IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 15 February 2023, accepted 1 March 2023, date of publication 10 March 2023, date of current version 22 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3255781

RESEARCH ARTICLE

Task Scheduling in Cloud Computing: A Priority-Based Heuristic Approach

SWATI LIPSA[1], RANJAN KUMAR DASH[1], (Senior Member, IEEE), NIKOLA IVKOVIĆ[2], AND KORHAN CENGIZ[3], (Senior Member, IEEE)

[1]Department of Information Technology, Odisha University of Technology and Research, Bhubaneswar 751029, India
[2]Faculty of Organization and Informatics, University of Zagreb, 10000 Zagreb, Croatia
[3]Department of Computer Engineering, Istinye University, 34010 Istanbul, Turkey

Corresponding author: Ranjan Kumar Dash (rkdash@outr.ac.in)

This work was supported by the Croatian Science Foundation under Project IP-2019-04-4864.

ABSTRACT In this paper, a task scheduling problem for a cloud computing environment is formulated by using the M/M/n queuing model. A priority assignment algorithm is designed to employ a new data structure named the waiting time matrix to assign priority to individual tasks upon arrival. In addition to this, the waiting queue implements a unique concept based on the principle of the Fibonacci heap for extracting the task with the highest priority. This work introduces a parallel algorithm for task scheduling in which the priority assignment to task and building of heap is executed in parallel with respect to the non-preemptive and preemptive nature of tasks. The proposed work is illustrated in a step-by-step manner with an appropriate number of tasks. The performance of the proposed model is compared in terms of overall waiting time and CPU time against some existing techniques like BATS, IDEA, and BATS+BAR to determine the efficacy of our proposed algorithms. Additionally, three distinct scenarios have been considered to demonstrate the competency of the task scheduling method in handling tasks with different priorities. Furthermore, the task scheduling algorithm is also applied in a dynamic cloud computing environment.

INDEX TERMS Fibonacci heap, cloud computing, preemptive scheduling, priority queue, task scheduling, virtual machine.

I. INTRODUCTION

Cloud computing refers to the provision of on-demand computing resources, including anything from software to storage and processing power [1]. Due to technological improvements, a wide range of sectors is now adopting cloud computing applications to improve and streamline their operations. These applications are accessible from different geographical locations at any given time. It provides diverse services across multiple sectors, including data storage, social networking, education, medical management, and entertainment, among others. Cloud computing services fall into three broad categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). A public cloud is made available to the general public on a pay-as-you-go basis, while a private cloud refers to a company's or organization's internal data centers that are not accessible to the general public.

A cloud permits workloads to be easily installed and scaled owing to the fast provisioning of a virtual or physical machine [2]. In a cloud computing environment, multiple virtual machines (VMs) can share physical resources (CPU, memory, and bandwidth) on a single physical host, and multiple VMs can share a data center's bandwidth using network virtualization. As there are usually many user requests, a significant challenge is to efficiently schedule user requests with a minimal turnaround time for tasks related to user demands.

Task scheduling is used to schedule tasks for optimum resource utilization by allocating specific tasks to certain resources at specific times. Tasks are computational activities that may necessitate diverse processing skills and resource requirements such as CPU, memory, number of nodes, network bandwidth, etc. Each task may have different

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Ching Ying.

criteria, such as task priority, a deadline for completion, an estimated execution time, and so on. The task scheduling problem covers two categories of users: cloud providers and consumers. Cloud consumers seek to run their tasks to solve problems of various scales and levels of complexity, whereas resources from cloud service providers will be used to execute custom tasks. Cloud consumers will benefit from prudent resource selection and aggregation, while cloud providers will gain from optimal resource utilization. Since many users and applications share device resources, appropriate task scheduling is essential and crucial [7].

Task scheduling in cloud computing includes two basic types of scheduling approaches: preemptive and non-preemptive scheduling methods. The VM is assigned to the tasks for a specified amount of time in preemptive scheduling, whereas in non-preemptive scheduling, the VM is assigned to the task until it finishes.

Task scheduling and resource management allow cloud providers to optimize revenue and resource usage to the maximum extent possible. The scheduling and distribution of resources appear to be significant bottlenecks in the effective utilization of cloud computing resources. This bottleneck in efficient scheduling in turn inspires researchers to explore task scheduling in cloud computing. The fundamental principle behind task scheduling is to arrange tasks to attenuate time loss and boost performance. Systems without a proper task scheduling feature may exhibit a longer waiting period and even compel the less important tasks towards starvation. Hence, scheduling strategies must include important parameters like the nature, size, and execution time of tasks as well as the availability of computing resources when calculating task priority and finalizing scheduling decisions.

In this paper, we address a time-efficient heuristic model for task scheduling in a cloud computing environment. The remainder of the paper is organized as follows: Section II gives an overview of existing approaches. Section III describes the proposed task scheduling model. Section IV specifies an illustration of the proposed algorithms. Section V demonstrates the experimental results and their comparisons with the existing algorithms, and finally, the entire paper is concluded in Section VI.

II. RELATED WORKS

In the work [3], the task scheduling problem is designed using an integer linear program (ILP) formulation. The work carried out in [4] finds the most suitable task for execution by integrating the pairwise comparison matrix technique and the Analytic Hierarchy Process (AHP). They use these techniques to rank the tasks for effective resource allocation. They also use an induced matrix to enhance the consistency among the tasks.

In the paper [5], a multi-objective optimization problem for scheduling tasks among VMs is formulated by considering the parameters such as execution cost, transfer time, power consumption, and queue length. They use Multi-Objective Particle Swarm Optimization (MOPSO) and Multi-Objective Genetic Algorithm (MOGA) to implement their model in the Cloudsim toolkit environment. They conclude that MOPSO is a better method than MOGA for solving such problems. In order to minimize the overall makespan of a set of tasks, the model [6] uses a Dynamic Adaptive Particle Swarm Optimization algorithm (DAPSO). They also propose a task scheduling algorithm that incorporates both the Dynamic Adaptive PSO (DAPSO) algorithm and the Cuckoo Search (CS) algorithm and is called MDAPSO. According to the simulation results provided in this paper, MDAPSO and DAPSO perform better than PSO.

The work carried out in [7] defines a divisible task scheduling problem by using a nonlinear programming approach. A divisible load scheduling algorithm has been proposed in this work that takes network bandwidth availability into account. The authors [8] propose an optimal task scheduling algorithm by using Discrete Symbiotic Organism Search (DSOS) algorithm. This work reveals that DSOS is well suited to handling large-scale scheduling problems as it converges more quickly than PSO for a wide search area.

A priority-based task scheduling algorithm is presented in [9]. The priority is assigned to different tasks as per their classification based on the deadline and minimum cost. The work [10] presented a fault tolerance awareness scheduling strategy based on the dynamic clustering league championship algorithm (DCLCA), which would reflect the currently available resources and reduce the premature failure of autonomous activities. The paper [11] presents a dynamic priority-based job scheduling algorithm where the priority is dynamically set considering CPU usage, IO usage, and job criticality. This algorithm decreases issues related to starvation.

The work carried out in [12] uses Orthogonal Taguchi Based-Cat Swarm Optimization (OTB-CSO) to optimize the overall task processing time. A Scheduling Cost Approach (SCA) is an approach proposed in [13] that can determine the cost of CPU, RAM, bandwidth, and storage, with tasks being prioritized based on the user's budget. They compared their work with FCFS and Round Robin Scheduling and found their work to be better than FCFS and the Round Robin Scheduling algorithm. The performance of different scheduling approaches is discussed in [14] considering different parameters like resource utilization, energy consumption, etc.

The authors [15] propose a grouped task scheduling (GTS) algorithm for task scheduling in a cloud computing environment. They use quality of service as the scheduling criteria. In this work, the tasks are first grouped into five different categories and then scheduled in increasing order of their execution time. In [16], various scheduling strategies have been discussed for the effective usage of resources so to minimize the power consumption and cost of processing.

A computational framework is proposed in [17] for cloud service selection and evaluation. This framework is an

integration of the Analytic Hierarchy Process (AHP) and Technique for Order Preference having Similarity to Ideal Solution (TOPSIS). The task scheduling algorithm presented in [18] uses the Genetic Gray Wolf Optimization Algorithm (GGWO), where the GGWO algorithm is a combination of the Gray Wolf Optimizer (GWO) and Genetic Algorithm (GA).

Four task scheduling algorithms, namely CZSN, CDSN, CDN, and CNRSN have been proposed by [19] for multi-cloud environments that are heterogeneous by nature. They use normalization techniques like z-score and decimal scaling to propose the first two algorithms, respectively, while the other two algorithms are based on distribution scaling and nearest radix scaling, respectively.

A multi-objective hybrid strategy, which amalgamates the desirable features of two algorithms, the bacteria foraging (BF) algorithm and the genetic algorithm (GA), is proposed in [20] for task scheduling in cloud computing. In the work [21], the proposed scheduling algorithm is an integration of four techniques, namely, the modified analytic hierarchy process (MAHP), longest expected processing time preemption (LEPT), bandwidth aware divisible scheduling(BATS)+BAR optimization and divide-and-conquer methods. The method [22] uses a queue to store and manage all the incoming tasks to the system. The task allocation is performed by assigning a priority to each task. They use Hybrid Genetic-Particle Swarm Optimization (HGPSO) algorithm for different tasks assignment.

The work carried out in [23] applies the mean grey wolf optimization algorithm to minimize the overall makespan and energy consumption in cloud computing networks. Gravitational Search and Non-dominated Sorting Genetic Algorithm (GSA and NSGA) have been found in [24] to select the candidate VMs. The cloudlet scheduling problem has been solved by applying the monarch butterfly optimization algorithm [25].

The paper [26] presents an improved version of the Moth Search Algorithm (MSA) for solving the cloud task scheduling problem. Similarly, the scheduling algorithm [27] uses a modified GA algorithm integrated with greedy strategy (MGGS). An intelligent meta-heuristic algorithm has been presented in [29] that combines the imperialist competitive algorithm (ICA) and the firefly algorithm (FA). Genetic algorithm has been used in [28] for task allocation among the VMs in cloud computing. They use the CloudSim toolkit for simulation purposes. A new method based on the nature of the grey wolf has been proposed in [30] to select the best scheduling algorithm. In the proposed method [31], tasks are prioritized considering two factors, such as consumer preferences and pre-defined criteria, using the Best-Worst Method (BWM) as a light Multi-Criteria Decision-Making (MCDM) technique for task scheduling in cloud computing.

The paper [32] proposes a new heuristic method termed Efficient Resource Allocation with Score (ERAS) for task scheduling in a cloud computing network. A supervised machine learning technique has been used in [33] to select the best scheduling algorithm for effectively allocating tasks to VMs.

The method proposed in [34] uses the Bumble Bee Mating Optimization (BBMO) algorithm to optimize the makespan of the tasks. Similarly, Total Resource Execution Time Aware Algorithm (TRETA) can be found in [35] for solving the task scheduling problem. In the work [36], a modified Harris Hawkes Optimization (HHO) along with the simulated annealing (SA) algorithm are used to propose the HHSOA approach for job scheduling in cloud computing. Game theory has been applied in [37] to propose a task scheduling algorithm while considering the reliability of the balanced tasks. In the work [38], the task scheduling problem is defined as a multi-objective optimization model, and its solution strategy includes the whale optimization algorithm (WOA). Further, an Improved WOA is also proposed in this paper for cloud task scheduling (IWC). A parental prioritization earliest finish time (PPEFT) based scheduling algorithm has been proposed in [39] for a heterogeneous environment. The tasks are scheduled in the parental priority queue (PPQ) on the basis of downward rank and parental priority. The simulated results show this algorithm outperforms HEFT and CPOP in terms of cost and makespan of schedules. An intelligent scheduling mechanism has been proposed in [40] that uses genetic algorithm based multiphase fault tolerance (MFTGA) to schedule tasks over VMs. This strategy works through four phases namely, the individual phase, local phase, global phase, and fault tolerance phase. This scheduling strategy was compared against GA and Adoptive Incremental Genetic Algorithm (AIGA) in terms of execution time, memory usage, overall energy consumption, SLA violation, and cost. This comparison reveals the proposed strategy to have better performance than its counterpart methods discussed above. Ajmal et al. [41] proposed a hybrid task scheduling approach that combines genetic algorithm and ant colony optimization for the allocation of tasks to various VMs. This work substantially decreased both execution time and data center cost by an amount of 64% and 11% respectively.

The works discussed so far are summarized in Table 1. The abbreviations used in Table 1 are P - Preemptive, NP - Non-preemptive, TAA - Task arrival assumption, E - Evolutionary, and NE - Non-evolutionary. This table suggests the following issues and challenges in task scheduling for a cloud infrastructure that must be taken into consideration while designing an efficient task scheduling model:

1) Efficiency of priority algorithms: Priority-based scheduling algorithms must be time-efficient to compute the priority of tasks while taking into account important decision criteria such as waiting time, execution time, etc.
2) Adoption of queuing model: The queuing model has been adopted to accommodate the tasks in the system, but its implementation details are missing in the literature. Further, the general queue cannot be used

here due to the randomness and dynamic nature of the tasks.

3) Nature of task scheduling: Most of the methods only concentrate on the non-preemptive allocation of tasks among the VMs. However, in reality, we require a method that can handle both the non-preemptive and preemptive natures of task scheduling. Preemption of tasks can occur due to many reasons as explained in [42] even though it may cause some overheads. However, in terms of waiting time, preemption of tasks occurs due to minimizing the overall waiting time of the tasks.

The main objectives of the work carried out in this paper, which is an attempt to resolve the above-enumerated issues, are presented below:

1) The task scheduling problem is formulated by using the queuing model to minimize the overall waiting time for each task.
2) A new data structure, namely the waiting time matrix, is defined in the proposed scheduling model.
3) Another algorithm is introduced in this work to assign priority to each incoming task based on its size.
4) A detailed implementation of the waiting queue is presented, which uses a Fibonacci heap data structure.
5) A new parallel algorithm is proposed for the non-preemptive and preemptive scheduling of tasks.

III. PROPOSED TASK SCHEDULING MODEL

The proposed task scheduling model for a cloud computing environment consists of the following mechanisms, which are depicted in Figure 1:

1) An algorithm to assign the priority to each task upon their arrival in the system, i.e., Priority assignment to tasks (PAT)
2) A priority queue is implemented using a Fibonacci heap
3) Task Scheduling Algorithms (both non-preemptive and preemptive)
4) A virtual machine associated with a queue called FIFO.

The task scheduling problem is NP-hard and is devised in the following sub-section so that the overall waiting time of each task must be minimized.

A. PROBLEM FORMULATION

Let there be n number of identical virtual machines denoted by $S_1, S_2 \cdots, S_n$ at one data center. Users make the requests by sending tasks to the cloud system to get the desired output. Let total number of tasks $T_1, T_2, \cdots, T_m$ at a particular time be m. The time required to send a task to the cloud and receive back the results is defined as transmission time (T_T). Similarly, the time taken by the VM to complete the execution of a task is called processing time (T_P). Thus, the turnaround time of the task can be defined as:

$$TT(T_i) = T_T(T_i) + T_P(T_i) \tag{1}$$

If a task needs to wait before its execution, equation (1) can be rewritten as:

$$TT(T_i) = T_T(T_i) + T_P(T_i) + T_W(T_i) \tag{2}$$

where $T_W(T_i)$ is the waiting time of task T_i.

In a cloud computing environment, the inter-arrival time and service time of the tasks can be assumed to be exponentially distributed [48] and thus follow $M/M/n$ queue model [49]. Let α be the mean rate of arrival and β be the mean service rate, then the probability that VMs are busy [44] is expressed as:

$$P_{Busy} = \frac{\alpha}{n \times \beta} \tag{3}$$

The mean number of tasks in the queue can be computed by using the following equation:

$$Q_M = \frac{P_0(\frac{\alpha}{\beta})^n P_{Busy}}{n!(1 - P_{Busy})^2} \tag{4}$$

where P_0 is the probability that there is 0 number of tasks in the system [44] and is expressed as:

$$P_0 = \frac{1}{\sum_{i=0}^{n-1} \frac{(nP_{Busy})^i}{i!} + \frac{(nP_{Busy})^n}{n!(1-P_{Busy}}} \tag{5}$$

The mean waiting time [44] in the queue can be calculated by using equation(4) i.e.

$$T_W = \frac{Q_M}{\alpha} \tag{6}$$

Thus, the turnaround time of a task can be expressed by using equations(2) and (6) as:

$$TT(T_i) = T_T(T_i) + \frac{1}{\beta(T_i)} + \frac{Q_M(T_i)}{\alpha} \tag{7}$$

The turnaround time of a task (T_i) can be minimized by minimizing the mean waiting time of the task in a queue since the transmission time and processing time of a task is normally fixed for a task. Thus, the minimization problem can be formulated as follows:

$$\text{Minimize: } \sum_{i=1}^{m} (\frac{Q_M(T_i)}{\alpha}) \tag{8}$$

such that the following constraints can be satisfied:

$$1 \leq Q_M \leq m \tag{9}$$

$$\alpha_l \leq \alpha \leq \alpha_u \tag{10}$$

and

$$P_0, P_{Busy} < 1 \tag{11}$$

The constraint (9) ensures that there must be an upper bound(m) to the mean number of tasks present in the system. The second constraint i.e. (10) defines a lower and upper bound to the task arrival rate without which the system will become highly volatile and unstable. The values of

TABLE 1. Summary of related work.

Paper #	Objective	Technique used	P/NP	E/NE
[3]	Workflow scheduling	Integer Linear Program(ILP)	NA	E
[4]	Task-oriented resource allocation	Analytic Hierarchy Process(AHP)	NP	NE
[5]	Minimize task transfer time, task execution cost, power consumption, and task queue length for task scheduling	Multi-Objective Particle Swarm Optimization (MOPSO) and Multi-Objective Genetic Algorithm (MOGA)	NP	E
[6]	Minimize makespan and optimum resource utilization	Dynamic Adaptive Particle Swarm Optimization algorithm (DAPSO) and MDAPSO	NP	E
[7]	Task scheduling	Nonlinear programming model	NA	NE
[8]	Task scheduling	Discrete Symbiotic Organism Search (DSOS) algorithm	NA	NE
[9]	Task scheduling	Priority with a round-robin scheduling scheme	NA	NE
[10]	Fault tolerance aware scheduling	Dynamic clustering league championship algorithm (DCLCA) scheduling technique	NP	E
[11]	Reduce starvation	Starvation Optimizing Scheduler	NP	NE
[14]	Priority based scheduling	Hungarian method	NP	NE
[15]	Task scheduling	Grouped Tasks Scheduling (GTS) algorithm	NP	NE
[17]	Cloud service selection	AHP and Technique for order preference by similarity to ideal solution (TOPSIS)	P	NE
[18]	Multi-objective function	Genetic Gray Wolf Optimization Algorithm (GGWO) by combining Gray Wolf Optimizer (GWO) and Genetic Algorithm (GA)	NA	E
[19]	Task scheduling	Cloud z-score normalization (CZSN) algorithm, Cloud decimal scaling normalization (CDSN) algorithm, Cloud distribution scaling normalization (CDN) algorithm and Cloud nearest radix scaling normalization (CNRSN) algorithm	NA	NE
[20]	Task scheduling	GA with Bacterial Foraging(BF) algorithms	NA	E
[21]	Task scheduling and resource allocation	Modified analytic hierarchy process (MAHP), Bandwidth aware divisible scheduling (BATS)+BAR optimization	P	NE
[23]	Minimize makespan and energy consumption	Mean grey wolf optimization algorithm	NP	NE
[24]	Task scheduling	Gravitational Search and Non-dominated Sorting Genetic Algorithm (GSA and NSGA)	NP	E
[25]	Cloudlet scheduling	Original and hybridized monarch butterfly optimization algorithm	NP	E
[26]	Minimize makespan	Moth Search Algorithm (MSA) using the Differential Evolution (DE)	NP	NE
[27]	Task scheduling	Modified genetic algorithm (GA) combined with greedy strategy (MGGS)	NP	E
[28]	Task scheduling	Genetic Algorithm based efficient task allocation approach	NP	NE
[29]	Load balancing	Intelligent meta-heuristic algorithm based on the combination of Imperialist competitive algorithm (ICA) and Firefly algorithm (FA)	NP	NE
[31]	Task scheduling	Multi-Criteria Decision-Making (MCDM) methods	P	NE
[32]	Task scheduling	Efficient Resource Allocation with Score (ERAS)	NA	NE
[33]	Task scheduling	Supervised learning algorithms such as logistic regression and decision trees	NP	NE
[34]	Minimize makespan	Bumble Bee Mating Optimization (BBMO) algorithm	NP	NE
[36]	Minimize makespan	Harris hawks optimization (HHO) algorithm based on the simulated annealing (SA)	NP	E
[37]	Task scheduling	Cooperative game model	NA	NE
[38]	Task scheduling	Improved WOA for Cloud task scheduling (IWC)	P	E
[39]	Task scheduling	Parental prioritization earliest finish time (PPEFT) based scheduling algorithm	NP	NE
[40]	Intelligent task scheduling	GA based multiphase fault tolerance (MFTGA)	NP	E
[41]	Task scheduling	Hybrid task scheduling approach that combines GA and ant colony optimization	NP	E

the lower and upper bound to the task arrival rate are system dependent [44](however, this work suggests any value between 1 and 65 can be a profitable task arrival rate measured per second). Both the probability terms in constraint (11) are tightly bound by 1 otherwise it will lead to an indefinite turnaround time.

B. PRIORITY ASSIGNMENT TO TASKS (PAT)

Every task remains in the queue after its arrival in the system. The priority of each task must be computed for its subsequent execution while minimizing its waiting time. Thus, if two tasks (T_i and T_j) are waiting for VM with priority values x and y respectively, the task T_i is executed

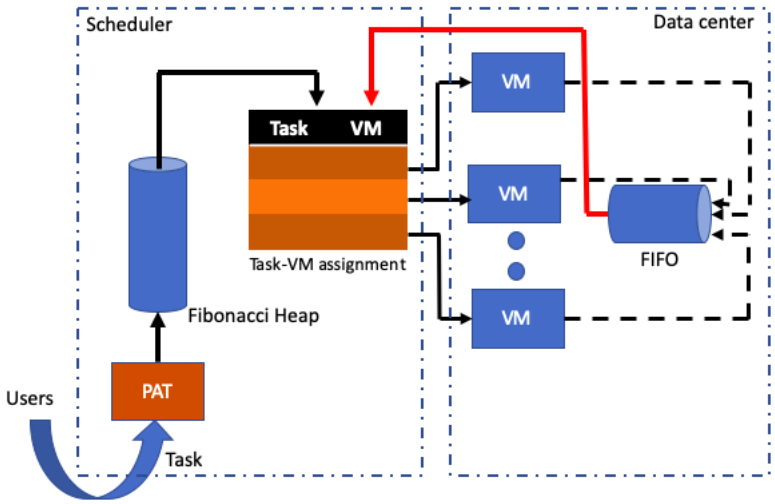

FIGURE 1. Proposed scheduling model.

before T_j *iff* $x > y$ to minimize overall waiting time of the tasks i.e.

$T_i \succ T_j$ *iff* $x > y$, where $\succ$ represents the ordering of task.

The processing time is the aspect to consider when prioritizing tasks since it depends largely on the size of the tasks in almost all cases. The following heuristic function [47] is used to approximate the execution time of each task:

$$P_T(T_i) = \frac{size(T_i) \times n \times 8}{MIPS} \tag{12}$$

where n specifies the n-bit architecture and $MIPS$ represents a million instructions per second.

A new data structure called a waiting time matrix is proposed whose elements are defined below:

1) The diagonal elements are set to 1.
2) Assuming a sequential execution of the tasks as per their arrival, the waiting time for each task is calculated and placed in the upper part of the diagonal of the matrix. e.g. if the ordering of the tasks is $T_1 \succ T_2 \succ T_3 \succ T_4$,
the waiting time for $W_T(T_1) = 0$,
$W_T(T_2) = P_T(T_1)$,
$W_T(T_3) = P_T(T_1) + P_T(T_2)$
and $W_T(T_4) = P_T(T_1) + P_T(T_2) + P_T(T_3)$

$$\begin{bmatrix} 1 & W_T(T_2) & W_T(T_3) & W_T(T_4) \\ - & 1 & W_T(T_3) & W_T(T_4) \\ - & - & 1 & W_T(T_4) \\ - & - & - & 1 \end{bmatrix}$$

3) The lower part of the diagonal of this matrix represents the reverse ordering of the tasks and is filled with the reciprocal value of the waiting time.

Algorithm 1 Priority_Assignment_to_Task(Task)

```
while True do
  for each task T_k do
    Calculate the waiting time W_T(T_k)
  end for
  Generate the waiting matrix WM
  Compute eigen vector (egvt) and eigen value (egv) of WM
  let λ_max = max(egv)
  compute consistency index (CI)= (λ_max − m)/(m − 1)
  compute consistency ratio(CR)= CI/RI
  if CR < 0.1 then
    break
  else
    decrease W_T(T_k) (i.e. the task with maximum waiting time)
  end if
end while
for k=1 to m do
  Priority(T_k)=1- maximum eigen value of egvt(k)
end for
```

Thus, the waiting matrix is represented as:

$$\begin{bmatrix} 1 & W_T(T_2) & W_T(T_3) & W_T(T_4) \\ \frac{1}{W_T(T_2)} & 1 & W_T(T_3) & W_T(T_4) \\ \frac{1}{W_T(T_3)} & \frac{1}{W_T(T_2)} & 1 & W_T(T_4) \\ \frac{1}{W_T(T_4)} & \frac{1}{W_T(T_3)} & \frac{1}{W_T(T_2)} & 1 \end{bmatrix}$$

Time complexity of Priority_Assignment_to_Task

The while loop (step-1) can be executed up to a number of tasks i.e. m times in the worst case for which the waiting time is decreased for each task. The calculation of the waiting time of m tasks involves a worst-case time complexity of (m^2)which supersedes all other operations of

the algorithms. Thus, the worst-case time complexity of the proposed Priority_Assignment_to_Task algorithm is found to be (m^3).

From the above discussion, the following observation can be made for the priority value computed for a task:

Lemma 1: The priority value for each task lies within 0 and 1.

Proof: As the priority is assigned based on the eigenvalue and eigenvector, it must lie between 0 and 1.

C. REPRESENTATION OF THE WAITING QUEUE

The task, upon its arrival in the system, will be assigned a priority using the algorithm and will be appended to the queue. Since each task is assigned a certain priority, a priority queue is preferred here over any other type of queue. This priority queue is implemented by using a Fibonacci heap [24] due to the following facts:

1) It has a height of h when it contains 2^h number of nodes and thus accommodates a very large number of tasks (Table 2).
2) The amortized time complexity is proven to be substantially more efficient than that of many other priority queue data structures. An explanation of amortized time complexity can be found in [46].
3) The worst case time complexity of basic operations such as creation, insertion, and the union is $\mathcal{O}(1)$ while its extraction of maximum requires $\mathcal{O}(m)$.

In Fibonacci heaps [45], each node x contains a pointer p[x] to its parent, as well as a pointer child[x] to any of its children. The children of x are connected together in the child list of x through a circular doubly linked list. Each child y in a child list has left[y] and right[y] pointers that point to y's left and right siblings, respectively. Left[y] = right[y] = y, if node y is the only child.

Initially, when the Fibonacci heap (H) is empty, new tasks based upon their priority values are inserted into the Fibonacci heap by Fib_Heap_Insertion(H, element). Fib_Heap_Union(H_1, H_2) performs the union operation between two heaps (H_1, H_2). The extraction of the maximum element of the Fibonacci heap is performed by Fib_Heap_Extraction_Max(H). To reduce the number of trees in the Fibonacci heap, the heap is rebuilt by using Rebuild(H). An example of the same is presented in Figure 2. The details of this operation can be found in [46].

Fib_Heap_Extraction_Max(H) is the most significant operation in the Fibonacci heap. This process involves removing the node from the heap that has the highest value and readjusting the heap. This operation proceeds through the steps illustrated in the following example.

1) Initially, the root nodes are 0.84, 0.99, and 0.96.
2) The max-pointer marks 0.99 as the maximum valued node.
3) *Fib_Heap_Extraction_Max*(H) extracts the maximum valued node, i.e., 0.99.

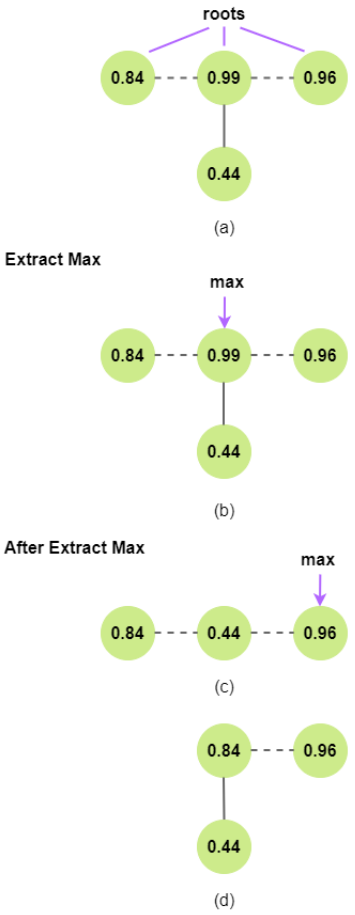

FIGURE 2. Graphical representation of heap data structure used in priority queue.

TABLE 2. Number of tasks vs the height of Fibonacci heap.

No. of tasks(m)	Height(h)
≈ 1 billion	26
≈ 10 billions	30
≈ 100 billions	33
≈ 1000 billions	36

4) This extraction operation makes the child node 0.99, i.e., 0.44, the root node.
5) The max-pointer now points to node 0.96.
6) As 0.84 and 0.44 have the same degree, they are united, making 0.84 the root and 0.44 its child.
   Thus, the final heap is built after the extract-max operation.

These steps are repeated for each extract-max operation to be performed subsequently.

D. PROPOSED TASK SCHEDULING ALGORITHM

When a task enters the system, its priority is computed and added to the Heap queue. The scheduling algorithm must allocate VM to the task with the highest priority. This allocation is implicitly true unless any new task arrives with

a higher priority than the currently running tasks. When it comes to accomplishing a certain task, the former adheres to its non-preemptive nature, whereas the latter adheres to its preemptive nature. Moreover, both time and memory are saved by adopting non-preemption of tasks. However, preemption of tasks though being less efficient in terms of CPU time and memory can not be avoided. Hence, it is necessary to switch between these methods for efficient task completion. However, this switching cannot be accomplished by sequential algorithms in which priority is calculated first and then the task is scheduled. Thus, the priority assigned to the task along with the building of the Heap must be executed concurrently with the scheduling algorithm. As initially there is no task in the system, the heap must be built first and then only the tasks can be scheduled, which in turn requires that for the arrival of each α number of tasks, the priority assignment and Heap construction must be accomplished while during the next the arrival of α number of tasks, the task must be allocated. In order to the above discussed steps, a parallel scheduling algorithm is proposed for optimizing the overall waiting of tasks. The algorithm uses the following operations and data structures:

1) The status of each S_i is defined as: $S_i.Busy() = 1$ if the VM S_i is currently executing a task(T_j) else $S_i.Busy() = 0$ indicating it is free.
2) Assign the task T_j to the VM S_i, i.e., $S_i(T_j)$
3) $ts-$ the time stamp recorded for each task as it arrives in the system
4) $W_T(T_k)-$ waiting time of i^{th} task
5) $W_T(S_i, T_k)-$ i^{th} task waits for i^{th} S, where S is the VM
6) FIFO - a queue to store the preempted tasks by a VM
7) Heap and FIFO are the global data structures

During the initial execution of the proposed algorithm, priority is assigned to the tasks. Then, the Heap is built in terms of the descending order of priority values of the tasks. The building of the heap is executed concurrently with the scheduling of the tasks (while $Heap \neq \phi$). The first step of this part of the algorithm delays its execution by a time of α ms in order to allow the building of heap of α number of tasks. In the Parallel_Task_Scheduling_algorithm, the assignment of priority to tasks and the building of Heap proceeds in parallel with the scheduling of tasks. If the arrival rate is α, α number of tasks are assigned with priority and then Heap is built for these tasks. The timestamp of each task(ts) is recorded. If T_j is the extracted task from the Fibonacci heap, its ts is compared against each $T_k \epsilon Heap$ only when there are no free VMs. The truth of this condition leads to task preemption and accordingly Preemptive_task_scheduling is executed. Failing of the above condition executes Non-preemptive_task_scheduling. The last *for* loop of this algorithm is used to calculate the waiting time of each task by considering across the VMs (i.e. $W_T(S_i, T_k)$ if $S_i(T_k) =$ True). The flow control of the proposed algorithm for task scheduling is depicted in Figure 3.

Algorithm 2 Parallel_Task_Scheduling_algorithm(Task)

```
do in parallel
  for arrival of each α no. of tasks do
    try{
    Priority_Assignment_to_Task(Task) where Task=
    T1, T2, ..., Tα
    }
    if Heap=NIL then
      try{ Fib_Heap_Insertion(Heap,T) for each new
      task(T) }
      WT(Tk) = 0 for each Tk∈Heap
    else
      for new task(Tj) do
        try{ Fib_Heap_Insertion(H, Tj) }
        WT(Tj) = 0
      WT(Si, Tk) = 0 for each Tk∈H and Si∈S
      try{ Fib_Heap_Union(Heap,H) }
    catch()
    {
    report the error message
    Rebuild(Heap) and Restart execution
    }
  while Heap ≠ ϕ do
    delay(α ms)
    try{
    Tj = Fib_Heap_Extraction_Max(Heap)
    }
    if ts(Tj) ≥ ts(Tk) ∀ Tk∈Heap and ∀ Si = 1 and
    key[Tj] > key[Si(Tk)]∀i, k then
      try{ execute Preemptive_task_scheduling
      }
    else
      try{ execute Non-preemptive_task_scheduling
      }
    for each VM Si do
      for each task Tk do
        WT(Tk) = WT(Tk) + WT(Si, Tk) if
        Si(Tk)=True
    catch()
    {
    report the error message
    Rebuild(Heap) and Restart execution
    }
```

Error handling mechanism in Parallel_Task_Scheduling algorithm-

In order to handle, the different types of errors such as task-specific errors, heap-specific errors, running time errors, etc, each error-sensitive step of this algorithm are embedded by a try block. The catch block used in this algorithm is used to report the errors, rebuild the heap, and then restart the execution (details can be found in [42]).

Time complexity of Parallel_Task_Scheduling_ algorithm-

The worst-case time complexity of for loop is $\mathcal{O}(\alpha)$. The worst-case time complexity of the while loop is

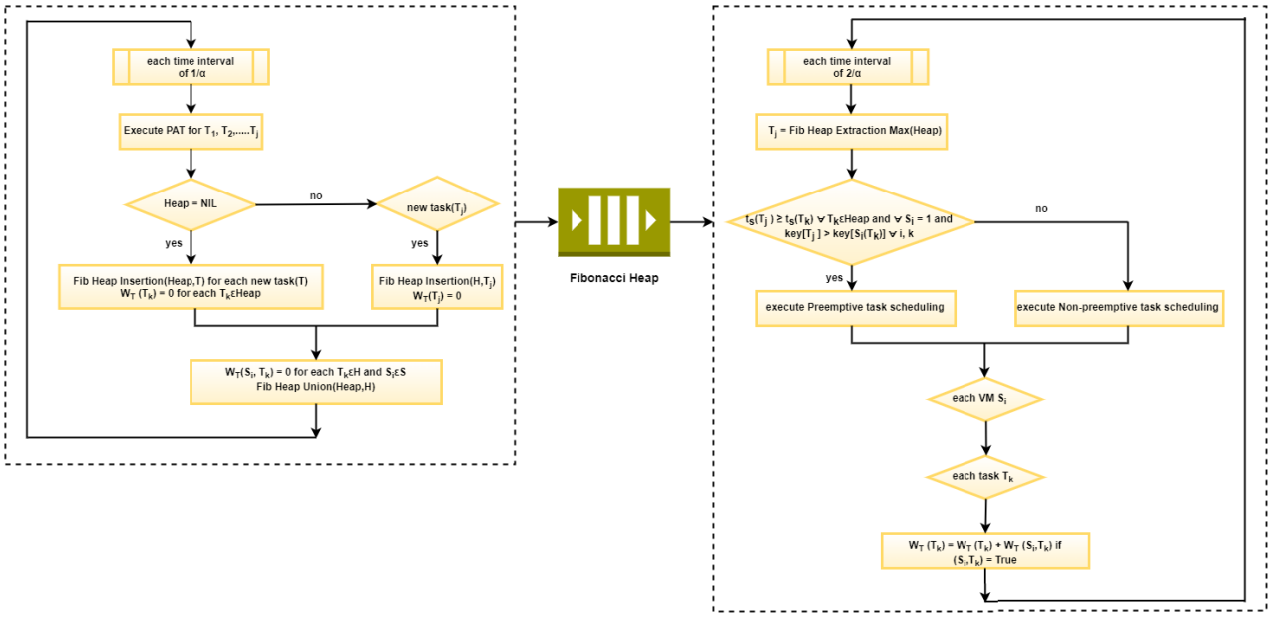

FIGURE 3. Flowchart of the proposed parallel task scheduling algorithm.

Algorithm 3 Non-preemptive_task_scheduling(Task)

```
1: bool=False
2: for each VM Si do
3:    if Si.Busy() = 0 then
4:       assign Si with Tj=Fib_Heap_Extraction_Max(Heap)
5:       bool = True
6:    else if bool=True then
7:       for each Tk∈Heap do
8:          WT(Si, Tk) = WT(Si, Tk) + PT(Si(Tj))
9:       end for
10:   end if
11: end for
```

$\mathcal{O}(|Heap| \times n)$. The heap can accommodate m number of tasks in the worst-case scenario. Thereby, the worst case time complexity of the proposed algorithm 2 i.e. Parallel_Task_Scheduling_algorithm is found to be $\mathcal{O}(m \times n)$.

1) NON-PREEMPTIVE TASK SCHEDULING ALGORITHM

Non-preemptive task scheduling involves steps mentioned in Non-preemptive_task_scheduling.

In algorithm 3, the Boolean variable bool is used to determine the waiting time of the remaining tasks (step-1). The new task must wait if any other task is assigned to any of the VM and their waiting time is updated accordingly. This happens for bool=True (steps 6-9). However, for bool=False, the tasks must also wait but their waiting time must not be updated (since these have been updated earlier during the assignment of the previous task to the VM).

Time complexity of Non-preemptive_task_scheduling()

The for loop(step-2) of the Non-preemptive_task _scheduling() can execute up to a maximum of n number of times where n is the number of VMs. The nested for loop mentioned in step-7 of this algorithm is executed up to a maximum number of times equal to the heap size. In the worst-case scenario, the heap can accommodate m number of tasks, thus, the overall worst-case time complexity of the proposed Non-preemptive_task_scheduling() algorithm is found to be $\mathcal{O}(m \times n)$.

2) PREEMPTIVE TASK SCHEDULING ALGORITHM

The steps involved in preemptive task scheduling are:

1) The preemption of tasks occurs when any task executed by the processor has less priority than the tasks which are waiting.
2) In order to ensure that no task waits for an indefinite amount of time(i.e. starvation problem), the priority of each task waiting for VM is updated by a small amount (i.e. reciprocal to its size) after each preemption operation.
   $Key[T_j] = Key[T_j] + \frac{1}{size[T_j]}$
   $size[T_j]$ is taken into account to preserve the initial priority ordering. The heap is rebuilt to update only the key values without any structural alteration.

Steps 4–11 of the proposed algorithm 4 are used to check whether any task from FIFO can be allocated to the available VM. The waiting time for each task is updated by using steps 14–16. Steps 17-29 are used for preemption of the running task and to assign the VM to the newly arrived task with higher priority than the priority of the running task.

Algorithm 4 Preemptive_task_scheduling(Task)

```
bool=False
FIFO = empty
for each VM S_i do
    if S_i.Busy() = 0 then
        T_j = Fib_Heap_Extraction_Max(Heap)
        assign S_i = max(T_j, max(FIFO))
        if key[T_j] is maximum then
            for each T_k∈FIFO do
                W_T(S_i, T_k) = W_T(S_i, T_k) + P_T(T_j)
            end for
        end if
        bool=True
    else if bool=True then
        for each T_k∈Heap do
            W_T(S_i, T_k) =W_T(S_i, T_k) + P_T(S_i(T_j))
        end for
    else if S_i.Busy() = 1 then
        if key[S_i(T_j)] ≤ key[T_k = Fib_Heap_Extraction_Max(Heap)] then
            FIFO.add(T_j)
            assign S_i with T_k
            for each T_k∈Heap do
                W_T(S_i, T_k) =W_T(S_i, T_k) + P_T(S_i(T_j))
            end for
            for each T_k∈FIFO do
                W_T(S_i, T_k) =W_T(S_i, T_k) + P_T(S_i(T_j))
            end for
        end if
    end if
end for
for each T_k∈Heap do
    Key[T_j] = Key[T_k] + 1/size[T_k]
    Rebuild(Heap)
end for
```

Time complexity of Preemptive_task_scheduling()

The for loop(step-3) of the Preemptive_task _scheduling() can execute up to a maximum of n number of times where n is the number of VMs. The nested for loop mentioned in step-14 of this algorithm is executed up to a maximum number of times equal to the heap size. In the worst-case scenario, the heap can accommodate m number of tasks, thus, the overall worst-case time complexity of the proposed Preemptive_task_scheduling() algorithm is found to be $\mathcal{O}(m \times n)$.

3) COMPLETENESS AND CONVERGENCE OF THE PROPOSED PARALLEL SCHEDULING ALGORITHM

The proposed algorithm increases the priority of the waiting tasks after every task preemption operation by a small amount, i.e., $Key[T_j] = Key[T_j] + \frac{1}{size[T_j]}$. Hence, no task will wait for a long or indefinite time, which in turn assures the convergence of the algorithm. Furthermore, the waiting time of each task is considered to compute the overall waiting time, which shows that the algorithm is complete in terms of the tasks.

TABLE 3. Four number of tasks with their size as per [21].

task	Size of task	E(P_T)
T_1	62,69,51,663	33.43
T_2	69,47,76,323	37.01
T_3	58,57,63,637	31.23
T_4	53,68,97,326	30.31

IV. ILLUSTRATION

The proposed algorithms are illustrated by taking the following four tasks (Table 3) and two VMs(S_1, S_2) with configuration as per [21]:

The estimated processing time for each of the tasks is calculated by using equation(12) and is shown in the third column of Table 3. Comparing the actual processing time [21] and estimated processing time, the heuristic adopted in this work seems to be correct(except for the task1). **Priority Assignment to Task algorithm:**

Step-5: The waiting matrix for the four tasks is

$$\begin{bmatrix} 1 & 33.43 & 70.44 & 100.75 \\ 0.0299 & 1 & 70.44 & 100.75 \\ 0.0141 & 0.0299 & 1 & 100.75 \\ 0.0092 & 0.0141 & 0.0299 & 1 \end{bmatrix}$$

Step-6: The eigenvalue and the eigenvector are calculated for the waiting time matrix WM. The absolute value of the eigenvector is shown below:

$$\begin{bmatrix} 0.77997646 & 0.77888532 & 0.77888532 & 0.77717669 \\ 0.19799301 & 0.20340335 & 0.20340335 & 0.21154905 \\ 0.02103499 & 0.02021466 & 0.02021466 & 0.01926724 \\ 0.0015625 & 0.00140796 & 0.00140796 & 0.00121732 \\ \hline 1.0005669 & 1.00391128 & 1.00391128 & 1.00921030 \end{bmatrix}$$

Step-7: $\lambda max = 9.3$
Step-8: CI = 1.76
Step-10: Since $CR < 0.1$ there is no decrease in waiting time.
Step-16: The absolute value of eigenvectors is normalized i.e. the sum of each column equals to ≈ 1. The maximum value of each eigenvector is extracted and is:

$$\begin{bmatrix} 0.7799 \\ 0.2115 \\ 0.0210 \\ 0.0015 \end{bmatrix}$$

In order to minimize the waiting time, the priority is calculated by subtracting each maximum value from 1(Table 4).

Task scheduling:
Step-2: Since, the Heap is initially Nil
Step-3: Fib_Heap_Insertion(Heap,T_1) will insert task T_1 into Heap. Subsequently, the following task will be inserted:
Fib_Heap_Insertion(Heap,T_2)
Fib_Heap_Insertion(Heap,T_3)

TABLE 4. Tasks with priority and estimated processing time.

task	Priority	$E(P_T)$
T_1	0.0220	33.43
T_2	0.7884	37.01
T_3	0.979	31.23
T_4	0.9985	30.31

Fib_Heap_Insertion(Heap,T_4)
Step-4 or 8: The waiting time of each task is set to 0.
Step-13: The root of the Heap contains task T_4 as it has the highest priority.
Step-14: Initially, $S_1.busy() = 0$
Step-17: Execute Non-preemptive_task_scheduling(Task)
Non-preemptive Task Scheduling:
Step-4: T_4 is assigned to S_1 by performing the operation *Fib_Heap_Extraction_Max*(*Heap*). Extraction of task T_4 from Heap makes T_3 its root.
Task T_3 is assigned to S_2 as its status is 0.
Step-5: The above assignments set the variable bool=True, which causes the tasks T_2 and T_1 to wait. Thus, their waiting time is updated as follows:
Step-8: $W_T(S_1, T_1) = 0 + 30.31$, $W_T(S_2, T_2) = 0 + 31.23$
$W_T(S_1, T_2) = 0 + 30.31$, $W_T(S_2, T_1) = 0 + 31.23$
When S_1 becomes free(i.e. status=0), the task T_2 is assigned to it, similarly, T_1 is assigned to S_2 when it will become free. The final waiting time for each task is:
$W_T(T_4) = 0$, $W_T(T_3) = 0$, $W_T(T_2) = 30.31$ and $W_T(T_1) = 31.23$
The total waiting time amounts to $W_T(T) = 61.54$
Preemptive Task scheduling:
Suppose during the execution of T_1 and T_2 (which has already been scheduled by a non-preemptive algorithm), a new task (T_5) with priority value (0.85) more than that of T_1 and T_2 enters the system having execution time of 30. Since the priority of the tasks is now in the order $key[T_5] > key[T_2] > key[T_1]$, the preemptive algorithm preempts the task T_1 to the waiting state and assigns VM to task T_5. Let 15 and 16 remaining times for completion of the task T_2 and T_1 respectively.
Step-5: $T_5 = Fib\_Heap\_Extraction\_Max(Heap)$
Step-18: Since $key[T_1] < key[T_5]$
Step-19: $FIFO.add(T_1)$
Step-20: Assign T_5 to S_2
Step-25: $W_T(S_2, T_1) = 31.23 + 30 = 61.23$
Once the VM S_1 completes the execution of T_2, then the scheduler checks for the priority value of the remaining tasks present in the priority queue as well as all FIFO queues. As there is only one task T_1 which is in a waiting state, it will be assigned to S_1. So, the overall waiting time of task T_1 will be $31.23 + 30 = 61.23$. Thus, the total waiting time for five tasks is $W_T(T) = W_T(T_1) + W_T(T_2) = 61.23 + 30.31 = 91.54$.

V. RESULTS AND DISCUSSION

All the algorithms are simulated by Python 3.8 in an HP-Work station with Intel Xeon W Processor(3 GHz, 10 cores, and 20 MB Intel smart cache), 192 GB DDR4 SDRAM (6×32 GB), and Ubuntu operating system. Virtual machines are created by using Oracle VM Virtual box having the configuration: 1 GB RAM, operating system - Ubuntu.

In subsection-V-A, the performance of the proposed scheduling model in terms of the overall waiting time is compared with some existing techniques like BATS, IDEA, and BATS+BAR. The proposed scheduling model is used to find the overall waiting time by considering three distinct cases and the simulated results are discussed in subsection-V-B. The behavior of the proposed algorithm in a dynamic cloud computing environment is elucidated in subsection-V-C.

A. COMPARISON

The Epigenomics tasks used in [21] have been executed by following the Epigenomics scientific workflow. Many tasks are executed in parallel to minimize the execution time. The exact execution time is mentioned in Table 5 which is as specified in [21]. However, the following are some of the limitations of [21]:

1) In reality, there is no way to know the exact execution time of a task before its actual execution. Hence, one must obtain an expected execution time in order to determine the priority to be assigned to different tasks.
2) The work done in [21] uses 20 VMs for 13 tasks, which leads to the direct assignment of VMs to the tasks. Hence, the tasks will not have any waiting time.

In order to compare the proposed PAT with BATS+BAR, the VMs configuration as specified in [21] has been adopted here,i.e., 9600 MIPS, average RAM 512 MB. The exact execution time(E_T) and expected execution time $E(E_T)$of different tasks are presented in Table 5. Although the $E(E_T)$ are not quite satisfactory when compared to their exact execution time, still it provides a better approximation for ranking the tasks in terms of some priority assignment.

In order to compare the total waiting time for these tasks, 8 VMs have been considered instead of 20.

The following observations can be inferred from Table 5:

1) The overall waiting time obtained by employing BATS+BAR over 13 tasks is 135.97 which is more than that obtained by using the proposed method i.e. 134.73.
2) The expected execution time though being an approximation can be effective enough to assign priority to different tasks so as to minimize the overall waiting time.

CPU-bound tasks are generated from [50] with size ≥ 1 *MB* to 5 MB by using Apache-airflow 2.3.4 in a Python environment to determine the efficiency of different algorithms in minimizing the waiting time for large-sized tasks.

The overall waiting time calculated by the proposed method is compared against that obtained from BATS, IDEA, and BATS+BAR for 25 tasks, 40 tasks, and 50 tasks respectively (Figure 4). Figure 5 demonstrates the waiting

TABLE 5. The priority assignment and waiting time of different tasks by BATS+BAR and Proposed PAT.

Task	Size	E_T	P_1	W_T	$E(E_T)$	P_2	W_T
Task 3	62,69,51,663	39.06	8	0	33.44	0.32	0
Task 5	69,47,76,323	38.49	10	24.56	37.05	0.28	26.46
Task 7	58,57,63,637	36.27	7	0	31.24	0.41	0
Task 9	53,68,97,326	32.29	5	0	28.63	0.34	0
Task 11	67,05,35,542	62.25	12	26.46	35.76	0.12	28.67
Task 14	40,67,28,38,798	96.91	13	32.29	2169.22	0.05	31.05
Task 16	45,23,96,996	45.6	9	23.99	24.13	0.20	24.56
Task 18	50,27,64,231	28.67	3	0	26.81	0.74	0
Task 20	62,41,88,532	24.56	2	0	33.29	0.79	0
Task 22	42,65,77,006	31.05	6	0	22.75	0.54	0
Task 24	51,58,32,878	54.87	11	28.67	27.51	0.16	23.99
Task 26	68,14,99,417	23.99	1	0	36.35	0.75	0
Task 28	44,14,51,516	26.46	4	0	23.54	0.95	0

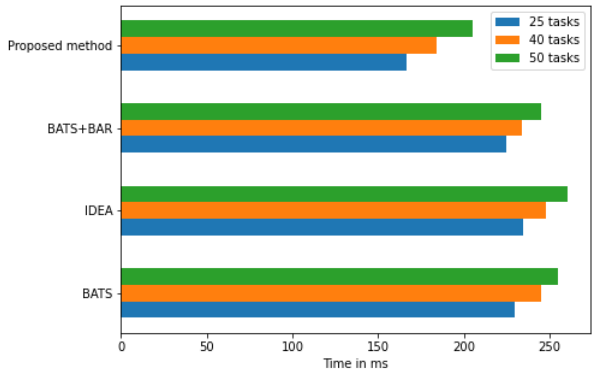

FIGURE 4. Waiting time obtained from BATS, IDEA, BATS+BAR, and proposed method.

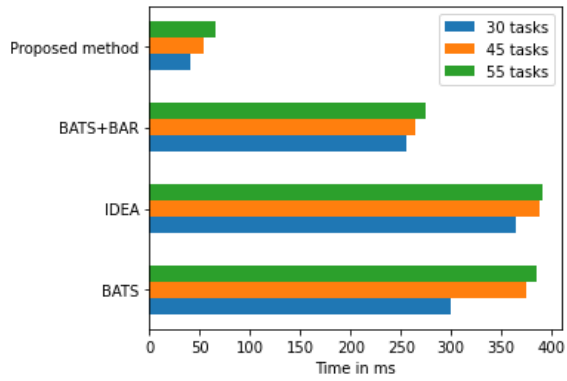

FIGURE 6. CPU time BATS, IDEA, BATS+BAR and proposed method.

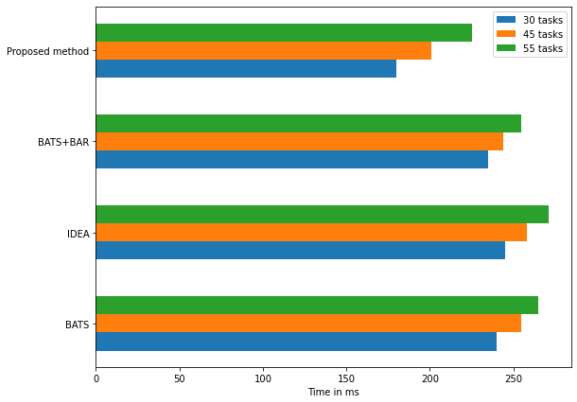

FIGURE 5. Waiting time obtained from BATS, IDEA, BATS+BAR and proposed method upon arrival of five more tasks.

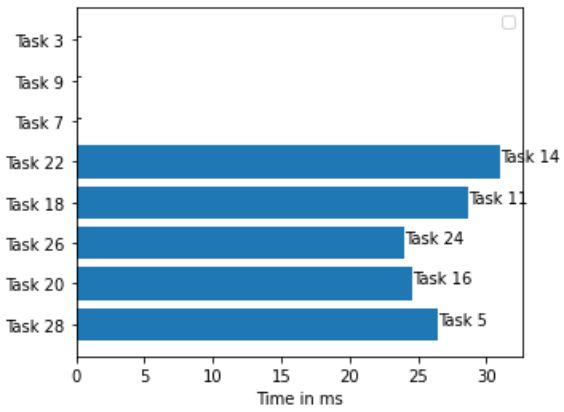

FIGURE 7. Impact of priority of tasks on waiting time.

time obtained upon arrival of five more tasks (preemptive scheduling). All these tasks are generated by using the above mentioned method. 16 number of VMs are used each with 9600 MIPS and 512 MB RAM. This figure entails BATS+BAR to be more efficient than BATS and IDEA. However, the proposed method outperforms BATS+BAR in terms of the overall waiting time.

The total CPU time in millisecond to complete the execution of all the tasks are shown for BATS, IDEA, BATS+BAR, and the proposed method considering 30 tasks, 45 tasks, and 55 tasks respectively (Figure 6). The CPU time of the proposed algorithm is very less as compared to the other mentioned methods due to its parallel nature.

The impact of high-priority tasks on the waiting time of low-priority tasks is shown in Figure 7. The low-priority tasks such as task 14, task 11, task 24, task 16, and task 5 have at least a waiting time equal to the processing time of high-priority tasks respectively.

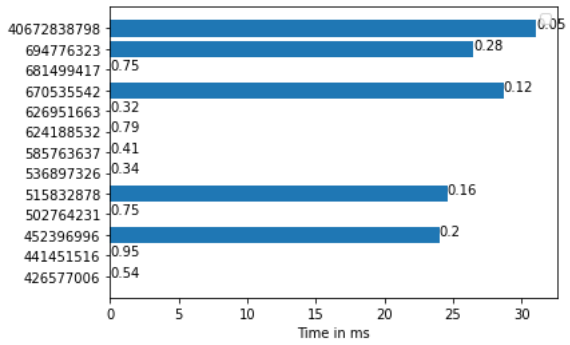

FIGURE 8. Impact of the size of tasks on waiting time of low and high priority tasks.

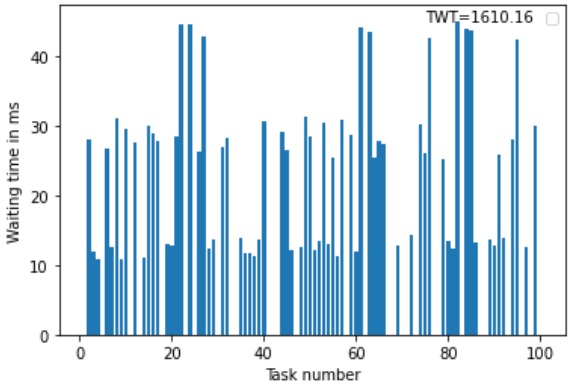

FIGURE 9. Waiting time for Case-I.

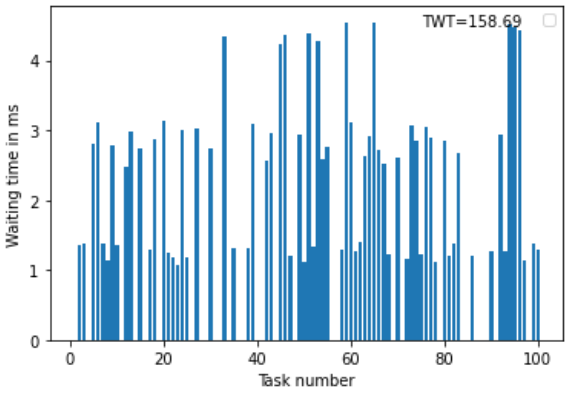

FIGURE 10. Waiting time for Case-II.

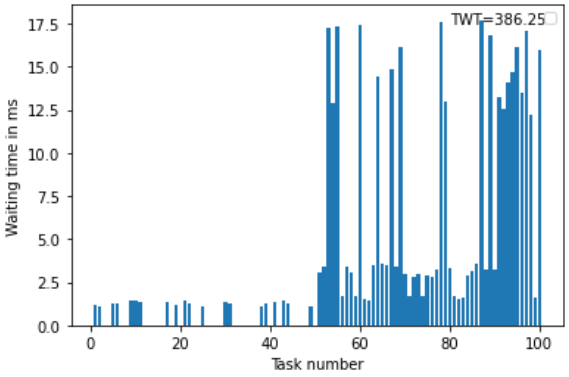

FIGURE 11. Waiting time for Case-III.

Similarly, the impact of the size of tasks on the waiting time of low and high-priority tasks is depicted in Figure 8. The priority of each task is shown in addition to its waiting time and size.

B. SCHEDULING OF TASKS IN CLOUD COMPUTING ENVIRONMENT

The following three distinct cases have been taken to show the efficiency of the proposed scheduling algorithm to assign priority and schedule tasks while minimizing the overall waiting time.

1) **Case-I:** *Many tasks with low priority and few or no ones with high priority*
   In order to have low-priority tasks, the very large-sized tasks ($\geq$ 5 *MB*) are generated by following the steps mentioned earlier. The waiting time of 100 such tasks is shown in Figure 9. This figure also makes it clear that all the tasks are getting a fair chance for a successful execution which in turn confirms the convergence of the proposed algorithm to optimal solutions.
2) **Case-II:** *Many tasks with high priority and few or no ones with low priority*
   As explained above, small-sized tasks ($\geq$ 10*KB* to 1 MB) are used for this case. The waiting time for 100 tasks is depicted in Figure 10. This case also yields results that are similar to those obtained in Case I.
3) **Case-III:** *Many tasks with an equal number of high and low priority tasks*
   Here, two categories of tasks are generated such as very large-sized tasks ($\geq$ 5 *MB*) and an equal number of small-sized tasks ($\geq$ 10*KB* to 1 MB). The behavior of the scheduling algorithm for such tasks is shown in Figure 11. This figure entails the fact that 50 high-priority tasks have been executed with minimal waiting time while rest 50 tasks contribute heavily towards the overall waiting time of 386.25.

C. SCHEDULING OF TASKS IN DYNAMIC CLOUD COMPUTING ENVIRONMENT

Dynamic cloud computing allocates the resources to the tasks by automatically adapting to the changes in workload. The proposed Task_scheduling algorithm is applied for task scheduling in a dynamic cloud environment. The waiting time for 500 tasks is shown in Figure 12. This figure depicts the fact that initially when there are fewer tasks, only 16 VM are considered for their execution. However, as per the increase in the number of tasks, more number of VMs are allotted to tasks to minimize the overall waiting time. Further, due to the

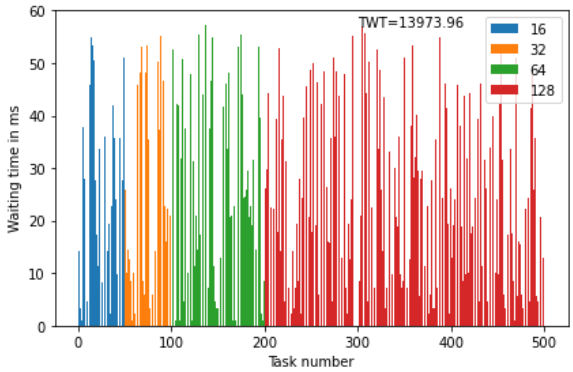

FIGURE 12. Waiting time of 500 tasks under dynamic cloud computing environment.

TABLE 6. 25 number of tasks scheduled by proposed model.

task	Size	E_T	Priority	VM	W_T
1	74396596	20.048	0.22764	8	8.825
2	46406206	12.505	0.51747	4	5.010
3	23071549	6.217	0.85513	5	0.000
4	45164575	12.171	0.55596	3	3.049
5	25207874	6.793	0.85083	6	0.000
6	32749026	8.825	0.80716	8	0.000
7	58274747	15.704	0.41348	5	6.217
8	35999672	9.701	0.76351	1	0.829
9	3077345	0.829	0.97133	1	0.000
10	61451893	16.560	0.35991	6	6.793
11	63171763	17.023	0.30681	7	6.830
12	78820207	21.240	0.15237	3	15.220
13	25347538	6.830	0.84989	7	0.000
14	79913672	21.535	0.1029	5	21.921
15	18590754	5.010	0.89070	4	0.000
16	74992186	20.208	0.20693	1	10.530
17	94523139	25.471	0.00602	1	30.739
18	90228635	24.314	0.01180	8	28.873
19	82814651	22.316	0.07657	6	23.353
20	76361013	20.577	0.19145	2	13.029
21	4752330	1.281	0.91335	2	0.000
22	11316262	3.049	0.906852	3	0.000
23	43596130	11.748	0.61825	2	1.281
24	86542245	23.321	0.02213	7	23.854
25	79160908	21.332	0.12289	4	17.515

gradual increase in the number of allotted VMs, the waiting time for the tasks is within the range of 0 to 60 which can be observed in Figure 12.

D. THE VALIDATION OF THE PROPOSED SCHEDULING ALGORITHM ON SYNTHETIC TASKS

Test case - 1

Here, 25 number of synthetically generated tasks are considered. The size of the tasks and their estimated processing time are presented in Table 6. Each task is assigned a priority value by using the algorithm. The non-preemptive scheduling algorithm is used to assign each task to the VMs. The calculated waiting time for each task is shown in Table 6.

Test case - 2

In order to validate the performance of the proposed scheduling model in handling large number of computationally less intensive tasks, a varying number of tasks

TABLE 7. Arrival of five new tasks.

task	Total size in MB	No. of VMs	W_T
1000	623.39	64 256	6826.49 165.16
2000	1188.09	64 256	26432.64 1130.26
3000	1807.12	64 256	62639.30 5665.98
4000	2376.10	64 256	110245.49 13952.67
5000	2431.83	64 256	113268.81 25566.48
6000	2996.52	64 256	174554.92 41383.55
7000	3567.14	64 256	249550.57 59982.22
8000	4218.48	64 256	346472.84 83721.33
9000	4835.95	64 256	453790.31 108409.72
10000	5447.16	64 256	581208.07 138837.57

from 1000 to 10000 is generated (Table 7). The priority of each task is calculated and then the tasks are allocated to VMs by allowing preemption among the tasks. The number of virtual machines is fixed to 64 in this case(however, it can be changed as per the system configuration). The overall waiting time ensures that the proposed scheduling model can be applied even when the number of tasks is very huge.

VI. CONCLUSION

This paper emphasizes on task scheduling for the cloud computing domain. We have introduced a priority assignment algorithm built on a new data structure known as a waiting time matrix for assigning priority to each task. The task with the highest priority is extracted from the waiting queue by adhering to the principle of the Fibonacci heap. We have proposed a parallel algorithm for task scheduling where task priority assignment and heap construction is carried out in a parallel manner concerning the preemptive and non-preemptive scheduling approaches. The efficiency of the proposed algorithms has been tested using a variety of benchmarks and synthetic data sets. The simulated results are compared with the existing techniques like BATS, IDEA, and BATS+BAR, and the comparison proves that our proposed algorithms perform better in terms of optimizing the overall waiting time as well as the CPU time consumed. Our work also exemplifies three distinct scenarios to evaluate the effectiveness of the proposed task scheduling approach while dealing with tasks of different priorities. Furthermore, a demonstration for applying a task scheduling algorithm in a dynamic cloud computing environment is also provided where the decision for virtual machine allocation is based on the number of tasks in the system.

REFERENCES

[1] D. C. Marinescu, *Cloud Computing: Theory and Practice*. San Mateo, CA, USA: Morgan Kaufmann, 2017.

[2] *Distributed and Cloud Computing, From Parallel Processing to the Internet of Things*, San Mateo, CA, USA: Morgan Kaufmann, 2012.

[3] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Workflow scheduling for SaaS/PaaS cloud providers considering two SLA levels," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Apr. 2012, pp. 906–912.

[4] D. Ergu, G. Kou, Y. Peng, Y. Shi, and Y. Shi, "The analytic hierarchy process: Task scheduling and resource allocation in cloud computing environment," *J. Supercomput.*, vol. 64, no. 3, pp. 1–14, 2013.

[5] F. Ramezani, J. Lu, J. Taheri, and F. K. Hussain, "Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments," *World Wide Web*, vol. 18, no. 6, pp. 1737–1757, 2015.

[6] A. Al-maamari and F. A. Omara, "Task scheduling using PSO algorithm in cloud computing environments," *Int. J. Grid Distrib. Comput.*, vol. 8, no. 5, pp. 245–256, Oct. 2015.

[7] A. Razaque, "Task scheduling in cloud computing," in *Proc. IEEE Long Island Syst., Appl. Technol. Conf. (LISAT)*, Apr. 2016, pp. 1–5.

[8] M. Abdullahi, M. A. Ngadi, and S. M. Abdulhamid, "Symbiotic organism search optimization based task scheduling in cloud computing environment," *Future Gener. Comput. Syst.*, vol. 56, pp. 640–650, Mar. 2016.

[9] D. Saxena, "Dynamic fair priority optimization task scheduling algorithm in cloud computing: Concepts and implementations," *Int. J. Comput. Netw. Inf. Secur.*, vol. 8, no. 2, pp. 41–48, Feb. 2016.

[10] S. M. Abdulhamid, M. S. A. Latiff, S. H. H. Madni, and M. Abdullahi, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," *Neural Comput. Appl.*, vol. 29, no. 1, pp. 279–293, Jan. 2018.

[11] E. S. Ahmad, E. I. Ahmad, E. S. Mirdha, and M. Tech, "A novel dynamic priority based job scheduling approach for cloud environment," *Int. Res. J. Eng. Technol. (IRJET)*, vol. 4, no. 6, pp. 518–522, 2017.

[12] D. Gabi, A. S. Ismail, A. Zainal, and Z. Zakaria, "Solving task scheduling problem in cloud computing environment using orthogonal taguchi-cat algorithm," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 7, no. 3, p. 1489, Jun. 2017.

[13] M. A. Alworafi, A. Dhari, A. A. Al-Hashmi, and A. B. Darem, "Cost-aware task scheduling in cloud computing environment," *Int. J. Comput. Netw. Inf. Secur.*, vol. 9, no. 5, p. 52, 2017.

[14] R. R. Patel, T. T. Desai, and S. J. Patel, "Scheduling of jobs based on Hungarian method in cloud computing," in *Proc. Int. Conf. Inventive Commun. Comput. Technol. (ICICCT)*, Mar. 2017, pp. 6–9.

[15] H. G. El Din Hassan Ali, I. A. Saroit, and A. M. Kotb, "Grouped tasks scheduling algorithm based on QoS in cloud computing network," *Egyptian Informat. J.*, vol. 18, no. 1, pp. 11–19, Mar. 2017.

[16] N. Solanki and N. C. Barwar, "Efficiency enhancing resource scheduling strategies in cloud computing," *Int. J. Eng. Techn.*, vol. 3, no. 3, pp. 149–151, 2017.

[17] R. R. Kumar, S. Mishra, and C. Kumar, "A novel framework for cloud service evaluation and selection using hybrid MCDM methods," *Arabian J. Sci. Eng.*, vol. 43, no. 12, pp. 7015–7030, Dec. 2018.

[18] N. Gobalakrishnan and C. Arun, "A new multi-objective optimal programming model for task scheduling using genetic gray wolf optimization in cloud computing," *Comput. J.*, vol. 61, no. 10, pp. 1523–1536, Oct. 2018.

[19] S. K. Panda and P. K. Jana, "Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment," *Inf. Syst. Frontiers*, vol. 20, no. 2, pp. 373–399, Apr. 2018.

[20] S. Srichandan, T. A. Kumar, and S. Bibhudatta, "Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm," *Future Comput. Inf. J.*, vol. 3, no. 2, pp. 210–230, Dec. 2018.

[21] M. B. Gawali and S. K. Shinde, "Task scheduling and resource allocation in cloud computing using a heuristic approach," *J. Cloud Comput.*, vol. 7, no. 1, pp. 1–16, Dec. 2018.

[22] A. M. S. Kumar and M. Venkatesan, "Task scheduling in a cloud computing environment using HGPSO algorithm," *Cluster Comput.*, vol. 22, no. S1, pp. 2179–2185, Jan. 2019.

[23] G. Natesan and A. Chokkalingam, "Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm," *ICT Exp.*, vol. 5, no. 2, pp. 110–114, Jun. 2019.

[24] V. Karunakaran, "A stochastic development of cloud computing based task scheduling algorithm," *J. Soft Comput. Paradigm*, vol. 2019, no. 1, pp. 41–48, Sep. 2019.

[25] I. Strumberger, M. Tuba, N. Bacanin, and E. Tuba, "Cloudlet scheduling by hybridized monarch butterfly optimization algorithm," *J. Sensor Actuator Netw.*, vol. 8, no. 3, p. 44, Aug. 2019.

[26] M. A. Elaziz, S. Xiong, K. P. N. Jayasena, and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowl.-Based Syst.*, vol. 169, pp. 39–52, Apr. 2019.

[27] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Comput. Appl.*, vol. 32, pp. 1531–1541, Mar. 2019.

[28] P. M. Rekha and M. Dakshayini, "Efficient task allocation approach using genetic algorithm for cloud environment," *Cluster Comput.*, vol. 22, no. 4, pp. 1241–1251, Dec. 2019.

[29] S. M. G. Kashikolaei, A. A. R. Hosseinabadi, B. Saemi, M. B. Shareh, A. K. Sangaiah, and G.-B. Bian, "An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm," *J. Supercomputing*, vol. 76, pp. 6302–6329, Aug. 2019.

[30] N. Bansal and A. K. Singh, "Grey wolf optimized task scheduling algorithm in cloud computing," in *Proc. Frontiers Intell. Comput., Theory Appl.*, 2020, pp. 137–145.

[31] A. Alhubaishy and A. Aljuhani, "The best-worst method for resource allocation and task scheduling in cloud computing," in *Proc. 3rd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS)*, Mar. 2020, pp. 1–6, doi: 10.1109/ICCAIS48893.2020.9096877.

[32] V. A. Lepakshi and C. S. R. Prashanth, "Efficient resource allocation with score for reliable task scheduling in cloud computing systems," in *Proc. 2nd Int. Conf. Innov. Mech. Ind. Appl. (ICIMIA)*, Mar. 2020, pp. 6–12, doi: 10.1109/ICIMIA48430.2020.9074914.

[33] C. Shetty and H. Sarojadevi, "Framework for task scheduling in cloud using machine learning techniques," in *Proc. 4th Int. Conf. Inventive Syst. Control (ICISC)*, Jan. 2020, pp. 727–731, doi: 10.1109/ICISC47916.2020.9171141.

[34] M. T. Alotaibi, M. S. Almalag, and K. Werntz, "Task scheduling in cloud computing environment using bumble bee mating algorithm," in *Proc. IEEE Global Conf. Artif. Intell. Internet Things (GCAIoT)*, Dec. 2020, pp. 01–06, doi: 10.1109/GCAIoT51063.2020.9345824.

[35] K. M. S. U. Bandaranayake, K. P. N. Jayasena, and B. T. G. S. Kumara, "An efficient task scheduling algorithm using total resource execution time aware algorithm in cloud computing," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2020, pp. 29–34, doi: 10.1109/SmartCloud49737.2020.00015.

[36] I. Attiya, M. A. Elaziz, and S. Xiong, "Job scheduling in cloud computing using a modified Harris hawks optimization and simulated annealing algorithm," *Comput. Intell. Neurosci.*, vol. 2020, pp. 1–17, Mar. 2020, doi: 10.1155/2020/3504642.

[37] J. Yang, B. Jiang, Z. Lv, and K.-K.-R. Choo, "A task scheduling algorithm considering game theory designed for energy management in cloud computing," *Future Gener. Comput. Syst.*, vol. 105, pp. 985–992, Apr. 2020.

[38] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Syst. J.*, vol. 14, no. 3, pp. 3117–3128, Sep. 2020, doi: 10.1109/JSYST.2019.2960088.

[39] M. S. Arif, Z. Iqbal, R. Tariq, F. Aadil, and M. Awais, "Parental prioritization-based task scheduling in heterogeneous systems," *Arabian J. Sci. Eng.*, vol. 44, no. 4, pp. 3943–3952, Apr. 2019, doi: 10.1007/s13369-018-03698-2.

[40] S. Kanwal, Z. Iqbal, F. Al-Turjman, A. Irtaza, and M. A. Khan, "Multiphase fault tolerance genetic algorithm for vm and task scheduling in datacenter," *Inf. Process. Manage.*, vol. 58, no. 5, Sep. 2021, Art. no. 102676, doi: 10.1016/j.ipm.2021.102676.

[41] M. S. Ajmal, Z. Iqbal, F. Z. Khan, M. Ahmad, I. Ahmad, and B. B. Gupta, "Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers," *Comput. Electr. Eng.*, vol. 95, Oct. 2021, Art. no. 107419, doi: 10.1016/j.compeleceng.2021.107419.

[42] M. Naghibzadeh, *Contemporary Operating Systems*, 2023.

[43] H. Goudarzi and M. Pedram, "Achieving energy efficiency in datacenters by virtual machine sizing, replication, and placement," in *Advances in Computers*, vol. 100, 2016, pp. 161–200.

[44] I. Adan and J. Resing, *Queueing Theory*. Eindhoven, The Netherlands: Eindhoven Univ. Technology, 2002.

[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2022.

[46] M. Naghibzadeh, "New generation computer algorithms," Tech. Rep., 2023.

[47] J. P. Hayes, *Computer Architecture and Organization*. New York, NY, USA: McGraw-Hill, 2007.
[48] S. Singhal and A. Sharma, "Resource scheduling algorithms in cloud computing: A big picture," in *Proc. 5th Int. Conf. Inf. Syst. Comput. Netw. (ISCON)*, Oct. 2021, pp. 1–6, doi: 10.1109/ISCON52037.2021.9702313.
[49] Y.-J. Chiang and Y.-C. Ouyang, "Profit optimization in SLA-aware cloud services with a finite capacity queuing model," *Math. Problems Eng.*, vol. 2014, pp. 1–11, Jan. 2014.
[50] Center SC. (2014). *Cybershake and Epigenomics Scientific Workflow*. Accessed: Jan. 1, 2016. [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

SWATI LIPSA received the B.Tech. and M.Tech. degrees from the Biju Patnaik University of Technology, Rourkela, Odisha, India, in 2008 and 2014, respectively. She is currently an Assistant Professor with the Department of Information Technology, Odisha University of Technology and Research, Bhubaneswar, Odisha. She has a good number of publications in different international journals/conferences. Her research interests include wireless sensor networks, cloud computing, network security, machine learning, and software-defined networking.

RANJAN KUMAR DASH (Senior Member, IEEE) received the Ph.D. degree from Sambalpur University, Sambalpur, Odisha, India, in 2008. He is currently a Professor and the Head of the Department of Information Technology, Odisha University of Technology and Research, Bhubaneswar, Odisha. He has more than 42 publications in different international journals/conferences. His primary interests include the reliability of distributed systems, wireless sensor networks, soft computing, machine learning, and cloud computing. His current research focus is on enhancing processor performance and energy consumption through the use of machine-learning techniques. He has served on the technical program and organization committees for several conferences.

NIKOLA IVKOVIĆ was born in Zagreb, Croatia, in 1979. He received the M.S. degree in computing and the Ph.D. degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb. His Ph.D. thesis was in the area of the swarm and evolutionary computation.

He was the Head of the Department of Computing and Technology, Faculty of Organization and Informatics, University of Zagreb, where he is currently an Assistant Professor. He is a member of two research laboratories, such as the Artificial Intelligence Laboratory, Faculty of Organization and Informatics, University of Zagreb, and the Laboratory for Generative Programming and Machine Learning, Faculty of Organization and Informatics, University of Zagreb. He teaches computer networks, operating systems, and computer architecture-related courses. He gave several invited talks at international scientific conferences and guest lectures at different universities in Europe and Asia. His research interests include computational intelligence and optimization, especially swarm intelligence, and also computer networks, security, and formal methods. He was a member of committees for creating new university study programs. He serves as a regular reviewer for high-quality scientific journals and takes part in a number of international conference committees. He won the Best Presentation Award from the International Conference on Computer Technology and Development (IACT 2015), Singapore, and the International Conference on Frontiers of Intelligent Technology (ICFIT 2018), Paris, and the Excellent Presentation Award from the International Conference on Computer Science and Information Technology (ICCSIT 2016), Ireland. He joined the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE), in 2008.

KORHAN CENGIZ (Senior Member, IEEE) was born in Edirne, Turkey, in 1986. He received the B.Sc. degree in electronics and communication engineering from Kocaeli University, in 2008, the B.Sc. degree in business administration from Anadolu University, Turkey, in 2009, the M.Sc. degree in electronics and communication engineering from Namik Kemal University, Turkey, in 2011, and the Ph.D. degree in electronics engineering from Kadir Has University, Turkey, in 2016. Since August 2021, he has been an Assistant Professor with the College of Information Technology, University of Fujairah, United Arab Emirates. Since April 2022, he has been the Chair of the Research Committee of the University of Fujairah. Since September 2022, he has been an Associate Professor with the Department of Computer Engineering, Istinye University, Istanbul, Turkey. He is the author of more than 40 SCI/SCI-E articles, including IEEE Internet of Things Journal, IEEE Access, *Expert Systems with Applications*, *Knowledge-Based Systems*, and *ACM Transactions on Sensor Networks*, five international patents, more than ten book chapters, and one book in Turkish. He is an editor of more than 20 books. His research interests include wireless sensor networks, wireless communications, statistical signal processing, indoor positioning systems, the Internet of Things, power electronics, and 5G. He is a Professional Member of ACM. He received several awards and honors, such as the Tubitak Priority Areas Ph.D. Scholarship, the Kadir Has University Ph.D. Student Scholarship, the Best Presentation Award at the ICAT 2016 Conference, and the Best Paper Award at the ICAT 2018 Conference. He is an Associate Editor of IEEE Transactions on Intelligent Transportation Systems, *IEEE Potentials*, *IET Electronics Letters*, and *IET Networks*. He is the Handling Editor of *Microprocessors and Microsystems* (Elsevier). He serves as a Reviewer for IEEE Internet of Things Journal, IEEE Sensors Journal, and IEEE Access. He serves several book editor positions in IEEE, Springer, Elsevier, Wiley, and CRC. He presented more than 40 keynote talks at reputed IEEE and Springer conferences about WSNs, the IoT, and 5G.