Cluster Computing (2024) 27:11427–11454
https://doi.org/10.1007/s10586-024-04538-9

Fault-tolerant allocation of deadline-constrained tasks through preemptive migration in heterogeneous cloud environments

Medha Kirti[1] · Ashish Kumar Maurya[1] · Rama Shankar Yadav[1]

Received: 19 October 2023 / Revised: 9 April 2024 / Accepted: 26 April 2024 / Published online: 27 May 2024

Abstract
In recent years, the occurrence of task failures are becoming prevalent in cloud computing due to various factors such as the increasing complexity of cloud environments, heterogeneity of resources, resource limitations and inadequate allocation. Task failure due to insufficient allocation poses a significant challenge in cloud computing. When tasks are not allocated effectively, they may not be completed within their deadlines which ultimately leads to failure. Hence, effective allocation strategies combined with appropriate fault tolerance measures are vital for addressing these challenges and mitigating the risk of task failures. This paper proposes a fault-tolerant task allocation algorithm (FTTA) for independent tasks with deadline through preemptive migration in heterogeneous cloud environments to reduce task failure. The proposed algorithm involves three phases: the initial phase decides the priority of tasks in the ready list to minimize the execution time and meet task deadlines, the second phase includes the selection of a suitable virtual machine with minimum execution time and the last phase assigns task on available or non-available (which may available in future) virtual machines to find the best execution time within the deadline limit. During the task allocation process, the algorithm adopts fault-tolerant strategy that includes preemptive migration if necessary which allows the migration of tasks to identify the best suitable virtual machine. An analysis of the proposed algorithm reveals that the overall time complexity is $O(n \log n + nm^2)$ where n is the number of tasks and m is the number of virtual machines. Further, the performance of the algorithm is evaluated for different sets of tasks (small to large) while varying the number of virtual machines. The experimental results demonstrate that FTTA outperforms First Come First Served (FCFS), Priority based algorithm, Shortest Job First (SJF), Dynamic Maximum Minimum (Dy max min) and RADL algorithms in terms of number of rejected tasks, makespan, speedup and efficiency.

Keywords Task allocation · Task failure · Fault tolerance · Preemptive migration · Cloud computing

1 Introduction

Cloud computing has become exceptionally popular and in high demand because of its services like higher scalability, higher availability, accessibility, cost efficiency and rapid deployment of resources [1]. Cloud computing provides services in two primary forms: software applications and hardware infrastructure. These services are offered through a pricing model where users pay based on their actual usage and are accessible over the Internet [2]. Cloud provides three different services, such as Infrastructure as a Service (IaaS), Platform as a service (PaaS), and Software as a Service (SaaS) [3] as shown in Fig. 1. IaaS provides all the infrastructure services like storage, servers, processing power, etc. PaaS offers a development environment to cloud users for deploying and managing applications, while SaaS provides software to the cloud user that can be used directly from the cloud. Cloud users can demand resources from the cloud service provider at any time [4]. The cloud provider assesses these requests and selects the best

✉ Ashish Kumar Maurya
ashishmaurya@mnnit.ac.in

Medha Kirti
medha.2020rcs10@mnnit.ac.in

Rama Shankar Yadav
rsy@mnnit.ac.in

[1] Department of Computer Science and Engineering, Motilal Nehru National Institute of Technology Allahabad, Prayagraj 211004, India

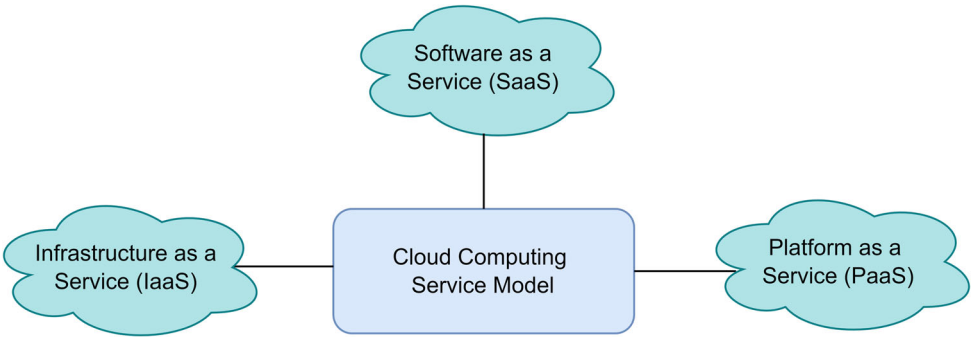

Fig. 1 Cloud service model

resource that fulfills the user's predetermined deadline and other constraints. The user then effortlessly receives the desired on-demand services from the cloud resource broker [5].

In order to meet the requirements of cloud users, the service provider introduces various methods including task allocation, fault tolerance, resource monitoring, load balancing etc. in cloud computing. These methods are essential for efficient utilization of resources in cloud computing, which offers multiple advantages such as increased scalability, cost-efficiency, and flexibility.

As the number of cloud users continues to grow steadily, the volume of tasks is experiencing exponential growth while the number of available virtual machines (VMs) remains constant. One of the main problems in cloud computing is allocation of such a huge number of tasks with varying lengths and deadlines among virtual machines with different computation power. Inefficient allocation may lead to task failure which minimizes the overall performance of the system. Hence, task failure is one of the significant challenges in meeting the requirements of cloud users. Task failure occurs in the system due to several reasons, such as hardware or software issues, resource constraints, poor allocation, network problems, or even due to the failure of the cloud provider's infrastructure. Efficiently allocating tasks is a key strategy for mitigating task failure in cloud computing environments. Task allocation is the process of assigning deadline-based tasks to suitable virtual machines (VMs) based on their respective deadlines within a given set of VMs [6, 7]. The goal of task allocation is to improve the performance of the system, typically measured in terms of makespan, speedup, and other relevant metrics. When tasks are not assigned to appropriate resources or not scheduled efficiently, it can lead to various issues, such as resource contention, missed deadlines, task starvation, insufficient resource utilization, etc., leading to the failure of deadline constrained tasks. Hence, an efficient fault tolerant task allocation strategy is required to allocate maximum tasks to the virtual machine [8]. Ensuring fault tolerance of tasks in a cloud environment is another challenge that cloud service providers and users must address to maintain cloud service performance [9]. Fault tolerance allows the system to persist in its functionality even in the presence of failures or errors. It guarantees that services and applications are accessible and dependable even in the presence of partial failure in cloud computing. Fault tolerance in cloud computing can be achieved using various strategies such as reactive, proactive, and adaptive approaches [10]. Reactive approaches are based on conventional fault tolerance approaches in cloud computing. It includes replication, checkpointing, retry, message logging, task resubmission, etc. Proactive approaches reduce the chances of system failure by predicting the failure in advance. software rejuvenation, self-healing, preemptive migration, load balancing, prediction and monitoring come under proactive approach. The adaptive approach includes machine learning and fault induction which are like proactive approaches except their ability to learn and adapt. The system learns and adapts the changes based on artificial intelligence and machine learning in adaptive approach.

In this paper, we consider preemptive migration to provide fault tolerance while allocating tasks to suitable virtual machines. Preemptive migration is a proactive task allocation strategy in cloud computing. It mitigates potential issues, such as resource constraints and task failures, by relocating tasks from one virtual machine (VM) or computing resource to another before any failures occur. By relocating tasks from VMs, preemptive migration enhances fault tolerance and minimizes the risk of task failures in task allocation process. Various scheduling algorithms have been developed in the literature for task allocation in cloud computing environments. These include traditional approaches like First Come First Served (FCFS) [11], Priority based algorithm [12], Shortest Job First (SJF) [13], Dynamic Maximum Minimum (Dy max min) algorithm [14] etc. However, these conventional algorithms exhibit certain limitations, such as inefficient utilization of resources, high task rejection, longer makespan, etc. which hinder their overall performance. To address these limitations, there is a need of an efficient fault tolerant task allocation approach which efficiently utilizes the resources,

reduces makespan, minimizes number of rejected tasks, improves speedup, efficiency and enhances overall performance. Hence, we propose a fault tolerant task allocation algorithm for heterogeneous cloud computing that minimizes task failure. The proposed algorithm combined task allocation with a proactive fault tolerance strategy i.e., preemptive migration. The algorithm allows tasks to be allocated on available and non-available virtual machines. The primary objectives are to minimize task rejections, reduce makespan [15], and enhance speedup and efficiency. The performance of the proposed task allocation approach is evaluated based on some existing task allocation algorithms such as FCFS, SJF, Priority-based, Dy max min and RADL [16] algorithms. Contribution of paper includes:

1. We propose a novel fault-tolerant task allocation algorithm for deadline constrained tasks in a heterogeneous cloud environment. It uses a proactive fault tolerant approach i.e., preemptive migration which migrates tasks from one virtual machine to another to reduce the number of rejected tasks.
2. The proposed algorithm allocates tasks to available and non-available virtual machines, which minimizes the number of rejected tasks, makespan, and enhances speedup and efficiency. It uses preemptive migration for allocating maximum tasks to virtual machines.
3. We demonstrate the working of the proposed algorithm for task allocation through illustrative examples.
4. We analyze the time complexity and perform simulations to evaluate the effectiveness of the proposed algorithm for different sets of tasks while varying virtual machines.
5. We evaluate the FTTA performance with state-of-the-art algorithms such as FCFS, SJF, Priority, Dy max min and RADL algorithms in terms of rejected task, makespan, speedup and efficiency. The proposed algorithm shows an improved performance as compared to state-of-the-art algorithms.

The rest of the paper is organized as follows: Sect. 2 discusses the existing work of the task allocation algorithms. Section 3 presents the system model, application model and formulation of the problem. Section 4 shows the proposed task allocation algorithm, time complexity analysis and an illustrative example. Section 5 provides simulation results and analysis of the proposed algorithm. Section 6 provides the conclusion of the paper along with future works.

2 Related work

This section presents a literature review of deadline constrained task allocation algorithms. Various algorithms have been introduced in the existing literature to provide the solution of task allocation to a suitable virtual machine with minimum task rejection and makespan in heterogeneous cloud computing. Each of these algorithms has its advantages and disadvantages. In this context, we delve into several task scheduling methods [17–23] associated with deadline-based task scheduling within the domain of cloud computing.

Nayak et al. [18] presented a scheduling algorithm based on backfilling concept for deadline sensitive tasks to reduce the task rejection in cloud computing. The algorithm schedules tasks based on their arrival time, start time and deadline. The proposed algorithm in this work tried to resolve the conflict of existing backfilling algorithm. Dubey et al. [19] discussed a hybrid task scheduling algorithm based on chemical reaction optimization and PSO to minimize the makespan and average execution time. The algorithm works in two phases. In the first phase, modified chemical reaction optimization is used, while modified partial swarm optimization maps tasks on virtual machines. In [24], authors presented a task scheduling algorithm for deadline based tasks in two phases. In the first phase, a global scheduling technique, i.e., Enhanced Ant Colony Optimization is introduced, which allocates the cloud task to the appropriate VM. In the later phase, cloud tasks are rearranged in the waiting queue of the virtual machine to enhance the completion rate. The authors in [25] discussed allocation algorithm which allocates deadline constrained independent tasks to resources at runtime in cloud. The algorithm schedules the tasks to the virtual machine with minimum competition time. The completion time of virtual machine in the proposed algorithm is computed based on execution time of current tasks and previous load assigned. Zhang et al. [26] proposed a deadline constrained allocation algorithm to reduce the finish time of tasks. It introduced working in two steps i.e., deciding the priority of parallel tasks and mapping virtual machine based on relative distance. The priority of parallel tasks is calculated using downward rank and max slack values. Kumar et al. [27] discussed a deadline constrained dynamic task scheduling algorithm with elasticity property to minimize the makespan time. Authors in this work discussed three algorithms for task sorting, task allocation to VMs and load balancing. The task sorting algorithm sorts the tasks based on deadline value. The sorting of tasks

is done on the basis of shorter deadline values. In task scheduling, the highest priority tasks are scheduled on a virtual machine with minimum execution time and deadline constraints. The authors also gave the concept of elasticity for the tasks which do not meet the deadline. In [28], authors proposed a task scheduling algorithm based on the deadline and budget to reduce the execution time and cost of executed tasks. The algorithm works in two levels for mapping the tasks to suitable resources. The first level infers the type of task constraint and based on the type; tasks are assigned to suitable clusters. In the second level, the algorithm finds the requirement of tasks based on task attributes defined in the paper. Nabi and Ahmed in [29] discussed a deadline based task scheduling algorithm using modified PSO to enhance resource utilization and reduce task rejection. The algorithm optimizes the performance by meeting the deadline and minimizes the execution time. The performance results show that the algorithm outperforms existing task scheduling algorithms for utilization of resources, makespan, response time of tasks, meeting deadline, and total cost of execution. In [30], the authors evaluated performance of different scheduling algorithms for deadline constrained tasks in cloud computing. The authors concluded that chicken swarm optimization outperforms other algorithm on Montage and CyberShake workflows. Sahoo et al. [31] discussed deadline based task scheduling as a bi-objective minimization problem to reduce the makespan. The authors introduced a Scheduling algorithm based on LA (LAS) for solving the bi-objective minimization problem. This algorithm incorporates reinforcement learning to determine the allocation of tasks to VMs. Authors in [32] discussed an approach for allocating independent tasks having deadline constraints to VMs. The approach uses an optimization method to map tasks efficiently. The authors introduced two methods of allocation i.e., Linear Weighted Sum technique and Ant colony. The proposed approach minimizes task rejection and makespan. Nabi et al.[16] proposed a dynamic deadline aware task scheduling algorithm to minimize the makespan and task rejection. This algorithm schedules the task according to the minimum finish time for the task. The completion time considered here includes the execution time of tasks and the current load on the VMs. The tasks are scheduled if they satisfy the deadline constraints. The authors discovered that this algorithm cannot facilitate workflow-based dynamic task scheduling.

Fault tolerant task allocation is widley used nowadays to minimize the task failures. Various fault tolerance strategies have been proposed for task allocation such as replication, resubmission, checkpointing prediction etc. The authors in [33] discussed a fault tolerant scheduling approach which maps tasks to virtual machine within deadline even in the presence of failure. The authors have introduced two approach such that replication and resubmission to ensure fault tolerance while mapping tasks. Replication maintains several backups of tasks and resubmission re allocates the tasks to different virtual machine when fault occurs. The authors in [34] proposed fault tolerant task allocation algorithm using genetic algorithm. The algorithm schedules tasks to virtual machine in three phases including virtual machine selection phase, local phase and global phase. The proposed algorithm again finds the optimal virtual machine in case of failure and allocates the scheduled tasks to another suitable VM. Malik et al. [35] discussed a task scheduling algorithm based on grey wolf algorithm and ant lion. The algorithm provides tolerance against host failure during task processing time using lively standby replication. To address host failures, the checkpoint strategy is employed as a rollback mechanism. Xu et al. [36] proposed best fit decreasing algorithm for task allocation based on greedy approach. The algorithm assumes that all failures are independent and handles only failure in cloud datacentre. Authors in [37] proposed a proactive fault tolerant approach for task allocation. The proposed approach includes two stages including task prediction and task allocation. In the first stage, tasks are classified into failure prone and non-failure prone tasks based on prediction model. In later phase, the failure and non-failure prone tasks are scheduled separately. The fault tolerance is provided using replication strategy. Authors in [38] discussed allocating independent tasks on heterogeneous and distributed resources in cloud environments. The authors introduced a population-based approach based on an enhanced differential evolution algorithm to assign tasks in suitable virtual machines. The authors in [39] discussed the task allocation approach in two stages. In the first stage, the tasks are prioritized based on the arrival time and allocated on the idle virtual machines. The failed tasks are detected and rescheduled in the second stage based on the proposed fault tolerance strategies. The authors concluded that the proposed approach is effective and provides a better solution to workload issues in cloud computing. Nanjappan et al. [40] introduced an algorithm for allocating tasks to VMs and ensuring the balanced utilization of each VM. The proposed approach first categorizes virtual machine into underloaded, balanced and overloaded. To reduce the probability of task failure, higher priority tasks are preempted from the overloaded VMs and checkpointing is used to save the current state. In the algorithm, lightweight tasks are allocated to virtual machines with high CPU utilization, whereas other tasks are given to low CPU utilization.

Li et al. [2] discussed a fault tolerant algorithm for scheduling deadline based tasks to virtual machine to minimize the cost and ensure reliability. The algorithm works in two stages including fault tolerant allocation of

tasks and virtual machine adjustment. Fault tolerance in the algorithm is incorporated using primary backup replication technique. Authors in [41] discussed a comprehensive cloud architecture designed to efficiently handle failures both before and after their occurrence. This architecture integrates various fault tolerance strategies, including fault prediction, migrations, virtual machine management, and load balancing. To further enhance fault tolerance, the authors implemented a prediction-based load balancing approach. This approach utilizes a heuristic model that incorporates restore techniques and checkpoints, supported by a statistical model that predicts the optimal checkpoint interval and monitors price fluctuations for efficient resource allocation. Sheikh et al. [42] discussed a fault tolerant model for resource allocation for tasks in grid environment. The tasks are allocated to the resources having least execution time and least workload history. Authors in [43] presennted a scheduling and checkpointing algorithm for tolerating Byzantine faults. This algorithm efficiently detects failed virtual machines and initiates task migration to another operational virtual machines. The algorithm also initiates migration of entire batch of tasks that comprise the job to minimize the overhead. Saidi and Bardou [44] discussed a state-of-the-art literature on resource allocation in cloud computing. The paper focussed challenges of resource allocation, especially task scheduling and virtual machine placement. The paper also identify the factors influencing energy consumption in resource allocation (RA) and investigate potential optimization strategies. Authors in [40] proposed a task scheduling algorithm aims to optimize fault tolerance, response time, efficiency, and makespan. The algorithm enhances fault tolerance capability by assigning tasks to suitable resources according to peak resource loads. The algorithm assigns the lightweight tasks to the resources with high CPU utilization and the computation-intensive tasks to the resources with low CPU utilization. Haidri et al. [45] introduced a receiver-initiated, deadline-aware load balancing strategy which aimed at migrating incoming cloudlets to suitable virtual machines (VMs) that can meet their deadlines. The strategy optimizes turnaround time by utilizing the remaining processing capacities of VMs. Yao et al. [46] presented a fault tolerant scheduling algorithm of deadline constrained independent tasks. The algorithm provides fault tolerance based on replication and resubmission to enhance the resource utilization. The proposed algorithm consists of two phases including scheduling and resource provisioning. The first phase allocates tasks to suitable resources based on deadline while later phase adjusts the resource accordingly.

In the literature, various algorithms have been discussed for task allocation of independent tasks with deadline. The literature also includes fault tolerant allocation of tasks in cloud computing. These algorithms are evaluated based on various metrics, including the number of rejected tasks, makespan, speedup, efficiency, throughput, and resource utilization. We can observe that most allocation algorithms experience challenges such as an increase in the number of rejected tasks, a longer makespan, fault tolerance. Very few research have been explored on fault tolerance using preemptive migration. Furthermore, none of the existing algorithms considered all the parameters like number of rejected tasks, makespan, speedup and efficiency simultaneously for performance evaluation. Our approach introduced a fault tolerant alllocation of tasks using preemptive migration, including all the performance metrics to evaluate overall performance. Table 1 presents a summary of various contributions made by the authors, along with the problems they have addressed.

3 System model and problem definition

This section includes system model and defines the task allocation problem. Table 2 presents basic notations and their respective definitions that have been used in the proposed task allocation algorithm.

3.1 Cloud computing model

A cloud computing model consists of a cloud broker and cloud provider. Whenever a user submits a request, the broker processes tasks, including task prioritization, resource allocation, scheduling and submits the request to the nearest datacenter in the cloud provider. A cloud datacenter consists of a finite number of hosts and each cloud host includes m number of heterogeneous virtual machines VM = (VM_1, VM_2,.....VM_m). The virtual machines are heterogeneous in terms of processing speed, execution time and computational capabilities. The computational power of the virtual machine is represented in MIPS (Million Instruction Per Second). The communication latency in the virtual machine is considered negligible. Since all tasks are scheduled in the same data center, there is no need for virtual machine communication. It is considered that initially, tasks are allocated to virtual machines based on the earliest ready time. Figure 2 illustrates the cloud computing model, in which users submit tasks to a cloud broker. Cloud broker acts as an intermediate between cloud users and cloud service providers. When the cloud broker gets these tasks, it analyzes the constraints and requirements of tasks regarding allocation, resource utilization etc. The cloud broker then communicates with the cloud provider to execute the tasks on the appropriate virtual machines.

Table 1 Summary of existing works

References	Contributions	Pros	Cons
[18]	• Scheduling problem based on maximizing the task acceptance ratio and minimizing the task rejection ratio	Improved lease acceptance ratio	Limited no of virtual machines and leases, with no consideration given to the switching cost of the VM
[19]	• Hybrid task scheduling algorithm aimed at addressing multi-objective optimization objectives, including minimizing computation cost, makespan time, average execution time, and energy consumption	The algorithm generates optimal solutions in lesser execution time, cost, and makespan	Not considered, task rejection ratio, load balancing, and turnaround time parameters as performance evaluation
[24]	• Ant Colony Optimization approach incorporating deadline constraints to achieve a near-optimal task scheduling scheme that balances energy consumption and task completion rates	Reduced makespan and energy consumption	Inconsistent results
[25]	• The proposed approach minimizes the load imbalance issue, support deadline-based tasks, and improve the overall Cloud performance	Enhance load balancing, and improve the overall performance gain	Task rejection can be reduced
[26]	• Introduced efficient priority and relative distance algorithm designed to minimize task scheduling duration for precedence-constrained workflow applications while ensuring compliance with end-to-end deadline constraints	Minimizes makespan	Lacks optimal resource utilization or workload balance
[27]	• A scheduling algorithm has been devised leveraging the concept of the last optimal k-interval, which ensures workload balance across all virtual machines through elastic resource provisioning and deprovisioning	Provides better elasticity and reduce the rejection ratio of task in comparison to FCFS, SJF and min-min algorithms	Some important QoS parameters are not considered like the cost for ensuring the high priority requests
[28]	• The paper presents the deadline budget scheduling model, where users submit their tasks along with associated budgets and deadlines to the data centres	Minimizes makespan and monetary costs	Not considered parameter like number of rejected tasks for performance evaluation
[29]	• An enhanced and adaptive dynamic load-balancing scheduler for non-preemptive, independent, and compute-intensive tasks in the cloud workload is proposed which aims to achieve lower task execution times, improved resource utilization, reduced task rejection, and minimized response times	Minimizes makespan, resource utilization, task response time	Simulated on limited number of hosts and vms
[30]	• This paper assessed the effectiveness of four algorithms, two are heuristic algorithms-ICPCP and SCS-while the other two are meta-heuristic algorithms-PSO and CSO in cloud computing environments	Compared heuristic and meta heuristic scheduling algorithms	Very limited research
[31]	• Introduced a scheduling algorithm based on learning automata for tasks with time constraints in cloud environments	Minimize energy consumption and makespan	Rejection ration of tasks can be minimized
[32]	• Introduced two methods that prioritize energy efficiency and makespan optimization when scheduling independent tasks with deadlines in the Cloud	Reduces energy consumption, Improves scheduling success	Not considered QoS-aware scheduling for workflow applications
[33]	• Introduced a fault-tolerant mechanism that strategically and dynamically selects between traditional resubmission and replication schemes	Achieves both fault tolerance and resource utilization efficiency	Static threshold Load imbalance Accuracy can be improved
[34]	• The proposed method utilizes the fundamental principles of Genetic Algorithms for optimal scheduling of VMs and tasks based on multi-user requirements	Efficient energy consumption, cost, utilization, execution time	Load imbalance

Table 1 (continued)

References	Contributions	Pros	Cons
[35]	• Introduced scheduling algorithm along with a lively standby replication (LSR) strategy, has been proposed to enhance the cloud computing paradigm	High throughput Less makespan	Chances of false prediction
[36]	• Introduced a heuristic algorithm, known as the Greedy-based Best Fault Tolerance (FT) Decreasing Scheduling Algorithm to meet users' Quality of Service (QoS) requirements	Fault tolerant scheduling In multiple geographical Regions	Possibility that VM requests fail will increase greatly
[37]	• Initially, a machine learning-based prediction model is trained to categorize incoming tasks as either "failure-prone" or "non-failure-prone" based on the predicted failure rate. Subsequently, two effective scheduling strategies are introduced to assign these task types to the most suitable hosts	Achieves better fault tolerance and reduces total energy consumption better than the existing schemes	Chances of inaccurate prediction result
[38]	• A population-based method is suggested to assign tasks to appropriate resources, aiming to minimize the overall time cost and minimizes makespan	Better performance in terms of makespan, convergence, and load balance	Experimented with limited number of tasks
[39]	• Designed a fault-tolerant mechanism to reschedule tasks upon the detection of task failure	Fault tolerant scheduling, Better performance	The accuracy and reliability of fault detection mechanisms may vary depending on the complexity of the system
[40]	• Introduced a hybrid firebug and Tunicate Optimization (HFTO) algorithm for task scheduling and load balancing in the cloud	Higher load balancing efficiency and improved cloud task scheduling performance	Overhead of cpu utilization computation
[41]	• Implemented a prediction-based load balancing approach. This strategy integrates a heuristic model featuring restore techniques and checkpoints, complemented by a statistical model	Reduces the execution time	No fault masking
[42]	• Introduced a fault-tolerant model for allocating resources to tasks within a grid environment	Better performance	Work can be further evaluated by real experiments in the dynamic grid environment
[43]	• Introduced an algorithm which identifies failed virtual machines and triggers the migration of tasks to other operational virtual machines	reduces fault tolerance overhead exponentially	Results can be improved by integrating fault detection with fault tolerance
[44]	• The main aim of this study is to review the current literature to gain understanding into the existing methods, strategies, and algorithms utilized for task scheduling and virtual machine (VM) placement	Review of Task scheduling and VM placement to resource allocation	Limited collection from 2016 to 2023
[45]	• The algorithm focuses on migrating incoming cloudlets to appropriate virtual machines (VMs) capable of meeting their deadlines	Better performance	Migration overhead
[46]	• The algorithm integrates fault tolerance through replication and resubmission strategies to improve resource utilization. It comprises two main phases: scheduling and resource provisioning	High resource utilization	Performance degradation of VMs due to migration

3.2 Application model

In this work, we represent the application model using $n * m$ cost computation matrix where n is total number of application tasks i.e., T= (T_1, T_2.....,T_n) having different execution time on m number of virtual machines VM = (VM_1, VM_2,.....VM_m). Each task is considered independent, non-preemptive and deadline constrained. The number of tasks is considered more than the number of virtual machines ($n \gg m$). Tasks are ordered according to their arrival time. It is assumed that only one task is allowed to run on the available virtual machine at a particular time. Each task is prioritized based on criteria such as execution time and deadline. The deadline of the tasks are represented by DT_i for task T_i; $1 \leq i < n$. For allocation of tasks on virtual machines, tasks must meet the deadline.

Each task is quantified in terms of its computational complexity, measured in Million Instructions (MI). The execution time [31] $ET(T_i, VM_j)$ for task T_i on virtual machine VM_j with computation power in Million Instructions per Second (MIPS) is computed using Eq. 1.

$$ET(T_i, VM_j) = \frac{LT_i}{CP_j} \tag{1}$$

where, LT_i represents the length of task T_i, measured in million instructions (MI), while CP_j corresponds to the computation cost of virtual machine VM_j, measured in million instructions per second (MIPS). The cost computation matrix calculates the execution time for tasks based on Eq. 1. For example, consider a cost computation matrix with 11 tasks and three virtual machines. The length of task T_0 is assumed to be 1200 MI and the computation power of virtual machines VM_1, VM_2, VM_3 are 600, 300 and 400 respectively. The execution time of each task is computed

Table 2 Notations and their meaning

Notations	Description
n	Number of tasks
m	Number of virtual machines
T_i and VM_j	i^{th} task and j^{th} virtual machine
LT_i	Length of task T_i
CP_j	Computation power of virtual machine VM_j
DT_i	Deadline of task T_i
PT_i	Priority of task T_i
RT_i	Rank of task T_i
$\overline{ET_i}$	Average execution time of task T_i
ET(T_i, VM_j)	Execution time of task T_i on VM_j
T_list	List of tasks
VM_list	Virtual machines list
ready list	Sorted list of tasks after priority assignment
VM_list_j	List of j^{th} virtual machine
Tl_j	Last task assigned on VM_j
$DT(Tl_j)$	Deadline of last task assigned on VM_j
RJ_i	Rejected list of task T_i

Table 3 Cost computation matrix of tasks on each virtual machine

Tasks	VM_1	VM_2	VM_3
T_0	2	4	3
T_1	4	3	5
T_2	12	7	8
T_3	10	9	11
T_4	3	2	4
T_5	7	8	9
T_6	2	7	3
T_7	4	3	5
T_8	8	12	7
T_9	7	15	8
T_{10}	5	4	6

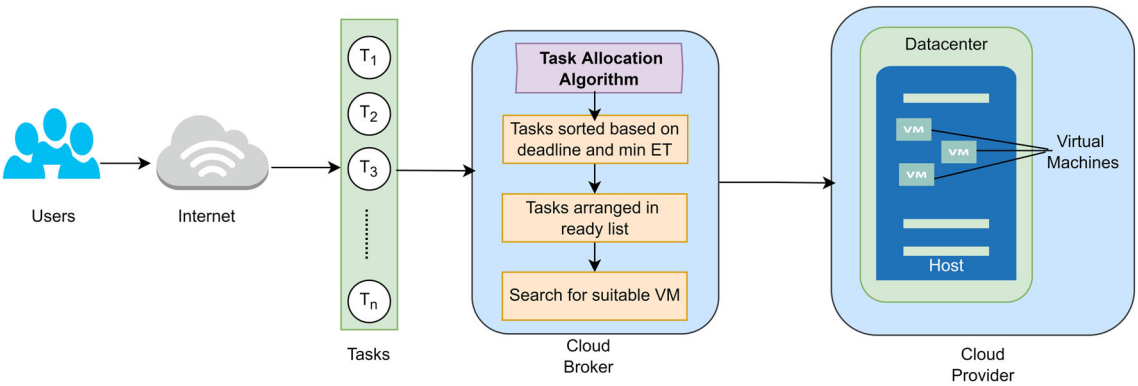

Fig. 2 Cloud computing model

in the cost computation matrix as shown in Table 3 using Eq. 1.

3.3 Problem formulation

In this subsection, we present some definitions that attribute characteristics to deadline constrained task allocation. Subsequently, we articulate the problem of task allocation.

Definition 1 (Average execution time: $\overline{ET_i}$). It is computed by taking the mean of the execution time of tasks on each virtual machine. Average execution time of task T_i on virtual machine VM_j is defined using Eq. 2.

$$\overline{ET_i} = \frac{1}{m} * \sum_{j=1}^{m} ET(T_i, VM_j) \tag{2}$$

Definition 2 (Actual Completion Time: ACT_i) It is defined as the finish time of the last task T_i completed by some assigned virtual machine.

Definition 3 (*Rank*) The rank of a task is a parameter that signifies its relative priority in a ready list. A lower rank implies higher priority or an earlier execution order.

Definition 4 (Deadline: DT_i) Deadline for task T_i is defined as the time constraint that determines when a particular task should finish.

Further, our problem can be formulated as follows. Considering a set of n independent tasks T =(T_1, T_2.....,T_n) in a cloud system arriving at time 0 and m heterogeneous virtual machines VM = (VM_1, VM_2,.....VM_m). Each task T_i has a set of attributes such as $T_i \in <LT_i, DT_i>$ where LT_i denotes the length of task T_i and DT_i represents the task's deadline where $(i \in 1,...,n)$. We compute the execution time of tasks based on the varying computation power of virtual machines. Our aim is to map maximum tasks to virtual machines while meeting deadline constraint of tasks and ensuring fault tolerance. We consider three task allocation aspects: task priority computation, virtual machine selection and task allocation. The main objective of the proposed work is to allocate independent tasks to minimize the makespan and number of rejected tasks while achieving the deadline for acceptable VMs in the data center.

4 Proposed approach for task allocation

In this section, we propose a fault tolerant task allocation algorithm, i.e., FTTA for independent tasks with deadline in a heterogeneous cloud environment. The proposed approach is shown in Algorithm 1. The main purpose of our algorithm is to allocate tasks on available virtual machines based on minimum execution time in cloud computing. Literature study shows that the conventional task allocation algorithms encounter certain limitations, including issues with resource utilization, high number of task rejections, extended makespan, and reduced efficiency. Additionally, many of these algorithms rely on factors like minimum execution time or arrival time to determine task priority, which leads to a higher chance of missing deadlines. One such algorithm, i.e., RADL [16], presents a task allocation algorithm that focuses on load-balancing resource utilization but does not consider task prioritization and fault tolerance, resulting in an increased number of rejected tasks. Our work extends the idea of RADL for the allocation of tasks. It prioritizes the tasks based on deadline and average execution time, which minimizes the number of rejected tasks and improves efficiency. The algorithm also provides fault tolerance using preemptive migration by reallocating tasks from available to non-available virtual machines and also uses preemptive migration to minimize the number of rejected tasks. The initial step of the algorithm involves sorting tasks in the ready list based on computed priority. The algorithm allocates maximum tasks to the available or non-available virtual machines (VMs) at each step. The proposed approach for task allocation of deadline constrained tasks is a three-phase algorithm. The first phase is the task prioritizing phase, which computes the priority of tasks. After priority computation, tasks are arranged in the ready list in a non-decreasing order of priority value. In the second phase, the algorithm finds the available and non available virtual machines for each task with minimum execution time as mentioned in Algorithm 2. FTTA also tries to adjust the incoming task in the buffer time of the last task of a particular virtual machine. In the last phase, after finding a suitable virtual machine with minimum execution time, the algorithm assigns a task to that virtual machine. The algorithm tries to find an efficient solution for solving the deadline-constrained task allocation problem, minimizing the number of rejected tasks. Overall, the algorithm attempts to minimize rejected tasks, makespan and enhances efficiency. An overview of the proposed algorithm is given using the flowchart shown in Fig. 3. FTTA has the following characteristics:

- In instances where a task remains unallocated to any available virtual machine during the current step of the algorithm, it tries to migrate the task from a non-available VM to make it available using preemptive migration, which minimizes the makespan and number of rejected tasks.

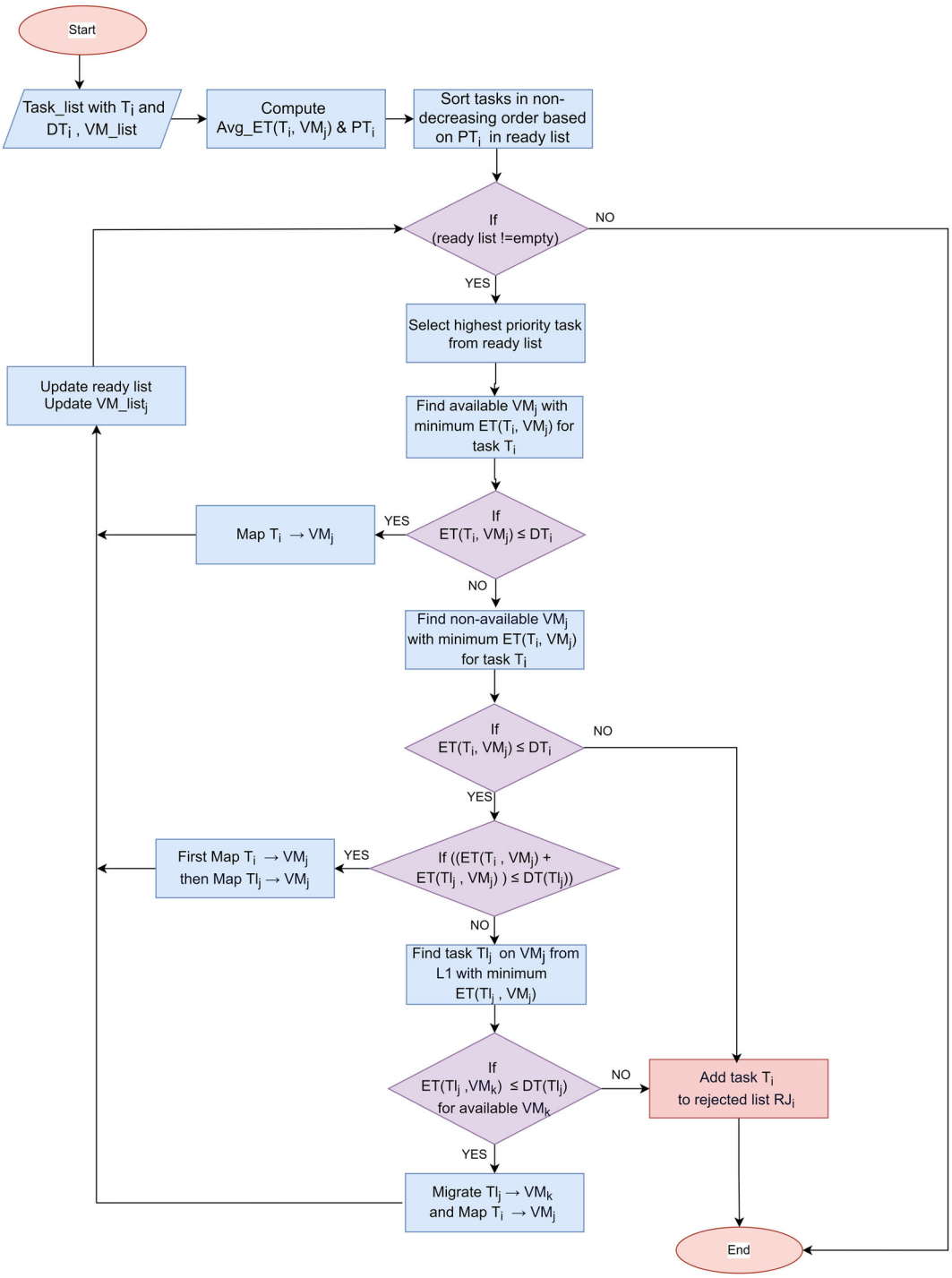

Fig. 3 Flowchart of the proposed task allocation algorithm

- The algorithm endeavors to optimize task allocation by accommodating tasks within the buffer time of virtual machines, thereby minimizing task rejection.
- It tries to allocate maximum tasks to the available as well as non-available VMs. The algorithm makes non available VM available by adjusting the allocated task of non available VM in order to reduce task failure.FTTA Algorithm

The following sections present key concepts essential for understanding our proposed algorithm. The first subsection illustrates task prioritization, while the second and the third subsections discuss the virtual machine selection and task allocation process in FTTA. We also provide the

Table 4 Comparision of rank computation of tasks

Tasks	VM_1	VM_2	VM_3	$(RT_i)_{FTTA}$	$(RT_i)_{prev}$
T_0	2	4	3	1	1
T_1	4	3	5	2	2
T_2	12	7	8	7	8
T_3	10	9	11	8	10
T_4	3	2	4	4	3
T_5	7	8	9	9	9
T_6	2	7	3	5	5
T_7	4	3	5	3	4
T_8	8	12	7	6	7
T_9	7	15	8	10	11
T_{10}	5	4	6	11	6

complexity analysis of our proposed algorithm in the following section. To further illustrate the functionality of FTTA, we demonstrate the working of FTTA with an illustrative example in the next section.

4.1 Task prioritizing phase

In this phase, we have computed the priority of a given task. For priority calculation, first $\overline{ET_i}$ for each task is computed based on 2. In this phase, priorities of all tasks are computed based on the difference between predefined deadlines and the average execution time of tasks computed on virtual machines, which is defined using Eq. 3.

$$PT_i = DT_i - \overline{ET_i}; 1 \leq i < n \tag{3}$$

Priority computation is done to ensure critical tasks i.e., tasks with shorter deadline are performed first so that fewer tasks get rejected.

Based on computed priority, each task is assigned a rank. In the FTTA (Fault-Tolerant Task Allocation) algorithm, the rank of each task is determined in non-decreasing order of priority.

Table 4 shows the rank calculated for FTTA where $(RT_i)_{FTTA}$ denotes the rank calculated using FTTA and $(RT_i)_{prev}$ represents the rank of tasks computed in previous work [27]. The priority in the previous work is computed based on deadline. The priority of each task is arranged in a non-decreasing order to determine the rannk i.e., $(RT_i)_{prev}$, and $(RT_i)_{FTTA}$.

Based on two different priority computation strategies, ranks $(RT_i)_{prev}$ and $(RT_i)_{FTTA}$ are computed. There is a slight difference in the ordering of tasks in the ready list. In table 4, T_7 is selected before T_4 as it has lower $(RT_i)_{FTTA}$ value whereas with $(RT_i)_{prev}$, T_4 is selected first. Considering the computation cost matrix, sorting tasks based on $(RT_i)_{prev}$ increases the task rejection ratio as compared to FTTA priority strategy $(RT_i)_{FTTA}$.Our algorithm prioritizes tasks by evaluating the minimum difference between the task deadline and execution time. Tasks with smaller differences are accorded higher priority, as they are closer to their respective deadlines and are at a greater risk of missing them. Prioritizing tasks based on $(RT_i)_{FTTA}$ ensures efficient allocation, enhancing overall system performance and guaranteeing improved deadline adherence by considering the time required for task completion, particularly in scenarios involving multiple tasks with diverse deadlines and execution times.

Algorithm 1 FTTA Algorithm

Input: Task_list with tasks T_i and deadline DT_i, $1 \leq i < n$, VM_list
Output: Map T_i to VM_j

for all Task T_i in Task_list **do**
  Compute average execution time of task T_i using equation 2
  Compute priority of each tasks using equation 3.
end for
Sort tasks in non-decreasing order according to their priority in ready list
while ready list is not empty **do**
  Select task T_i with highest priority from ready list.
  Find virtual machine VM_j from available virtual machines whose execution time is minimum for task T_i.
  if ET(T_i , VM_j) $\leq DT_i$ **then**
    Assign T_i to virtual machine VM_j.
    Update ready list
    Update VM_list_j
  else
    call Algorithm 2
  end if
end while

4.2 Virtual machine selection phase

The second phase is virtual machine selection, primarily focusing on allocating an available virtual machine for each deadline constrained task from the ready list. The algorithm follows three basic steps to select the virtual machine from the ready list. When the task arrives, the algorithm first checks the availability of virtual machines. Among available virtual machines, the algorithm selects the virtual machine with minimum execution time to meet the task deadline constraint. Selecting the virtual machine with the minimum execution time by considering the deadline constraints of the tasks, the algorithm ensures that the tasks are completed within their specified deadlines with minimal delay.

During the task allocation to VMs, specific tasks fail to meet the deadline criteria. i.e., the execution time of such tasks exceeds the deadline value; therefore, these tasks cannot adhere to their deadlines when allocated to the available virtual machines. In that case, the algorithm makes a sorted list of non-available (L1) and available (L2) virtual machines. Now, in the non-available list of virtual machines, the algorithm tries to adjust the task within the deadline of the last task in that virtual machine. A virtual machine is selected when the tasks satisfy the deadline constraint on that virtual machine, as it will also check for the buffer time of the last task allocated on that virtual machine. If the execution time of current task and the last task on that virtual machine lies within the deadline value, the algorithm selects that virtual machine for allocation.

If the task is not mapped to a non-available virtual machine, i.e., no virtual machine is selected from L1, the algorithm looks for the virtual machines in L2. The algorithm tries to migrate the last lask on a non-available virtual machine to an available virtual machine with minimum execution time. Then, the non-available virtual machine is selected for the tasks. To migrate the last task, it first checks the deadline constraints of respective tasks on the available virtual machine. With the migration of tasks from non-available VM to available VM, the non-available VM is available now. If the requested task satisfies the deadline constrained on a non-available virtual machine, that virtual machine is selected for the current task, and last lask is migrated using preemptive migration on the available VM. Migration of tasks is done when the virtual machine fails to satisfy the deadline constraint on available VMs. Migrating tasks between virtual machines ensures efficient task allocation to suitable virtual machines, which can help meet deadline constraints and mitigate task failures.

4.3 Task allocation phase

After priority computation, the algorithm sorts all the tasks in a non-decreasing order of task priority value in the ready list. Tasks in the ready list delineate the order in which tasks should be scheduled for allocation in the available virtual machines. Two tasks with the same priority are arranged chronologically in the ready list. This phase allocates the tasks from the ready list to the selected available virtual machines. Each task is allocated to execute on the virtual machine, ensuring its earliest completion while meeting the deadline constraints. After mapping the tasks to a selected virtual machine, the algorithm updates the ready list and task list of virtual machines.

Algorithm 2 Task allocation on non-available vitual machine VM_j

Input: Task_list with tasks T_i and deadline DT_i, $1 \leq i < n$, VM_list
Output: Map T_i to VM_j , rejected list RJ_i

1: Create a sorted list L1 of non-available VMs in non-decreasing order of ET(T_i , VM_j) , j∈ non-available VMs wrt T_i.
2: Similarly, create a sorted list L2 of available VMs in non-decreasing order of ET(T_i , VM_j) , j∈ available VMs w.r.t T_i.
3: Set flag = False
4: **for all** VM_j in L1 **do**
5: **if** ET(T_i , VM_j) ≤ DT_i and ET(T_i , VM_j) + ET(Tl_j, VM_j) ≤ $DT(Tl_j)$ **then**
6: Assign T_i to virtual machine VM_j before Tl_j
7: Update ready list and VM_list_j
8: flag = True
9: break
10: **end if**
11: **end for**
12: **if** flag == False **then**
13: **for all** non-available virtual machines VM_j in L1 **do**
14: **for all** available virtual machines VM_k in L2 **do**
15: **if** ET(T_i , VM_j) ≤ DT_i and ET(Tl_j, VM_k) ≤ $DT(Tl_j)$ **then**
16: Migrate Tl_j on virtual machine VM_k
17: Assign T_i to virtual machine VM_j
18: Update ready list , VM_list_j and VM_list_k
19: break
20: **else**
21: Add T_i to rejected list RJ_i
22: **end if**
23: **end for**
24: **end for**
25: **end if**

4.4 Complexity analysis of algorithm

This section outlines the complexity analysis of our proposed algorithm. To evaluate the complexity, we considered n number of tasks in the task list and m number of virtual machines in the cloud datacentre. It is assumed that the number of tasks is much greater than the number of virtual machines, i.e., $n \gg m$. Algorithm 1 presents the best case possibility of mapping tasks to a virtual machine. To perform the mapping, the algorithm receives a set of tasks from the task list with their deadlines and execution times and a list of virtual machines as input. The for loop (lines 1-4, Algo 1) repeats n several times and, computes the average execution time on each VM and decides the priority of each task defined by Eq. 3. Both parameters have a time complexity of $O(n)$. After computation of priority PT_i, tasks are sorted in the ready list in non-decreasing order (line 5). The time complexity of sorting the tasks takes $O(n \log n)$. It iteratively schedules the task selected by a task priority method (Line 6-7) using a while loop. The algorithm finds the available virtual machine with minimum execution time for each task in the ready list (line 8). After finding an available virtual machine for tasks, its execution time ET(T_i, VM_j) is compared with deadline DT_i (line 9). For a set of n tasks, finding minimum execution time on m virtual machines takes O(m) time and checking for deadline constraints and swapping of tasks takes O(1) time complexity each. If such a virtual machine exists, then the task T_i is allocated on VM_j (line 10) and the ready list and VMs list are updated (lines 11-12). Therefore, the time complexity from steps 6-15 is $O(n \cdot m)$. Hence, Algorithm 1 exhibits overall time complexity $O(n \log n + n * m)$. When no available virtual machine satisfies the deadline constraint, Algorithm 1 calls Algorithm 2 (line 17).

When the task deadline DT_i is less than the execution time of the task on virtual machine ET(T_i, VM_j) i.e., the task is unable to find available VM, Algorithm 1 invokes Algorithm 2. Algorithm 2 first creates two sorted lists of non-available VMs and available VMs (lines 1-2). Finding a sorted list of m virtual machines in Step 1-2 takes $O(m \log m)$ time complexity each. Before checking the condition, the algorithm sets the flag value as False (line 3). The algorithm iterates for each non-available virtual machine from the sorted list using a for loop (line 4). Line (5-10) shows the condition that first checks the deadline constraint of tasks on non-available virtual machines, i.e., [ET(T_i, VM_j) ≤ DT_i] (line 5). If the task deadline is met on the virtual machine VM_j, the algorithm checks for the

available buffer time [$DT_{l,j}$ - ET(Tl_j, VM_j)] of the last task Tl_j on that virtual machine, so that the requested task T_i execute within that buffer time and satisfies the deadline constraint $DT(Tl_j)$ of last task Tl_j on that virtual machine VM_j (line 5) which takes time complexity $O(m)$. If the condition is satisfied, the algorithm maps the requested task T_i on the virtual machine VM_j along with the last task of that virtual machine (line 6). After mapping tasks, the ready and virtual machine lists are updated and the flag values are set to true (line 7-9). If the task T_i is not mapped to non-available virtual machine VM_j and the flag value is still false, then the algorithm iterates for each available virtual machine VM_k (line 12-14). It checks the deadline constraint of task T_i on VM_j and also the deadline of the last task of VM_j on VM_k (line 15). In case the condition is true, the algorithm tries to migrate the last task Tl_j of non-available virtual machine VM_j to available virtual machine VM_k with minimum execution time (line 16). After migrating, task T_i is mapped to VM_j (line 17). Step 12-17 in Algorithm 2 are the most time consuming (referring to the 2 for loops in step 13-14), they need $O(m^2)$ time for each task in ready list. The ready list, available and non-available virtual machine lists are updated (line 18). Hence, Algorithm 2 exhibits $[O(m \log m) + O(m) + O(m^2)]$ i.e., $O(m^2)$ time. These steps are repeated for each task in the ready list and if any task does not satisfy the deadline condition on the virtual machine, they are added to the rejected list RJ_i (line 21). Therefore, FTTA has overall $O(n \log n + nm^2)$ time complexity.

4.5 An illustrative example

Consider a cost computation matrix consisting of 11 tasks with their respective deadlines and three virtual machines as shown in Table 5. The proposed algorithm i.e., FTTA, first computes the average execution time of tasks using Eq. 2 and then determines the priority of tasks using Eq. 3. Tasks are allocated to the virtual machine in the order they are arranged in a ready list. Table 6 illustrates an example demonstrating the step-by-step allocation of tasks in FTTA using the cost computation matrix.

Table 5 Average execution time of tasks on 3 virtual machines

Tasks	VM_1	VM_2	VM_3	Avg_ET	DT
T_0	2	4	3	3	3
T_1	4	3	5	4	5
T_2	12	7	8	9	12
T_3	10	9	11	10	13
T_4	3	2	4	3	5
T_5	7	8	9	8	12
T_6	2	7	3	4	6
T_7	4	3	5	4	5
T_8	8	12	7	9	11
T_9	7	15	8	10	14
T_{10}	5	4	6	5	9

The illustrative example uses 11 tasks T_0, T_2.....,T_{10} having different execution time on *3* heterogeneous virtual machines VM_1, VM_2 and VM_3. The execution time of each task for corresponding virtual machine is given in the cost computation matrix as shown in table 3. The execution time of each task on virtual machines is calculated using Eq. 1. Using Eq. 3, the priority of the task is calculated. Each tasks are assigned a rank based on priority as shown in Table 4. Based on the rank, tasks are added to the ready list in non-decreasing order. When the tasks have the same priority, they are sorted based on their arrival.

- Initially, all virtual machines are idle, as shown in Fig. 4a. At first, task T_0 has the topmost priority in the ready list, which is allocated first on available VMs i.e., VM_1,VM_2 and VM_3. FTTA finds the available virtual machine with lowest execution time (ET) for task T_0, i.e., minimum value among ET(0,1), ET(0,2), ET(0,3). Virtual machine VM_1 has the minimum execution value of 2. Then the minimum ET value is compared with the task T_0 deadline, i.e., ($2 \leq 3$) where ET value is 2 and deadline of task T_0 is 3. The ET value lies within the deadline of task T_0. Therefore, T_0 is assigned to VM_1 as shown in Fig. 4b
- Subsequently, the algorithm follows the same step for the next priority task, T_1. The ET of task T_1 on available VMs i.e., VM_2, VM_3 are 3 and 5 respectively. The algorithm selects VM_2 with minimum ET for task T_1, i.e., 3. As tasks satisfies the deadline constraint, T_1 is mapped to VM_2 as shown in Fig. 4c
- Similarly, the next priority task in the ready list is T_7. Virtual machine VM_2 have the minimum ET value, but the algorithm selects the available virtual machine with earliest ready time which also satisfy deadline constraint of task T_7. Therefore, T_7 is mapped to VM_3 which has earliest ready time as shown in Fig. 4d.
- Next priority task is T_4. The algorithm follows the same step shown in Fig. 4a. The ET value for T_4 on available VMs i.e., VM_1 is 3 and the respective deadline constraint is satisfied. Therefore, task T_4 is mapped to VM_1 as illustrated in Fig. 4e
- FTTA maps task T_6 to VM_1. The algorithm first finds the ET value of T_6 on available VMs i.e., VM_2. It compares the deadline of task T_6 with its ET value on VM_2, i.e., ($7 \leq 6$), which is not satisfying. Therefore, the algorithm looks for the minimum ET value of T_6 on non-available VMs (VM_1,VM_3). Virtual machine VM_1 has the minimum ET value 2, which is compared with

Table 6 Step by step mapping of tasks using FTTA

Step	Ready list	Task selected	Execution time			Deadline	VM selected
			VM_1	VM_2	VM_3		
1	T_0, T_1,T_7,T_4,T_6,T_8, T_2,T_3,T_5,T_9,T_{10}	T_0	**2**	4	3	3	VM_1
2	T_1,T_7,T_4,T_6,T_8, T_2,T_3,T_5,T_9,T_{10}	T_1	4	**3**	5	5	VM_2
3	T_7,T_4,T_6,T_8, T_2,T_3,T_5,T_9,T_{10}	T_7	4	**3**	5	5	VM_2
4	T_4,T_6,T_8, T_2,T_3,T_5,T_9,T_{10}	T_4	**3**	2	4	5	VM_1
5	T_6,T_8, T_2,T_3,T_5,T_9,T_{10}	T_6	**2**	7	3	6	VM_1
6	T_8, T_2,T_3,T_5,T_9,T_{10}	T_8	8	12	**7**	11	VM_3
7	T_2,T_3,T_5,T_9,T_{10}	T_2	12	**7**	8	12	VM_2
8	T_3,T_5,T_9,T_{10}	T_3	**10**	9	11	13	VM_1
9	T_5,T_9,T_{10}	T_5	7	8	**9**	12	VM_3
10	T_9,T_{10}	T_9	7	15	**8**	14	VM_1
11	T_{10}	T_{10}	5	**4**	6	9	VM_1

the deadline constraint of task T_6 first. When the deadline is satisfied, the algorithm finds the last task mapped on VM_1. The last task in VM_1 list is T_4 with ET value and deadline as 3, 5 respectively. Afterwards, the algorithm checks whether the requested task T_6 lies within the deadline of T_4. As the condition is satisfied, T_6 is mapped to VM_1 first then task T_4 is mapped as shown in Fig. 4f.

- The next task in the ready list is T_8 with ET value 8, 12 and 7 on VM_1, VM_2, VM_3 respectively. The ET value on available VM, i.e., VM_2, is 12, which is compared with the deadline of task T_8 i.e., ($12 \leq 11$) where 11 is the deadline of T_8. Following the same steps in step 4 (f), the algorithm finds the ET of task T_8 on non-available VMs (VM_1, VM_3). T_8 have minimum ET on virtual machine VM_3 i.e., 7. After comparing the ET value with the deadline, the algorithm finds the last task (T_7) in the VM_3 list. Task T_8 does not satisfy the deadline constraint of task T_7 such that T_8 does not lie within the deadline of the last task of VM_3. Therefore, it checks for the next non-available VM, i.e., VM_1 with ET value and deadline 8 and 5 respectively. After checking the deadline constraint, the requested task T_8 does not lie within the last task of VM_1 list. Now, the algorithm tries to migrate the last task of the non-available VM, i.e., VM_3, to the available VM (VM_2). To migrate the last task T_7 to VM_2, it checks the task deadline and ET value in the respective VM. After checking, the algorithm checks the requested task T_8 deadline constraint on VM_3, which is satisfied. Therefore, task T_7 is migrated to VM_2 and task T_8 is mapped to VM_3 as illustrated in Fig. 4g.
- Next priority task is T_2 with ET and deadline value 7 and 12 respectively on available virtual machine VM_2. The algorithm follows the same step discussed in Fig. 4a. After comparing the deadline of T_2 with ET value such that ($7 \leq 12$), task T_2 is mapped to VM_2 as shown in Fig. 4h.
- Similarly, task T_3 is mapped to available VM i.e., VM_1 with minimum ET value 10 and deadline 13 i.e., ($10 \leq 13$) as shown in Fig. 4i. Also, task T_5 with ET value and deadline 9,12 respectively on available VM i.e., VM_3 is mapped to VM_3 as shown in Fig. 4j.
- Now considering the next priority task T_9 in the ready list with ET value and deadline as 15 and 14 respectively on available VM (VM_2). Comparing deadline constraint ($15 \leq 14$), task T_9 does not satisfy the condition as shown in step 4(f). Following the same step in Fig. 4f, the algorithm finds the ET of task T_9 on non-available VMs (VM_1, VM_3). T_9 have minimum ET i.e., 7 on virtual machine VM_1. After comparing ET value with the deadline, the algorithm finds the last task (T_3) in the VM_1 list. Task T_9 does not satisfy the deadline constraint of task T_3 such that T_9 does not lie within the deadline of the last task of VM_1. Therefore, it checks for the next non-available VM i.e., VM_3, with ET value and deadline 8 and 13, respectively. After checking the deadline constraints, the requested task T_9 does not lie within the last task of VM_3 list. Now, the algorithm tries to migrate the last task of the non-available VM i.e., VM_1 to available VM (VM_2). To migrate the last task T_3 to VM_2, it checks the ET and deadline of task in the respective VM. After checking, the algorithm checks the requested task T_9 deadline constraint on VM_3, which is satisfied. Therefore, task T_3 is migrated from VM_1 to VM_2 and task T_9 is mapped to VM_1 as illustrated in Fig. 4k. Similarly, next priority task T_{10} is mapped to VM_1 as shown in Fig. 4l.
- Again, the algorithm checks the ready list. The algorithm stops when the ready list is empty.

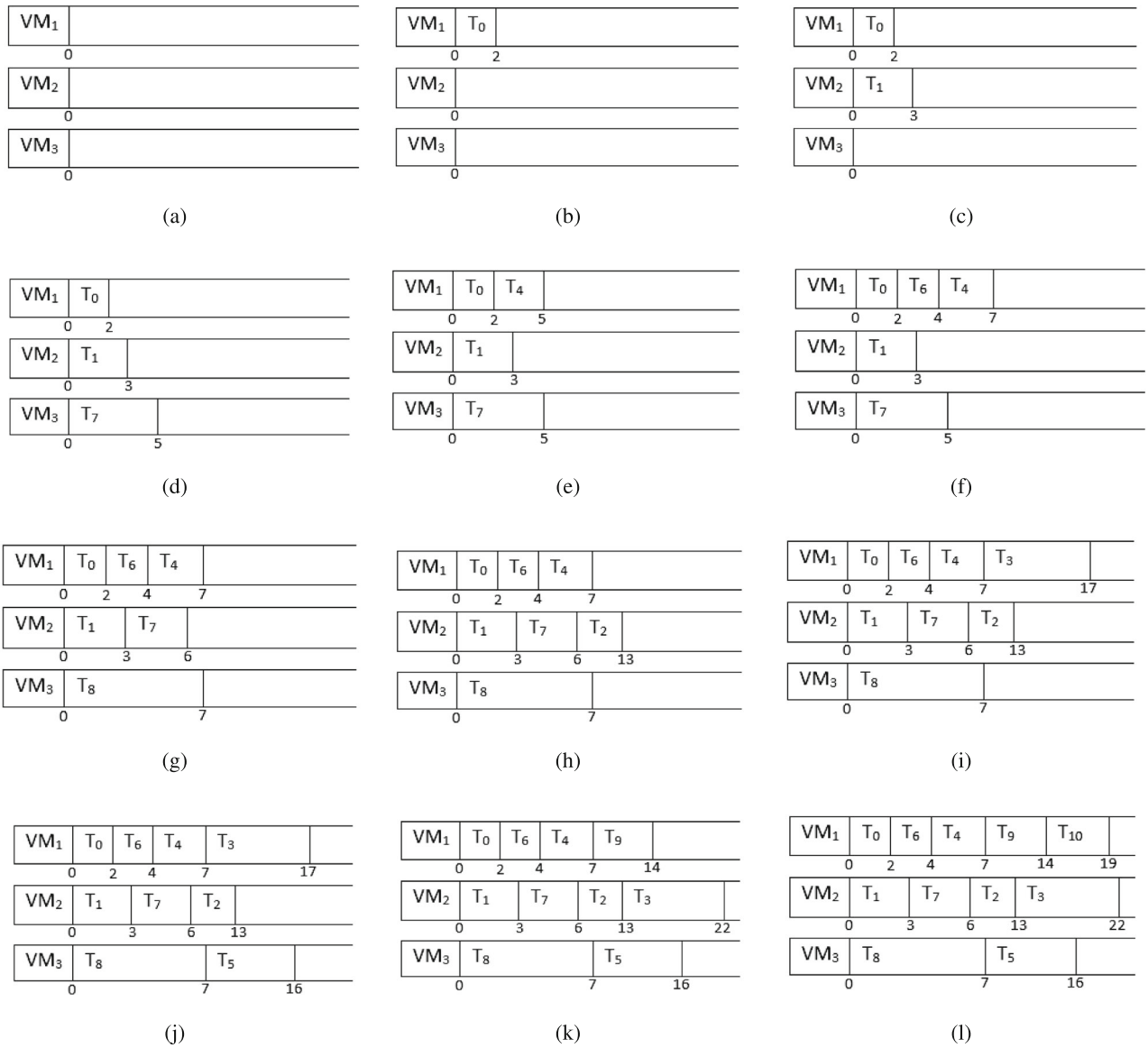

Fig. 4 **a** All virtual machines are idle **b** Highest priority task T_0 from the ready list is allocated on available virtual machines VM_1 **c** Task T_1 allocated to VM_2 **d** Task T_7 allocated to VM_3 **e** Task T_4 allocated to the next available virtual machine VM_1 **f** Task T_6 adjusted in the buffer time of Task T_4 on VM_1 **g** Task T_7 migrated to VM_2 and task T_8 allocated to VM_3 (h) Task T_2 allocated to next available virtual machine VM_2 **i** Task T_3 allocated to VM_1 **j** Task T_5 allocated to VM_3 **k** Task T_3 migrated to VM_2 and task T_9 allocated to VM_1 **l** Task T_{10} allocated to the next available virtual machine VM_1

5 Simulation results and analysis

This section outlines the experimental setup, workload details, performance analysis, discussions of the results of the proposed task allocation algorithm, and comparison of its performance with the existing allocation techniques. We analyze the simulation outcomes with the five well-known existing algorithms in literature: First Come First Served (FCFS) [11], Priority based algorithm [12], Shortest Job First (SJF) [13], Dynamic Maximum Minimum (Dy max min) algorithm [14] and RADL [16]. Comparison parameters such as number of rejected tasks, makespan, speedup and efficiency are analyzed for performance evaluation. Based on these factors, we have evaluated and compared the existing algorithms with our proposed algorithm FTTA. Here, the obtained results of the algorithms are analyzed based on average analysis, where results are computed based on the input parameters. We performed simulations on various sets of tasks, varying the number of virtual machines.

5.1 Experimental setup

To measure the performance of existing allocation algorithms, simulation is conducted using Python programming language. The implementation of the cloud environment, VMs, and data center (DC) was achieved using the SimPy library in Python. In our study, we utilized SimPy to create and simulate the cloud environment, including the instantiation of virtual machines, task allocation, and resource management within the data center. Specifically, we designed classes and functions within the SimPy framework to represent the components of the cloud environment, such as VMs, tasks, and the data center infrastructure. The primary goal of our experiments is to assess the effectiveness of these algorithms in managing deadline constrained task execution on a set of virtual machines (VMs). The allocation algorithm is experimented with using Spyder IDE version 3.9. The simulation was conducted on a Dell Inspiron 15 machine with Windows 10 Home. The machine is equipped with 11th Gen Intel(R) Core(TM) i5-1135G7 processor operating at a base frequency of 2.40 GHz frequency with a memory of 8 GB.

In the experiment, three sets of independent tasks are generated to find the performance results of FTTA on different virtual machines, as shown in Table 7.

5.2 Performance metrics

The performance of algorithms is evaluated based on four metrics as number of rejected tasks, makespan, speedup and efficiency.

1. Number of rejected tasks:
   A rejected task is defined as any task that could not be scheduled within their specified deadlines. Minimizing the count of rejected tasks helps maximize virtual machine utilization and enhance overall efficiency.
2. Makespan:
   Makespan represents the time at which the last task from the ready list completes its execution on a particular virtual machine. The goal is to optimize the time it takes to finish all tasks while meeting deadline constraints.

$$makespan = max(ACT_i) \quad \forall i \in \{1, 2, \ldots, n\} \tag{4}$$

3. Speedup:
   Speedup is expressed as the ratio between the time required for sequential execution and that for parallel execution. Sequential execution involves executing tasks sequentially on a single processor. Parallel execution, on the other hand, corresponds to the makespan, denoting the time taken for task completion on multiple processors.

$$Speedup = \frac{Sequentialexecutiononthefastestprocessor}{Makespan} \tag{5}$$

4. Efficiency:
   Efficiency refers to how effectively an allocation algorithm utilizes available virtual machines to schedule tasks. It is computed as the ratio of speedup and number of virtual machines.

$$Efficiency = \frac{Speedup}{Numberofvirtualmachines} \tag{6}$$

Table 7 Experimental environment

Workload	GoCJ dataset
Number of VMs	3, 5, 7
Small Set of Tasks	10, 20, 30, 40, 50
Medium Set of Tasks	100, 200, 300, 400, 500
Large Set of Tasks	1000, 2000, 3000, 4000, 5000

5.3 Workload generation

The benchmark dataset, i.e., GoCJ dataset [47], is used for simulation. This benchmark dataset does not include any information about the deadline. The deadline is included according to the method introduced in [27] for priority task assignment. The GoCJ dataset is taken from real-world traces collected from Google Cluster traces [48] and logs generated by MapReduce of M45 supercomputing cluster [49]. The generated workload comprised of different files. Each file consists of a length of tasks measured in Million Instructions (MI). The GoCJ dataset consists of tasks with different lengths from 15000 MIs to 900000 MIs and is formed using a well-known Monte Carlo simulation method. The number of tasks within each file is determined by the filename itself; for example, "GoCJ_Dataset_1000" contains the length of 1000 tasks. For simulation, these tasks are executed on virtual machines with different computation capacity in MIPS. We have fixed the computation capacity of the virtual machine based on [16]. The computation power of VMs are taken from 100MIPS to 1500 MIPS as shown in Fig. 5.

This workload is categorized in the ranges from 10 tasks to 5000 tasks. The length of a small set of tasks ranges from 1 to 50 tasks, a medium set from 100 to 500 tasks, and a large set from 1000 to 5000 tasks.

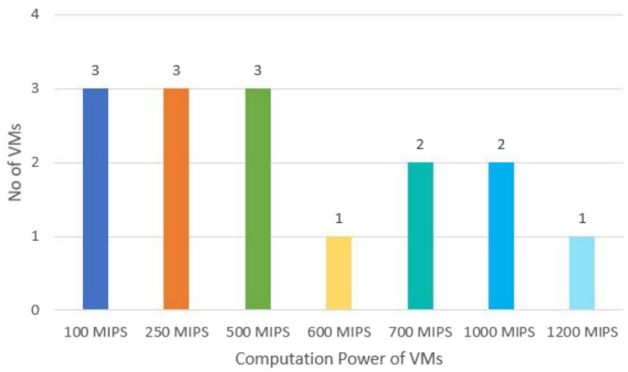

Fig. 5 Computation power of VMs

5.4 Experiment for a small set of tasks

In this section, we outline a series of simulation results performed for a small set of tasks to evaluate the performance of the algorithms. We conducted a comparative analysis between FTTA and the existing algorithms, i.e., FCFS, SJF, Priority, Dy max min and RADL, across varying numbers of VMs. The number of VMs is adjusted to 3, 5, and 7 to analyze the results obtained.

5.4.1 When virtual machine count is 3

The performance for a small set of tasks when the number of virtual machines is three, shown in Fig. 6. The results are evaluated based on four metrics such as number of rejected tasks, makespan, speedup and efficiency.

Figure 6a shows the comparison result of FTTA with the FCFS, SJF, Priority, Dy max min and RADL, respectively for rejected tasks count while varying the tasks. FTTA prioritizes tasks with closer deadlines and minimum execution time. Hence, it outperforms the compared algorithms for each task count. The computation of task priorities in the FTTA is based on a concept that aims to minimize task rejection. Figure 6b shows the evaluation result of the FTTA with the existing algorithms in terms of makespan while changing the number of tasks. The result shows that the makespan of FTTA is better than existing algorithms. Overall, FTTA produces an average improvement of 45%, 49%, 44%, 50%, and 37% over FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms respectively for makespan.

Figure 6c illustrates the speedup results obtained for task allocation algorithms with varying numbers of tasks. For a small set of tasks, the FTTA algorithm produces better speedup than other algorithms. Overall, the figure shows that FTTA outperforms the existing algorithms by 36%, 46%, 39%, 47% and 33%, respectively, on average in terms of speedup. Figure 6d shows the performance of FTTA for efficiency. Efficiency evaluates how effectively virtual machines are used in a parallel system. The result obtained shows that FTTA performs more efficiently when compared to other existing task allocation algorithms.

5.4.2 When virtual machine count is 5

The performance results for a small set of task allocations when the number of VMs increases to five are illustrated in Fig. 7. The result shows that with the increase in virtual machines, algorithms give better results. FTTA performs better than the existing algorithms in terms of rejected tasks, makespan, speedup and efficiency.

Figure 7a shows that the FTTA algorithm has less number of rejected tasks while varying the tasks count. This is because, with the increase in the number of virtual machines, tasks are more likely to find the suitable VM for allocation within the deadline, which inturn, reduces rejected tasks. The analysis of makespan results obtained for a small set of tasks when a number of virtual machines is illustrated in Fig. 7b. FTTA gives better results than compared allocation algorithms with an increase in virtual machines. Overall, FTTA improves on an average of 31%, 43%, 30%, 45%, and 25% over FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms respectively for makespan.

Similarly, FTTA outperforms in terms of speedup and efficiency when compared to existing algorithms. Figure 7c and d show the evaluation results of speedup and efficiency, respectively for a small set of tasks. It shows that FTTA improves the speedup by 29%, 46%, 30%, 45%, and 25%, respectively on average, compared to existing algorithms.

5.4.3 When virtual machine count is 7

The performance results for a small set of task allocations when VM is increased to seven are shown in Fig. 8. As the number of virtual machines increases, the FTTA algorithm demonstrates enhanced results compared to existing algorithms in terms of reduced rejected task count, decreased makespan, enhanced speedup, and improved efficiency.

It is observed from Fig. 8a that the number of rejected tasks decreases while allocating tasks to virtual machines in FTTA when compared to existing algorithms. With a further increase in the number of VMs, FTTA efficiently allocates maximum tasks to VMs, reducing the number of rejected tasks. The makespan analysis for a small set of tasks when the number of virtual machines increased to 7 is illustrated in Fig. 8b. With the increase in virtual machines, FTTA gives better results than allocation algorithms. Overall, FTTA improves by 34%, 50%, 38%, 51%, and 13% respectively on an average over FCFS, SJF, Priority,

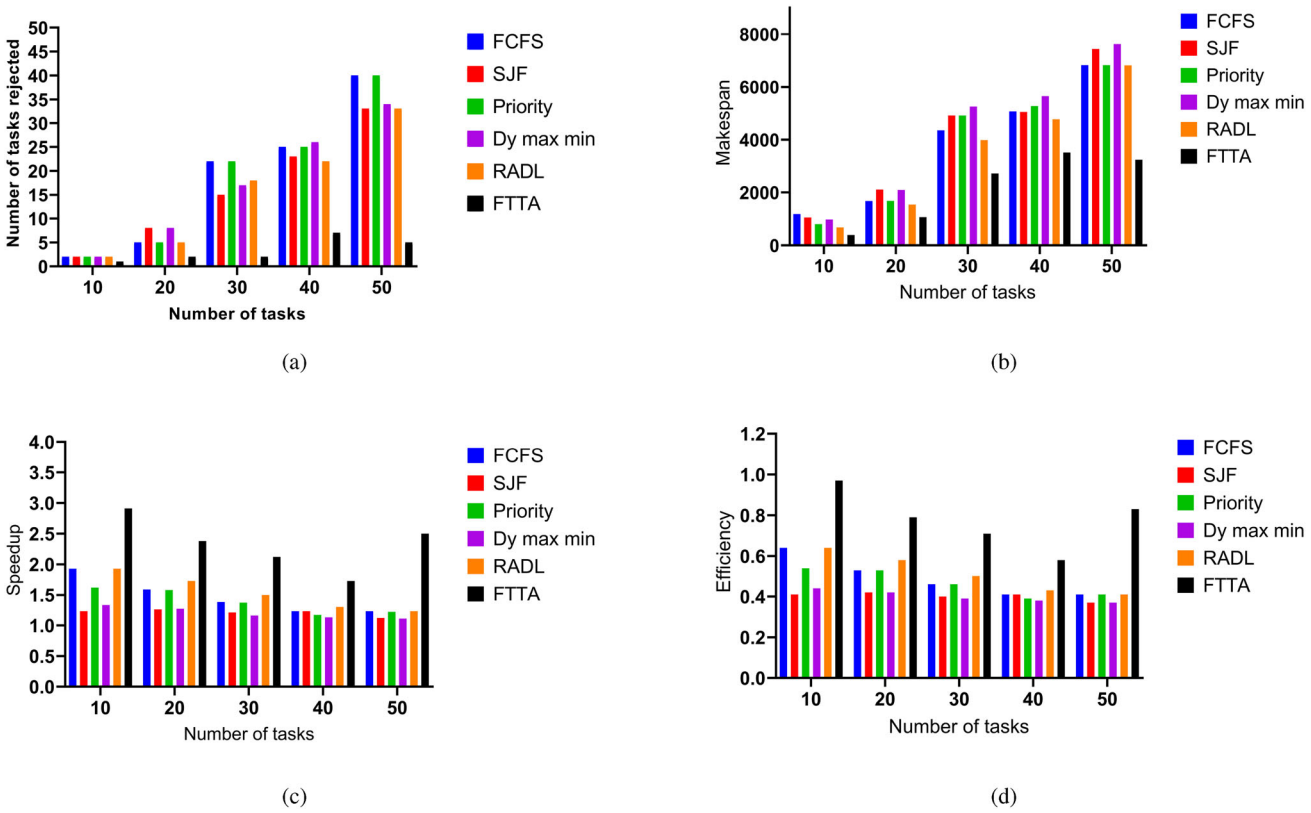

Fig. 6 **a** Number of rejected tasks for small set of tasks when VM = 3 **b** Makespan for small set of tasks when VM = 3 **c** Speedup for small set of tasks when VM = 3 **d** Efficiency for small set of tasks when VM = 3

Dy max min and RADL task allocation algorithms respectively for makespan. Figure 8c shows the analysis results of FTTA when compared with existing task allocation algorithms for speedup. The result shows that FTTA has improved results than all existing task allocation algorithms in terms of speedup. Overall, FTTA gives an average improvement of 38%, 55%, 39%, 56%, and 23% respectively, compared to FCFS, SJF, Priority, Dy max min, and RADL task allocation algorithms.

Similarly, FTTA also outperforms existing algorithms with respect to efficiency illustrated in Fig. 8d.

5.5 Experiment for medium set of tasks

In this section, we present the results obtained from experiments conducted with a medium set of tasks to evaluate all the task allocation algorithms. We conducted a comparative analysis between FTTA and the existing algorithms, considering different numbers of VMs. The number of virtual machines was adjusted, specifically set to 3, 5, and 7, to analyze the results obtained for a medium set of tasks.

5.5.1 When virtual machine count is 3

The result analysis for a medium set of tasks when VM is set to three is illustrated in Fig. 9. The result of FTTA is compared with existing algorithms for four metrics, i.e., rejected tasks, makespan, speedup and efficiency.

Fig. 9a illustrates the evaluation result of FTTA for the number of rejected tasks with a medium set of tasks. The result demonstrates that the FTTA algorithm allocates a larger number of tasks to the suitable VM when compared with existing algorithms. Figure 9b illustrates the result analysis of all algorithms for makespan. It is observed that FTTA has 44%, 49%, 46%, 50% and 45% improvement on an average against FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms, respectively in terms of makespan. The overall average improvement of the FTTA algorithm in terms of speedup is 40%, 43%, 41%, 44% and 38% over existing task allocation algorithms as shown in Fig. 9c. Figure 9d presents the efficiency value for FTTA along with state-of-the-art algorithms. It is concluded that FTTA performs better than FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms in terms of efficiency.

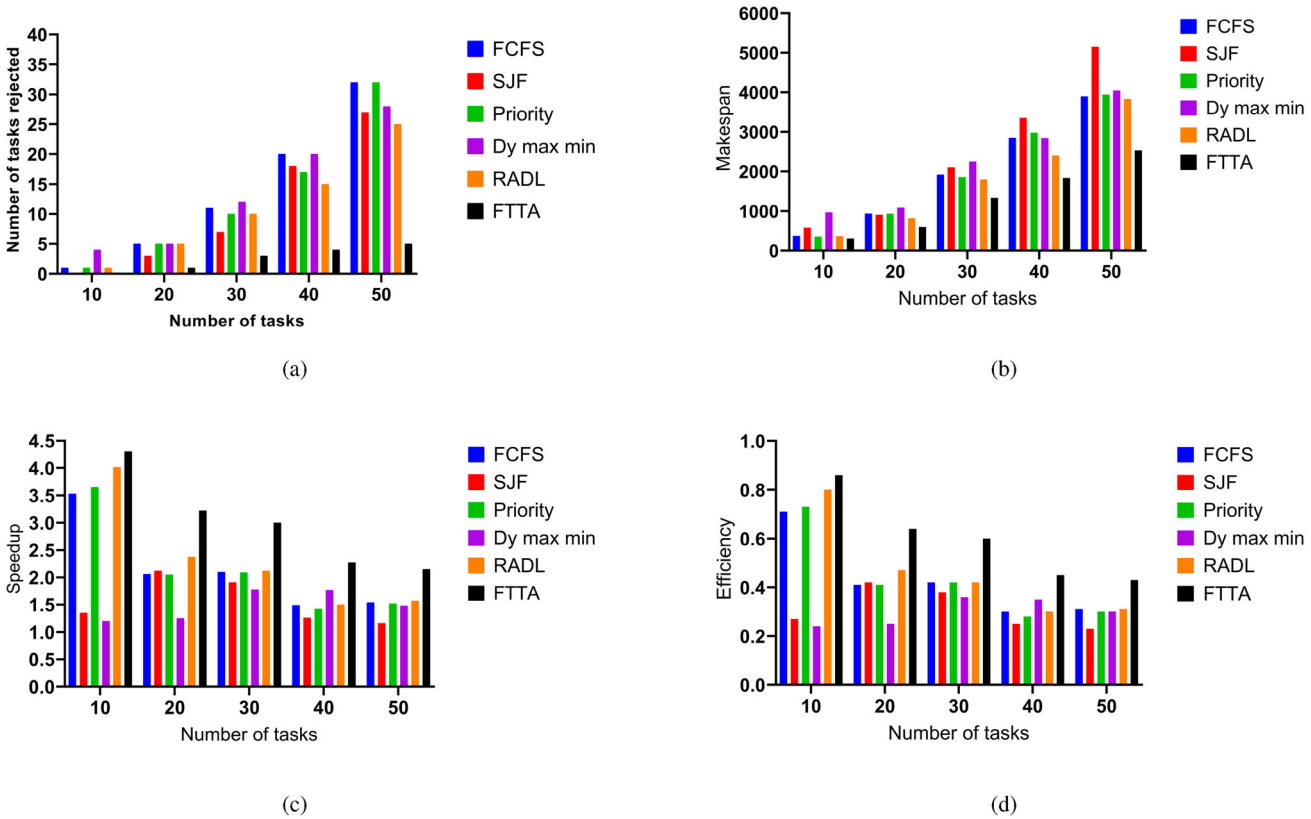

Fig. 7 **a** Number of rejected tasks for small set of tasks when VM = 5 **b** Makespan for small set of tasks when VM = 5 **c** Speedup for small set of tasks when VM = 5 **d** Efficiency for small set of tasks when VM = 5

5.5.2 When virtual machine count is 5

The performance analysis for a medium set of tasks is illustrated in Fig. 10 when the count of VM increased to 5. The results show that the FTTA performs more efficiently in terms of the number of rejected tasks, makespan, speedup and efficiency.

Fig. 10a shows that FTTA has less number of rejected tasks than existing algorithms. Figure 10b illustrates FTTA performs an average improvement of 53%, 58%, 54%, 49% and 53%, respectively against FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms respectively in terms of makespan. Similarly, FTTA performs better than the existing algorithms in terms of speedup and efficiency as shown in Fig. 10c and d respectively. Overall, FTTA gives an average improvement of 45%, 53%, 47%, 41% and 45% respectively as compared to FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms in terms of speedup.

5.5.3 When virtual machine count is 7

The results for a medium set of tasks when the VM count is increased to seven are shown in Fig. 11.

Fig. 11a illustrates that the performance of FTTA is improved than existing algorithms in terms of rejected tasks. Overall, FTTA has an average improvement of 36%, 39%, 26%, 36%, and 24%, respectively, when compared with FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms respectively in terms of makespan as shown in Fig. 11b. Figure 11c shows the speedup evaluation for all task allocation algorithms. Overall, FTTA performs better by 42%, 45%, 36%, 45%, 30%, respectively on average against existing algorithms. Similarly, our proposed algorithm performs more efficiently than other algorithms, as shown in Fig. 11d.

5.6 Experiment for a large set of tasks

In this section, we provide a series of simulation results conducted on a large set of tasks to analyze the performance of all the algorithms. We performed a comparative analysis between FTTA and existing ones while varying VMs. The virtual machines are systematically configured to 3, 5, and 7 to evaluate the outcomes comprehensively.

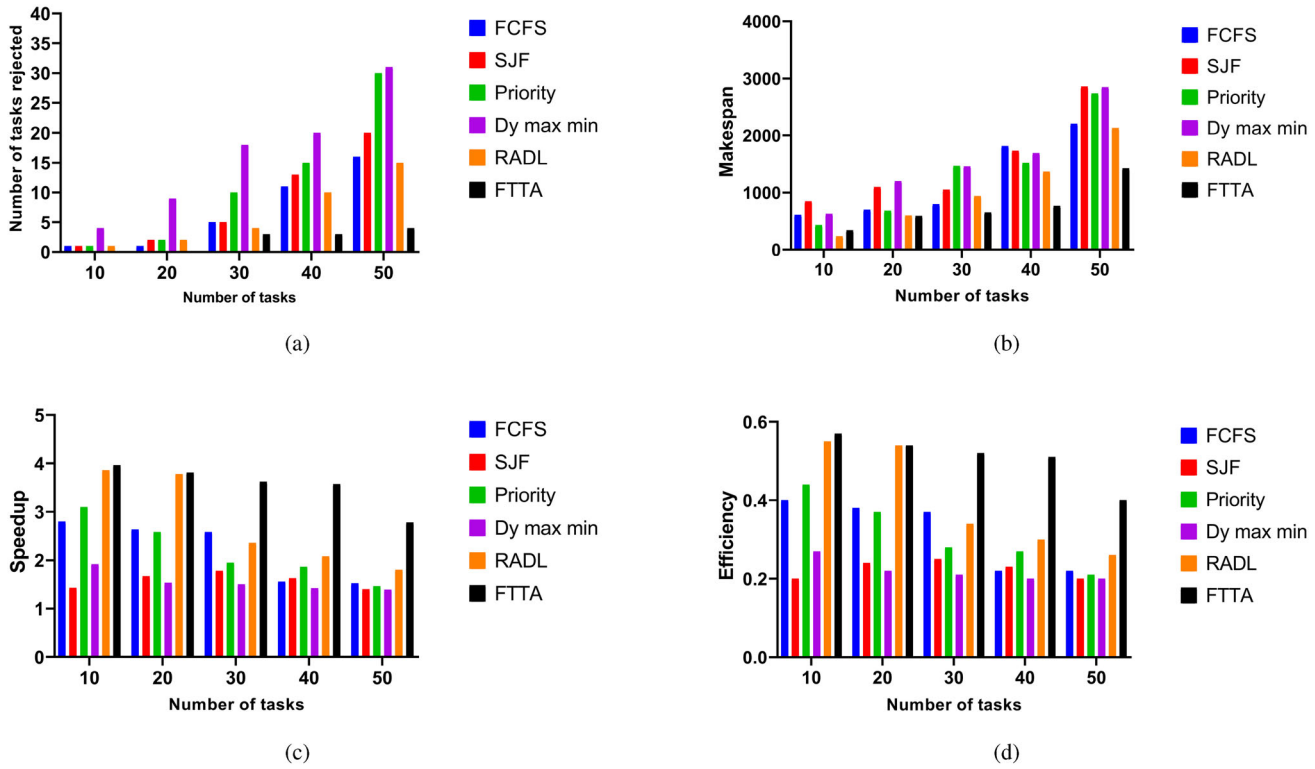

Fig. 8 **a** Number of rejected tasks for small set of tasks when VM = 7 **b** Makespan for small set of tasks when VM = 7 **c** Speedup for small set of tasks when VM = 7 **d** Efficiency for small set of tasks when VM = 7

5.6.1 When virtual machine count is 3

The comparison results of the allocation algorithms for performance metrics such as rejected tasks, makespan, speedup and efficiency while varying the number of tasks are shown in Fig. 12. The FTTA algorithm performs better than other allocation algorithms when the number of VMs is 3 while varying the number of tasks.

Figure 12a demonstrates that the FTTA algorithm performs better than FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms regarding rejected tasks. Figure 12b compares the makespan value of all the algorithms. Overall, FTTA has an average improvement of 16%, 15%, 15%, 17% and 12%, respectively when compared with existing task allocation algorithms. Figure 12c illustrates that FTTA performs better for speedup value as compared to existing algorithms. Overall, FTTA has an average improvement of 12%, 12%, 12%, 13% and 11% respectively, when compared with FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms in terms of speedup. FTTA also performs better in terms of efficiency as compared to existing algorithms, as shown in Fig. 12d.

5.6.2 When virtual machine count is 5

The results for large set of tasks varying from 1000 to 5000 are shown in Fig. 13, when total VM count is five.

Fig. 13a illustrates that FTTA algorithm shows improved performance than existing allocation algorithms for rejected tasks. Overall FTTA has an average improvement of 62%, 62%, 61%, 62%, and 62% respectively when compared with FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms respectively in terms of makespan as shown in Fig. 13b. Figure 13c illustrates the speedup evaluation for all task allocation algorithms. Overall, FTTA performs better by 62%, 62%, 62%, 62%, and 57% respectively on average against existing algorithms in terms of speedup. Similarly, our proposed algorithm performs more efficiently than other algorithms as shown in Fig. 13d.

5.6.3 When virtual machine count is 7

The performance evaluation for a large set of task allocation when VM count is increased to seven are shown in Fig. 14.It is observed from the Fig. 14a that number of rejected tasks decreases while allocating tasks to virtual machines in FTTA when compared to existing algorithms.

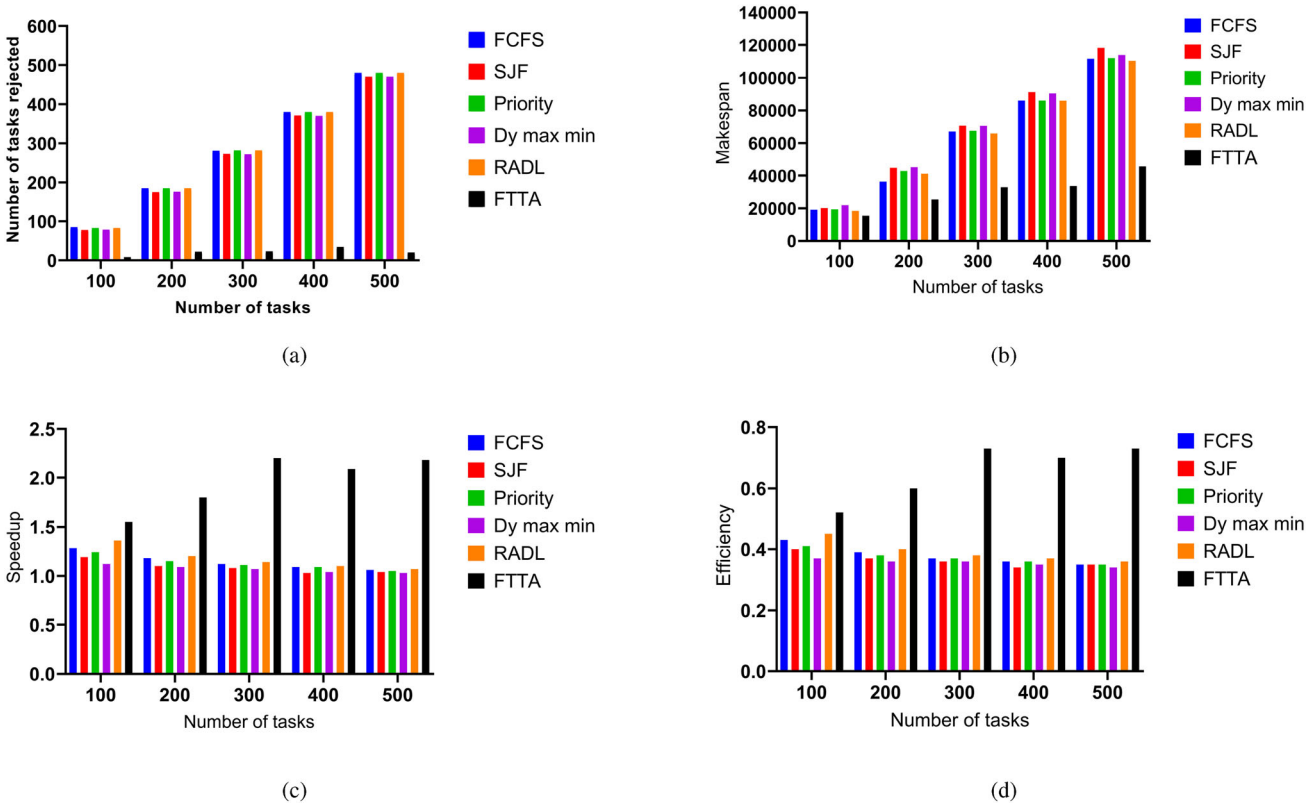

Fig. 9 **a** Number of rejected tasks for medium set of tasks when VM = 3 **b** Makespan for medium set of tasks when VM = 3 **c** Speedup for medium set of tasks when VM = 3 **d** Efficiency for medium set of tasks when VM = 3

The makespan analysis for a set of tasks varying from 1000 to 5000 is illustrated in Fig. 14b. FTTA gives better result than compared allocation algorithms. Overall, FTTA produces an average improvement of 50%, 50%, 50%, 51%, and 48% respectively over FCFS, SJF, Priority, Dy max min and RADL task allocation algorithms respectively for makespan with increase in VM. Figure 14c shows the speedup value analysis of FTTA when compared with existing task allocation algorithms. The result shows that overall, FTTA gives an average improvement of 51%, 51%, 50%, 51%, and 49% respectively, compared to FCFS, SJF, Priority, Dy max min, and RADL task allocation algorithms. Similarly, FTTA also outperforms existing algorithms with respect to efficiency as shown in Fig. 14d.

5.7 Result analysis

We have performed simulation for small set of tasks, medium set of tasks and large set of tasks while varying number of virtual machines to 3, 5 and 7. The performance of FTTA is analyzed and compared with FCFS, SJF, Priority, Dy max min, and RADL algorithms based on number of rejected tasks, makespan, speedup and efficiency.

The results shown in Figs. 6a, 7a, and 8a for small set of tasks, Figs. 9a, 10a, and 11a for medium set of tasks, and Figs. 12a, 13a, and 14a for large set of tasks concludes that FTTA algorithm produces fewer tasks that are rejected after task allocation with virtual machine 3, 5 and 7 respectively. It happens because the FTTA algorithm assigns tasks to VMs in a way that optimally utilizes maximum tasks. It allows tasks prioritization based on the remaining time available for the completion of a task in order to meet the deadline of maximum tasks. FTTA also attempts to allocate tasks within the buffer time of the last task on a non-available virtual machine. Additionally, the algorithm attempts to minimize task rejections by migrating tasks between available and non-available virtual machines. This proactive tasks allocation in suitable VMs significantly reduces the likelihood of missed deadlines. We have compared FTTA based on average of small, medium and large set of tasks while varying the virtual machines. Overall, FTTA shows enhancement in average improvement of rejected tasks with respect to FCFS, SJF, Priority, Dy max min, and RADL algorithms while varying tasks and virtual machines.

The performance comparision of makespan value for FTTA with existing algorithms are illustrated in Figs. 6b, 7b, and 8b for small set of tasks, Figs. 9b, 10b, and 11b for medium set of tasks, and Figs. 12b, 13b, and 14b for large set of tasks. We have evaluated FTTA based on average of

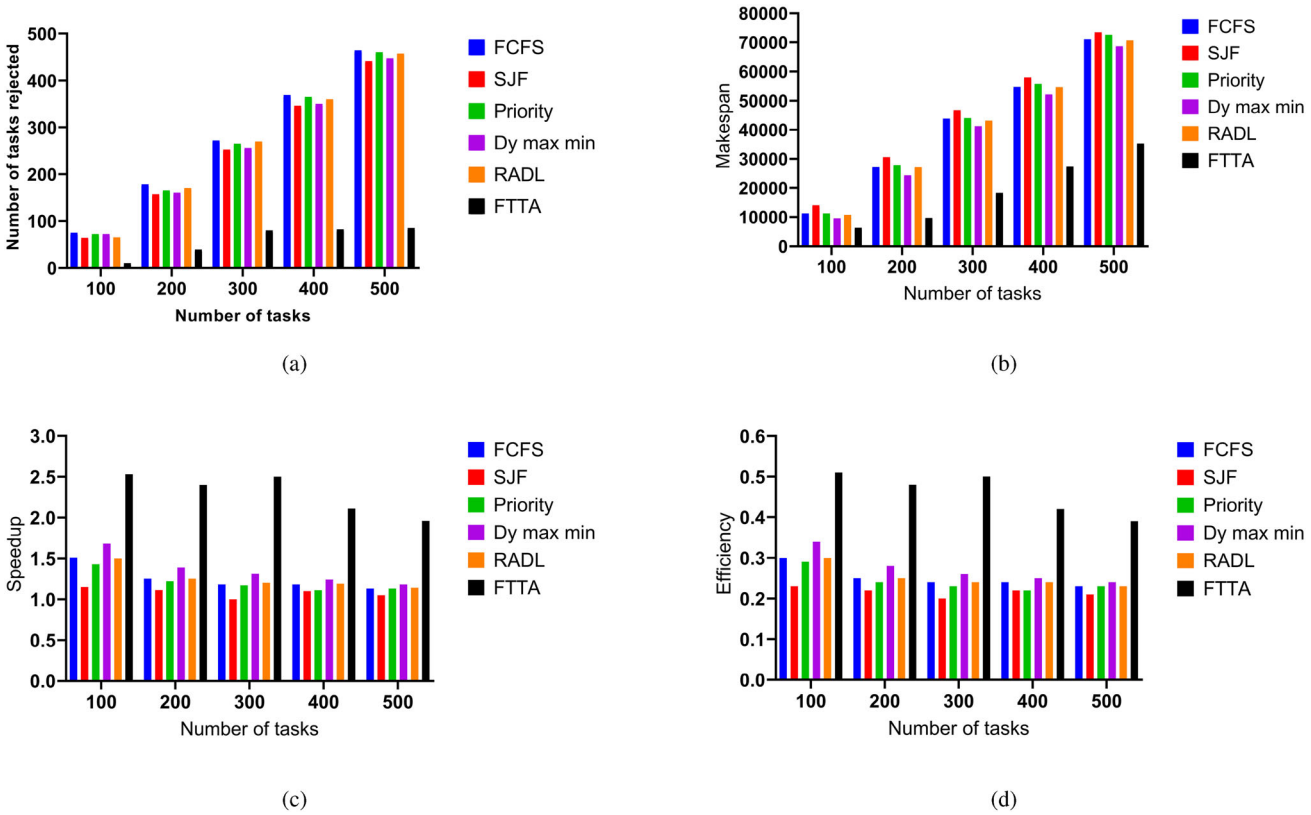

Fig. 10 **a** Number of rejected tasks for medium set of tasks when VM = 5 **b** Makespan for medium set of tasks when VM = 5 **c** Speedup for medium set of tasks when VM = 5 **d** Efficiency for medium set of tasks when VM = 5

small, medium anf large set of tasks while varying the virtual machines. It is concluded from the comparision results that FTTA outperforms existing task allocation algorithms. FTTA algorithm ensures that the tasks are always assigned to available virtual machines. This approach of efficient virtual machine selection allows tasks to select virtual machine with shorter execution time that reduces the overall time required to finish all tasks. Furthermore, Overall, FTTA shows enhancement in average improvement of makespan with respect to FCFS, SJF, Priority, Dy max min, and RADL algorithms. over FCFS, SJF, Priority, Dy max min and RADL algorithm while varying tasks and virtual machines.

We also evaluated the results of proposed algorithm over existing algorithms for speedup as shown in Figs. 6c, 7c, and 8c for small set of tasks, Figs. 9c, 10c, and 11c for medium set of tasks, and Figs. 12c, 13c, and 14c for large set of tasks. It is concluded from the results that FTTA performs better. FTTA allows task allocation to available virtual machines which reduces the amount of idle time of VMs. These VMs process new tasks immediately on the available VM instead of waiting. When idle VMs are effectively utilized, tasks complete faster, minimizing the overall time of the workload leading to improved speedup. We have evaluated FTTA based on average of small, medium anf large set of tasks while varying the virtual machines. Overall, FTTA shows an average improvement Overall, FTTA shows enhancement in average improvement of speedup with respect to FCFS, SJF, Priority, Dy max min, and RADL algorithms while varying tasks and virtual machines.

It is concluded from the results that FTTA performs more efficiently compared to FCFS, SJF, Priority, Dy max min and RADL respectively as shown in Figs. 6d, 7d, and 8d for small set of tasks, Figs. 9d, 10d, and 11d for medium set of tasks, and Figs. 12d, 13d, and 14d for large set of tasks. We have evaluated FTTA based on average of small, medium anf large set of tasks while varying the virtual machines. Overall, When tasks are distributed among 3 virtual machines, FTTA shows an average efficiency improvement when compared to FCFS, SJF, Priority, Dy max min, and RADL algorithms while varying tasks and virtual machines.

The results show that the proposed task allocation algorithm has a much shorter makespan as compared to existing algorithms for all sets of tasks while varying number of virtual machines. The overall performance increases, when the number of virtual machines increase from 3 to 7, FTTA outperforms as compared to existing algorithms.

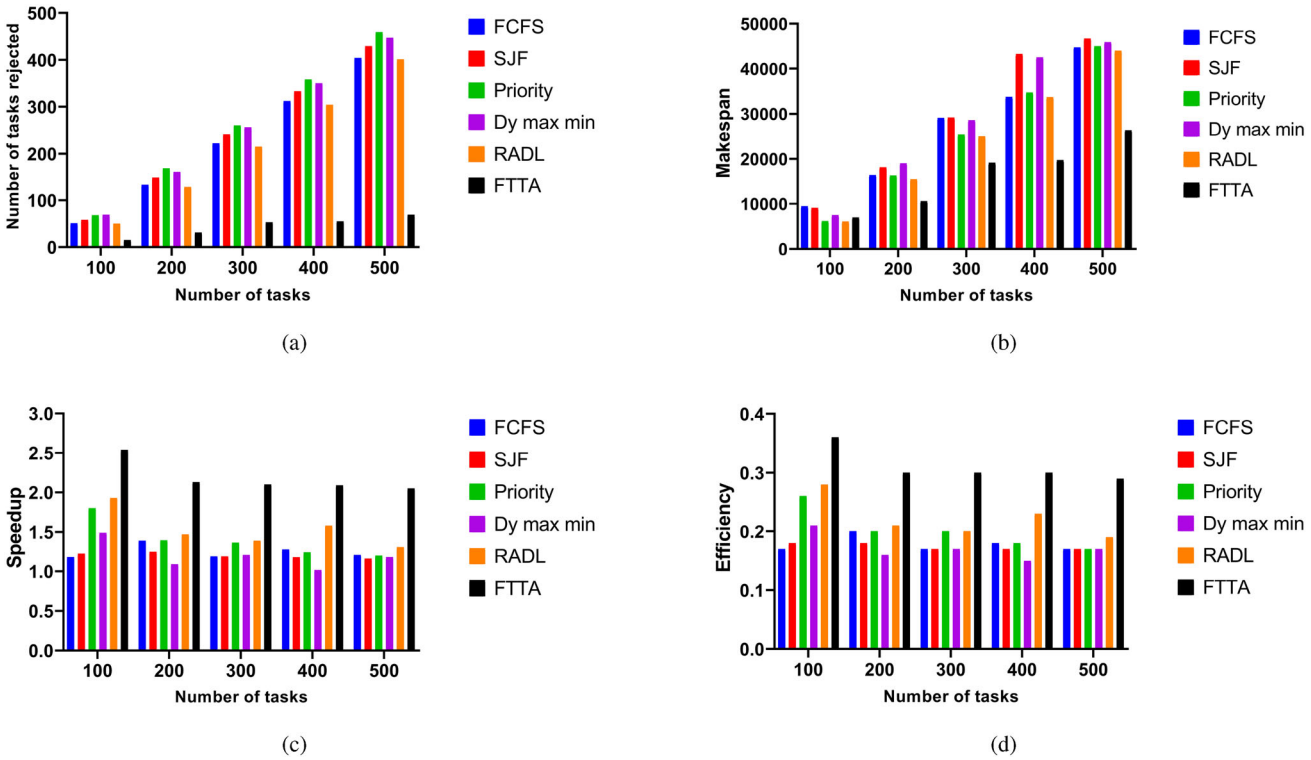

Fig. 11 **a** Number of rejected tasks for medium set of tasks when VM = 7 **b** Makespan for medium set of tasks when VM = 7 **c** Speedup for medium set of tasks when VM = 7 **d** Efficiency for medium set of tasks when VM = 7

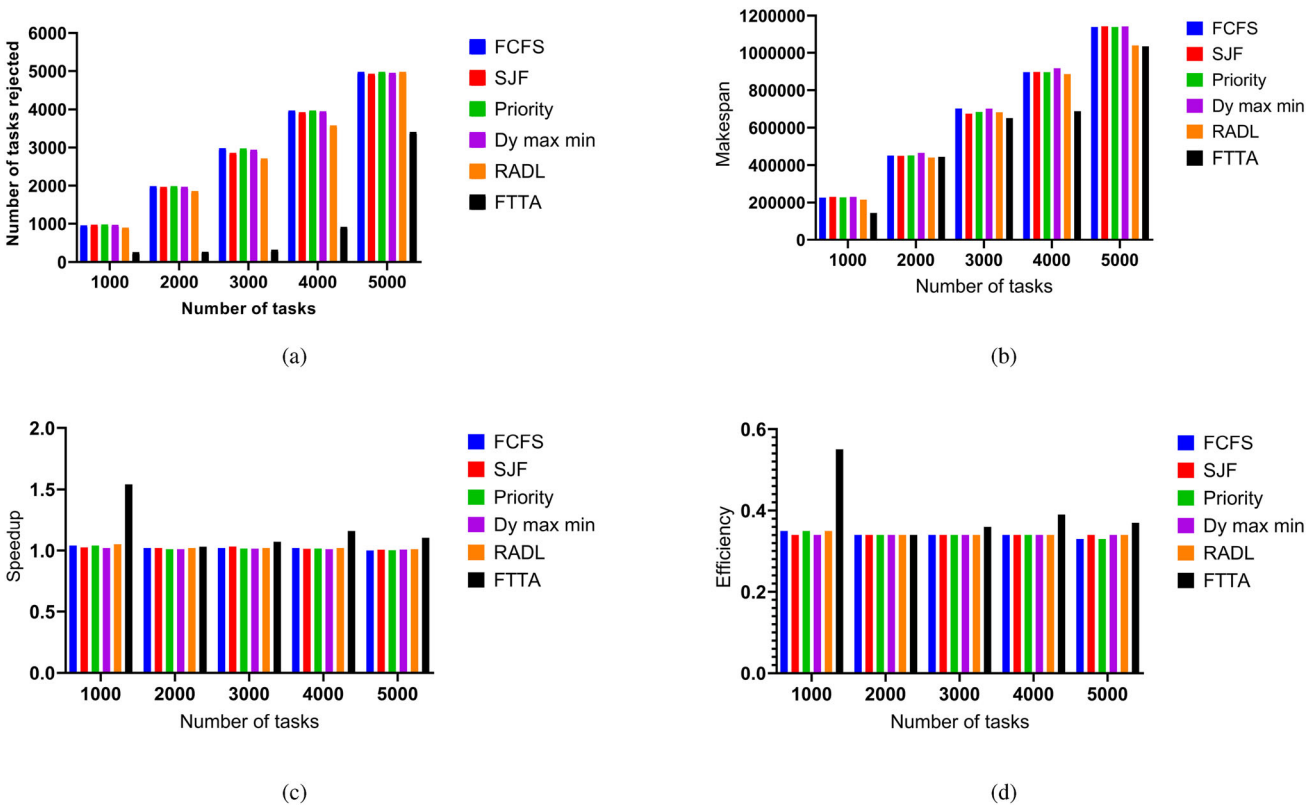

Fig. 12 **a** Number of rejected tasks for large set of tasks when VM = 3 **b** Makespan for large set of tasks when VM = 3 **c** Speedup for large set of tasks when VM = 3 **d** Efficiency for large set of tasks when VM = 3

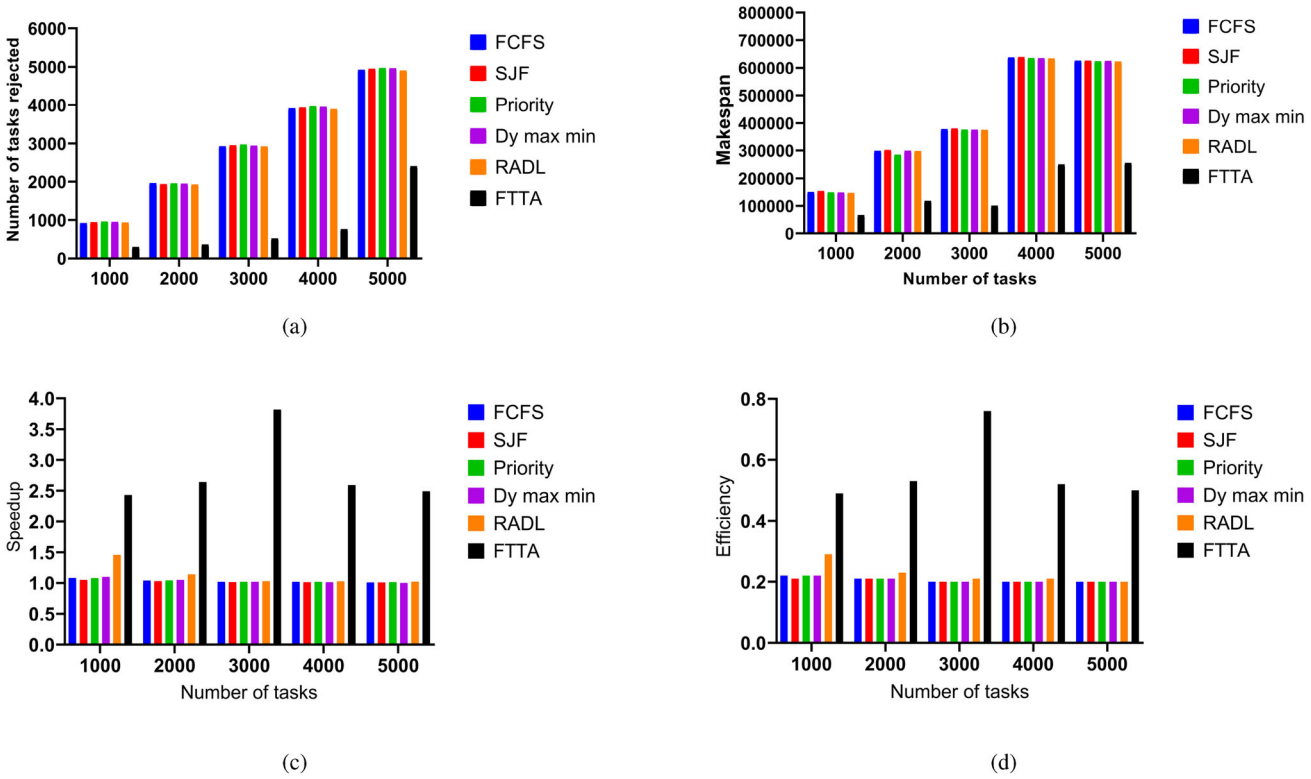

Fig. 13 **a** Number of rejected tasks for large set of tasks when VM = 5 **b** Makespan for large set of tasks when VM = 5 **c** Speedup for large set of tasks when VM = 5 **d** Efficiency for large set of tasks when VM = 5

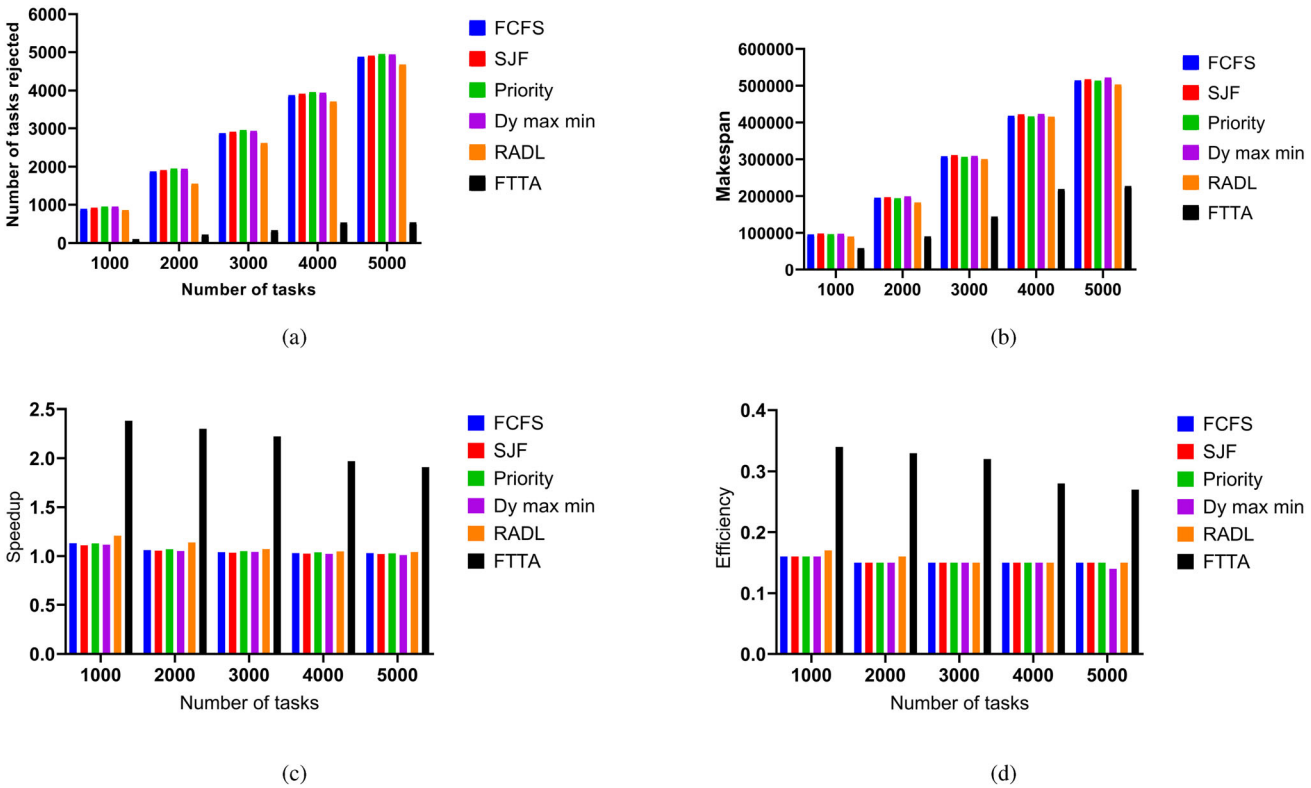

Fig. 14 **a** Number of rejected tasks for large set of tasks when VM = 7 **b** Makespan for large set of tasks when VM = 7 **c** Speedup for large set of tasks when VM = 7 **d** Efficiency for large set of tasks when VM = 7

6 Conclusion

In this article, we have introduced a Fault-Tolerant Task Allocation Approach (FTTA) for allocation of independent tasks in heterogeneous cloud environment. Since the state-of-the-art algorithms lacks fault tolerant task allocation approach, we propose a novel fault tolerant task allocation algorithm for deadline constrained tasks. The proposed algorithm consists of three phases including task prioritization, virtual machine selection and task allocation phase. The algorithm prioritizes incoming tasks based on deadline and shorter execution time to minimize the probability of task rejection as compared to priority decided based on deadline. FTTA tries to improve resource efficiency by utilizing the available buffer time of virtual machine efficiently. The algorithm uses preemptive migration for mitigating the task failure. The performance of FTTA is evaluated and compared with the existing task allocation approaches such as FCFS, SJF, Priority, Dy max min, and RADL algorithms. The algorithms are evaluated using benchmark dataset i.e., GoCJ dataset having three sets of tasks while varying number of virtual machine from 3, 5 and 7. The result analysis shows that FTTA performs better than FCFS, SJF, Priority, Dy max min, and RADL algorithms in terms of rejected tasks, makespan, speedup and efficiency. In the future, the performance of proposed algorithm can be improved by minimizing the number of migration between the virtual machines.

References

1. Hu, B., Yang, X., Zhao, M.: Energy-minimized scheduling of intermittent real-time tasks in a CPU-GPU cloud computing platform. In: IEEE Transactions on Parallel and Distributed Systems (2023)
2. Li, Z., Yu, H., Fan, G., Zhang, J.: Cost-efficient fault-tolerant workflow scheduling for deadline-constrained microservice-based applications in clouds. In: IEEE Transactions on Network and Service Management (2023)
3. Zhang, L., Bai, J., Xu, J.: Optimal allocation strategy of cloud resources with uncertain supply and demand for SAAS providers. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3300735
4. Singh, S., Chana, I., Buyya, R.: Star: SLA-aware autonomic management of cloud resources. IEEE Trans. Cloud Comput. **8**(4), 1040–1053 (2017)
5. Taheri, H., Abrishami, S., Naghibzadeh, M.: A cloud broker for executing deadline-constrained periodic scientific workflows. In: IEEE Transactions on Services Computing (2023)
6. Hai, T., Zhou, J., Jawawi, D., Wang, D., Oduah, U., Biamba, C., Jain, S.K.: Task scheduling in cloud environment: optimization, security prioritization and processor selection schemes. J. Cloud Comput. **12**(1), 15 (2023)
7. Maurya, A.K., Modi, K., Kumar, V., Naik, N.S., Tripathi, A.K.: Energy-aware scheduling using slack reclamation for cluster systems. Clust. Comput. **23**, 911–923 (2020)
8. Chen, X., Lu, C.-D., Pattabiraman, K.: Failure analysis of jobs in compute clouds: a google cluster case study. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering, pp. 167–177. IEEE (2014)
9. Liakath, J.A., Krishnadoss, P., Natesan, G.: Dccwoa: a multi-heuristic fault tolerant scheduling technique for cloud computing environment. In: Peer-to-Peer Networking and Applications, pp. 1–18 (2023)
10. Kirti, M., Maurya, A.K., Yadav, R.S.: Fault-tolerance approaches for distributed and cloud computing environments: a systematic review, taxonomy and future directions. In: Concurrency and Computation: Practice and Experience, p. e8081 (2024)
11. Hamid, L., Jadoon, A., Asghar, H.: Comparative analysis of task level heuristic scheduling algorithms in cloud computing. J. Supercomput. **78**(11), 12931–12949 (2022)
12. Kumar, A.M.S., Venkatesan, M.: Task scheduling in a cloud computing environment using HGPSO algorithm. Clust. Comput. **22**(Suppl 1), 2179–2185 (2019)
13. Kaur, R., Laxmi, V., Balkrishan: Performance evaluation of task scheduling algorithms in virtual cloud environment to minimize Makespan. In: International Journal of Information Technology, pp. 1–15 (2022)
14. Nabi, S., Ibrahim, M., Jimenez, J.M.: Dralba: dynamic and resource aware load balanced scheduling approach for cloud computing. IEEE Access **9**, 61283–61297 (2021)
15. Mishra, A., Narayan Sahoo, M., Satpathy, A.: H3csa: a Makespan aware task scheduling technique for cloud environments. Trans. Emerg. Telecommun. Technol. **32**(10), e4277 (2021)
16. Nabi, S., Aleem, M., Ahmed, M., Islam, M.A., Iqbal, M.A.: RADL: a resource and deadline-aware dynamic load-balancer for cloud tasks. J. Supercomput. **78**(12), 14231–14265 (2022)
17. Amini Motlagh, A., Movaghar, A., Rahmani, A.M.: Task scheduling mechanisms in cloud computing: a systematic review. Int. J. Commun. Syst. **33**(6), e4302 (2020)
18. Nayak, S.C., Parida, S., Tripathy, C., Pattnaik, P.K.: An enhanced deadline constraint based task scheduling mechanism for cloud environment. J. King Saud Univ. Comput. Inf. Sci. **34**(2), 282–294 (2022)
19. Dubey, K., Sharma, S.C.: A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing. Sustain. Comput. **32**, 100605 (2021)
20. Houssein, E.H., Gad, A.G., Wazery, Y.M., Suganthan, P.N.: Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. Swarm Evol. Comput. **62**, 100841 (2021)
21. Arunarani, A.R., Manjula, D., Sugumaran, V.: Task scheduling techniques in cloud computing: a literature survey. Future Gener. Comput. Syst. **91**, 407–415 (2019)
22. Zhang, P.Y., Zhou, M.C.: Dynamic cloud task scheduling based on a two-stage strategy. IEEE Trans. Autom. Sci. Eng. **15**(2), 772–783 (2017)
23. Maurya, A.K., Tripathi, A.K.: On benchmarking task scheduling algorithms for heterogeneous computing systems. J. Supercomput. **74**(7), 3039–3070 (2018)
24. He, X., Shen, J., Liu, F., Wang, B., Zhong, G., Jiang, J.: A two-stage scheduling method for deadline-constrained task in cloud computing. Clust. Comput. **25**(5), 3265–3281 (2022)
25. Nabi, S., Ahmed, M.: OG-RADL: overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks. J. Supercomput. **77**, 7476–7508 (2021)
26. Zhang, L., Zhou, L., Salah, A.: Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments. Inf. Sci. **531**, 31–46 (2020)
27. Kumar, M., Sharma, S.C.: Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment. Comput. Electr. Eng. **69**, 395–411 (2018)

28. Alworafi, M.A., Mallappa, S.: A collaboration of deadline and budget constraints for task scheduling in cloud computing. Clust. Comput. **23**(2), 1073–1083 (2020)
29. Nabi, S., Ahmed, M.: PSO-RDAL: particle swarm optimization-based resource-and deadline-aware dynamic load balancer for deadline constrained cloud tasks. J. Supercomput. (2022). https://doi.org/10.1007/s11227-021-04062-2
30. Maurya, A.K., Tripathi, A.K.: Deadline-constrained algorithms for scheduling of bag-of-tasks and workflows in cloud computing environments. In: Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications, pp. 6–10 (2018)
31. Sahoo, S., Sahoo, B., Turuk, A.K.: A learning automata-based scheduling for deadline sensitive task in the cloud. IEEE Trans. Serv. Comput. **14**(6), 1662–1674 (2019)
32. Tarafdar, A., Debnath, M., Khatua, S., Das, R.K.: Energy and Makespan aware scheduling of deadline sensitive tasks in the cloud environment. J. Grid Comput. **19**, 1–25 (2021)
33. Yan, H., Zhu, X., Chen, H., Guo, H., Zhou, W., Bao, W.: Deft: dynamic fault-tolerant elastic scheduling for tasks with uncertain runtime in cloud. Inf. Sci. **477**, 30–46 (2019)
34. Kanwal, S., Iqbal, Z., Al-Turjman, F., Irtaza, A., Khan, M.A.: Multiphase fault tolerance genetic algorithm for VM and task scheduling in datacenter. Inf. Process. Manag. **58**(5), 102676 (2021)
35. Malik, M.K., Singh, A., Swaroop, A.: A planned scheduling process of cloud computing by an effective job allocation and fault-tolerant mechanism. J. Ambient Intell. Hum. Comput. **13**, 1–19 (2022)
36. Heyang, X., Sen, X., Wei, W., Guo, N.: Fault tolerance and quality of service aware virtual machine scheduling algorithm in cloud data centers. J. Supercomput. **79**(3), 2603–2625 (2023)
37. Marahatta, A., Xin, Q., Chi, C., Zhang, F., Liu, Z.: PEFS: AI-driven prediction based energy-aware fault-tolerant scheduling scheme for cloud data center. IEEE Trans. Sustain. Comput. **6**(4), 655–666 (2020)
38. Chen, J., Han, P., Liu, Y., Xiaoyan, D.: Scheduling independent tasks in cloud environment based on modified differential evolution. Concurr. Comput. **35**(13), e6256 (2023)
39. Indhumathi, R., Amuthabala, K., Kiruthiga, G., Yuvaraj, N., Pandey, A.: Design of task scheduling and fault tolerance mechanism based on GWO algorithm for attaining better QoS in cloud system. Wirel. Personal Commun. **128**(4), 2811–2829 (2023)
40. Nanjappan, M., Natesan, G., Krishnadoss, P.: HFTO: hybrid firebug tunicate optimizer for fault tolerance and dynamic task scheduling in cloud computing. Wirel. Personal. Commun. **129**(1), 323–344 (2023)
41. Tamilvizhi, T., Parvathavarthini, B.: A novel method for adaptive fault tolerance during load balancing in cloud computing. Clust. Comput. **22**(Suppl 5), 10425–10438 (2019)
42. Sheikh, S., Nagaraju, A., Shahid, M.: A fault-tolerant hybrid resource allocation model for dynamic computational grid. J. Comput. Sci. **48**, 101268 (2021)
43. Chinnathambi, S., Santhanam, A., Rajarathinam, J., Senthilkumar, M.: Scheduling and checkpointing optimization algorithm for byzantine fault tolerance in cloud clusters. Clust. Comput. **22**, 14637–14650 (2019)
44. Saidi, K., Bardou, D.: Task scheduling and VM placement to resource allocation in cloud computing: challenges and opportunities. Clust. Comput. **26**(5), 3069–3087 (2023)
45. Haidri, R.A., Alam, M., Shahid, M., Prakash, S., Sajid, M.: A deadline aware load balancing strategy for cloud computing. Concurr. Comput. **34**(1), e6496 (2022)
46. Yao, G., Ren, Q., Li, X., Zhao, S., Ruiz, R.: A hybrid fault-tolerant scheduling for deadline-constrained tasks in cloud systems. IEEE Trans. Serv. Comput. **15**(3), 1371–1384 (2020)
47. Hussain, A., Aleem, M.: GOCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures. Data **3**(4), 38 (2018)
48. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format+ schema. Google Inc. White Pap. **1**, 1–14 (2011)
49. Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P.: An analysis of traces from a production Mapreduce cluster. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 94–103. IEEE (2010)

Medha Kirti is presently pursuing Ph.D. degree from Motilal Nehru National Institute of Technology Allahabad, Prayagraj, India. She has completed her B.Tech. degree in Computer Science and Engineering from Banasthali University, Rajasthan, India and M.Tech. degree in Computer Science and Engineering from National Institute of Technology Patna, India. Her research area is Distributed Computing and Cloud Computing. She is currently working in the area of Fault Tolerance in Distributed Computing.

Ashish Kumar Maurya has completed his M.Tech. degree in Computer Science & Engineering from Indian Institute of Technology Roorkee, India, and Ph.D. from Department of Computer Science & Engineering, Indian Institute of Technology (B.H.U.) Varanasi, India. He is currently working as an assistant professor in the Department of Computer Science & Engineering at Motilal Nehru National Institute of Technology Allahabad, Prayagraj, India. He has been engaged in teaching and research for more than seventeen years. He has published many research papers in various conferences and peer-reviewed journals including IEEE Transactions, Elsevier, Springer, and Wiley. He has served in several program committees of national and international conferences, journals and workshops. He has chaired technical sessions in many International Conferences and delivered expert lectures at various Institutes and Universities in India. He is a Senior Member of IEEE, Member of ACM, and a Life Member of the Computer Society of India. His research interests include Analysis of Algorithms, Parallel & Distributed Computing, and Cloud and Fog Computing.

Rama Shankar Yadav is currently working as a Professor in the Department of Computer Science & Engineering at Motilal Nehru National Institute of Technology Allahabad, Prayagraj, India. He received Ph.D. degree from IIT Roorkee, M.S. degree from BITS Pilani and B.Tech. degree from I.E.T., Lucknow. He has extensive research and academic experience. Before joining MNNIT Allahabad, he had worked in leading institutions such as GBPEC, Pauri Garhwal, BITS, Pilani. He has authored more than 80 research papers in National/International conference and referred Journals/Book chapters. His areas of interest are Real-Time System, Embedded System, Fault Tolerant System, Energy Aware Scheduling, Computer Architecture, Distributed Computing and Cryptography.