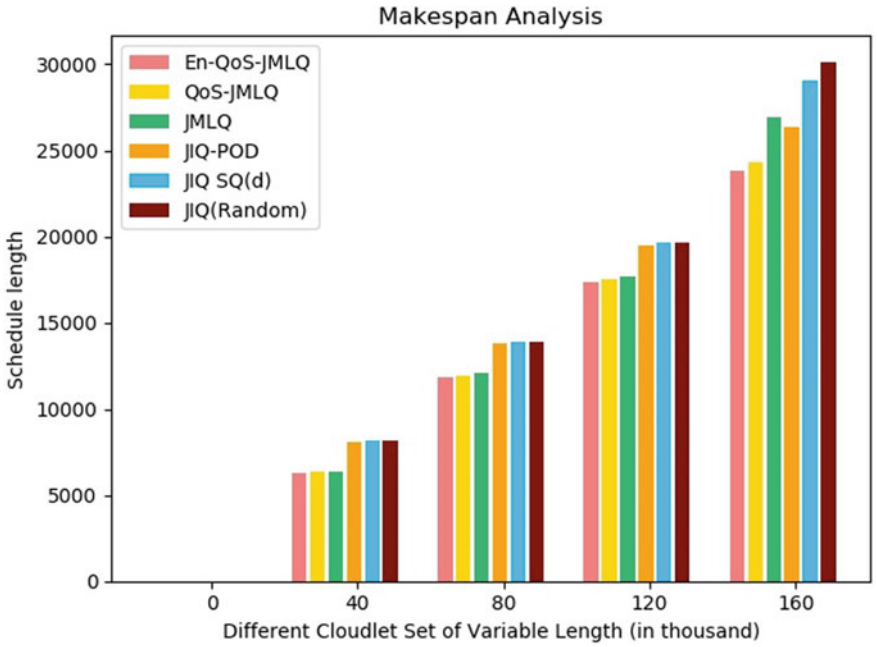

Fig. 4 Makespan versus a different set of cloudlets

4.4 Validation of Results for the Makespan

The next parameter considered to determine the effectiveness of the proposed approach on the scale of performance is makespan. Makespan is compared for the proposed approach for a set of 160,000 cloudlets of variable length in comparison with its variants. It was found a decrease in makespan by 1.04%, 4.56%, 13.74%, 16.55%, 17.47% from QoS-enabled JMLQ, JMLQ, JIQ-pod, JIQ-SQ(d), JIQ-random approaches respectively. Figure 4 depicts the decrease in the makespan of this proposed approach in comparison with other variants.

4.5 Statistical Analysis of Cloudlet Distribution

Equal distribution of tasks (cloudlets) among available resources is one of the most desirable features to achieve load balancing for an algorithm. It increases the stability of the system so cloudlet distribution among the available resources is one of the important parameters to determine the efficiency of an algorithm. Statistical analysis is done to determine the cloudlet distribution. It is based on the standard deviation that determines the variance of cloudlet distribution of the proposed approach in comparison with other distributed approaches. For the proposed approach, the coefficient of

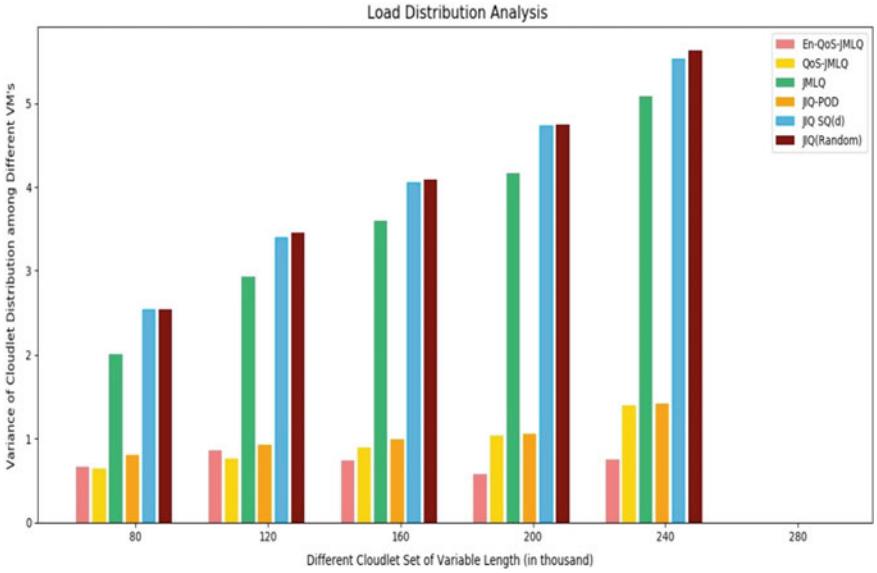

Fig. 5 Number of cloudlets versus the variance of cloudlets distribution

variation has a value less than one, which signifies the distribution of cloudlets with a low variance in comparison with other distributed approaches. Figure 5 represents the variation of the cloudlet distribution of this proposed approach in comparison with others.

5 Conclusion and Future Scope

This research work presented here is a load balancing approach based on task scheduling that efficiently balances the load in a cloud environment by optimizing the performance parameters. The experimental results have been validated and analyzed in a simulation environment developed using CloudSim. The objective of this research work is to achieve multi-dimensional performance parameters. This proposed algorithm minimizes the response time of user requests by utilizing the resource more efficiently by balancing the load in the system. As a future scope, enhanced QoS-enabled JMLQ work can also be extended by developing more hybrid methods during task allocation at high load to minimize response time by invoking intelligence during task allocation based on some intelligent information.

References

1. Sharma M, Kumar R, Jain A (2019) A system of quality of service enabled (QoS) join minimum loaded queue (JMLQ) for cloud computing environment. Patent Application, no. 201911039375, 51328, India
2. Sharma M, Kumar R, Jain A (2020) A proficient approach for load balancing in cloud computing-join minimum loaded queue: join minimum loaded queue. Int J Infor Syst Model Des (IJISMD) 11(1):12–36
3. Sharma M, Kumar R, Jain A (2021) A QoS enabled load balancing approach for cloud computing environment join minimum loaded queue(JMLQ): QoS enabled JMLQ. Int J Grid High-Perform Comput (IJGHPC) 4(14), Article 5
4. Sharma M, Kumar R, Jain A (2019) Implementation of various load-balancing approaches for cloud computing using CloudSim. J Comput Theor Nanosci 16(9):3974–3980
5. Sharma M, Kumar R, Jain A (2021) Load balancing in cloud computing environment: a broad perspective. Intelligent data communication technologies and internet of things. Sringer science and business media, India
6. Åström E (2016) Task scheduling in distributed systems. Model and prototype
7. Mishra SK, Sahoo B, Parida PP (2020) Load balancing in cloud computing: a big picture. J King Saud Univ Comput Inform Sci 32(2):149–158
8. Lu Y, Xie Q, Kliot G, Geller A, Larus JR, Greenberg A (2011) Join-Idle-queue: a novel load balancing algorithm for dynamically scalable web services. Perform Eval 68(11):1056–1071
9. Wu X, Deng M, Zhang R, Zeng B, Zhou S (2013) A task scheduling algorithm based on QoS-driven in cloud computing. Proc Comput Sci 17:1162–1169
10. Babu KR, Samuel P (2015) Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud. Innovat Bio-Insp Comput Applic 424:67–78
11. Ali HGEDH, Saroit IA, Kotb AM (2017) Grouped tasks scheduling algorithm based on QoS in cloud computing network. Egypt Inform J 18(1):11–19
12. Gupta A, Garg R (2017) Load balancing based task scheduling with ACO in cloud computing. In: Proceedings of international conference on computer and applications IEEE (ICCA), pp 174–179
13. Wang C, Feng C, Cheng J (2018) Distributed Join-the-Idle-Queue for low latency cloud services. IEEE/ACM Trans Network 26(5):2309–2319
14. Abraham GT, James A, Yaacob N (2015) Group-based Parallel Multi-scheduler for grid computing. Futur Gener Comput Syst 50:140–153
15. Bramson M, Lu Y, Prabhakar B (2010) Randomized load balancing with general service time distributions. ACM Sigmet Perform Eval Rev 38(1):275–286
16. Atmaca T, Begin T, Brandwajn A, Castel-Taleb H (2015) Performance evaluation of cloud computing centers with general arrivals and service. IEEE Trans Parallel Distrib Syst 27(8):2341–2348
17. Sharma M, Kumar R, Jain A (2019) A system of distributed join minimum loaded queue (JMLQ) for load balancing in cloud environment. Patent Application, no. 201911007589, pp. 12780, India