Cluster Computing (2024) 27:13407–13419
https://doi.org/10.1007/s10586-024-04594-1

Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems

Mostafa Raeisi-Varzaneh[1] · Omar Dakkak[1] · Yousef Fazea[2] · Mohammed Golam Kaosar[3]

Received: 20 February 2024 / Revised: 18 May 2024 / Accepted: 23 May 2024 / Published online: 27 June 2024
© The Author(s) 2024

Abstract

Cloud computing has emerged as an efficient distribution platform in modern distributed computing offering scalability and flexibility. Task scheduling is considered as one of the main crucial aspects of cloud computing. The primary purpose of the task scheduling mechanism is to reduce the cost and makespan and determine which virtual machine (VM) needs to be selected to execute the task. It is widely acknowledged as a nondeterministic polynomial-time complete problem, necessitating the development of an efficient solution. This paper presents an innovative approach to task scheduling and allocation within cloud computing systems. Our focus lies on improving both the efficiency and cost-effectiveness of task execution, with a specific emphasis on optimizing makespan and resource utilization. This is achieved through the introduction of an Advanced Max–Min Algorithm, which builds upon traditional methodologies to significantly enhance performance metrics such as makespan, waiting time, and resource utilization. The selection of the Max–Min algorithm is rooted in its ability to strike a balance between task execution time and resource utilization, making it a suitable candidate for addressing the challenges of cloud task scheduling. Furthermore, a key contribution of this work is the integration of a cost-aware algorithm into the scheduling framework. This algorithm enables the effective management of task execution costs, ensuring alignment with user requirements while operating within the constraints of cloud service providers. The proposed method adjusts task allocation based on cost considerations dynamically. Additionally, the presented approach enhances the overall economic efficiency of cloud computing deployments. The findings demonstrate that the proposed Advanced Max–Min Algorithm outperforms the traditional Max–Min, Min–Min, and SJF algorithms with respect to makespan, waiting time, and resource utilization.

Keywords Advanced Max–Min algorithm · Task scheduling · Cloud computing · Task allocation · Makespan · Cost aware

✉ Omar Dakkak
omardakkak@karabuk.edu.tr

Mostafa Raeisi-Varzaneh
mostafaraeisi1991@gmail.com

Yousef Fazea
fazeaalnades@marshall.edu

Mohammed Golam Kaosar
Mohammed.Kaosar@murdoch.edu.au

[1] Department of Computer Engineering, Karabük University, 78050 Karabük, Türkiye

[2] Department of Computer and Information Technology, Marshall University, 1 John Marshall Drive, Huntington, WV 25755, USA

[3] School of Information Technology, College of Science, Technology, Engineering and Mathematics, Murdoch University, Perth 6150, Australia

1 Introduction

Cloud computing has rapidly gained prominence in contemporary corporate environments, driven by the widespread expansion of Internet connectivity and the proliferation of big data [1–4]. Defined as a collection of interconnected computers offering diverse computing resources, the cloud presents a novel paradigm for computing, storage, and analytics services, distinguished by its high agility, availability, scalability, and flexibility over the Internet [5–8]. This technology has garnered significant interest from individuals, industries, and academia [9, 10]. While sharing characteristics with grid computing, distributed computing, and utility computing, cloud computing sets itself apart by leveraging virtualization to efficiently manage resources [11, 12]. Offering a flexible and scalable service delivery model compared to traditional distributed computing systems, thanks to its pay-per-use policy, the cloud eliminates the need for users to invest in infrastructure, computational resources, and platforms [13]. Users may access a range of services through the cloud, including Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Expert as a Service (EaaS), and Software as a Service (SaaS) [14].

The cloud infrastructure comprises geographically distributed data centers, each housing thousands of servers subdivided into multiple virtual machines, each equipped with memory, CPU, and storage components [15]. Task scheduling has emerged as a significant challenge in cloud computing, requiring active scanning and decision-making to determine the appropriate VM for each task [16, 17]. In other words, scheduling is the process of allocating jobs or tasks to the VMs for execution. In the cloud computing context, efficient task scheduling is essential for optimum resource utilization. Efficient task scheduling is paramount for optimizing resource utilization, preserving VM makespan, and ensuring adherence to Service Level Agreement (SLA) criteria for Quality of Service (QoS) [10]. However, task scheduling is inherently a computationally complex problem, categorized as NP-hard with a runtime complexity of $O(mn)$ for scheduling n tasks on m virtual machines [18, 19]. As the size of tasks and the number of VMs increase, the computation time for discovering the optimal (task, VM) pair increases exponentially [20]. Without efficient algorithms that can identify the optimal solution in polynomial runtime, task scheduling consumes an expanding solution space [21]. Consequently, an appropriate mapping algorithm between client tasks and resources is imperative to simultaneously reduce makespan and waiting time while enhancing resource utilization [22].

This paper proposes an innovative approach to address these challenges through the utilization of an Advanced Max–Min algorithm. By employing this algorithm, we aim to enhance the efficiency and performance of task scheduling compared to conventional methods. Experimental findings demonstrate the superiority of our approach, showcasing significant improvements in makespan, and resource utilization over the traditional Max–Min algorithm, Min–Min and SJF. Furthermore, our research extends beyond mere optimization of scheduling parameters, as we introduce a novel cost-aware Max–Min algorithm tailored to meet the requirements of both users and cloud providers. This algorithm offers a mechanism for regulating users' expenditures while ensuring optimal task execution, providing flexibility and efficiency in cloud resource management. A key contribution of our work lies in the dynamic adaptation of task scheduling, facilitated by the integration of dynamic variables to determine task execution and waiting times in real-time. This online approach enhances the responsiveness and adaptability of the scheduling system, enabling it to effectively handle varying workloads and resource availability. The rest of the paper is organized as follows: Sect. 2 establishes the literature review and related works. Section 3 provides the description of the problem. Section 4 presents the proposed algorithm. Section 5 evaluates the experimental results for the model and analysis the performance with respect to existing previous works, and finally Sect. 6 concludes this paper.

2 Related works

Task scheduling in cloud computing has been extensively studied due to the increasing demand for effective resource allocation in cloud environments. Various research efforts have focused on addressing this challenge and optimizing key performance metrics such as makespan, response time, VM utilization, and cost [23].

Cui et al. [24] Analysed cloud service reliability using a Markov-based approach, formulating cloud scheduling as a multi-objective optimization problem. They proposed a multi-objective model for cloud task scheduling, incorporating queuing theory and Markov process, and presents a genetic based algorithm, which demonstrated effective reduction in cost, execution time, and makespan. Similarly, Liu et al. [25] proposed a workflow scheduling framework based on genetic algorithm, emphasizing cost criteria optimization, outperforms state-of-the-art algorithms like Particle Swarm Optimization (PSO) in meeting deadlines and reducing total execution costs. A cost-aware task scheduling mechanism in the cloud environment is proposed by Zhang and Zhou [26], dynamically assigning virtual machines to tasks to minimize completion time. The CloudSim toolbox simulation proved that this mechanism

achieved low cost and makespan. Sreenu and Sreelatha [14] introduces W-Scheduler, a task scheduling algorithm for cloud environments utilizing a multi-objective model and the Whale Optimization Algorithm (WOA). W-Scheduler optimally allocates tasks to virtual machines, minimizing both makespan and cost.

To further improve the optimal search capability of the WO-based method solution, Chen et al. [27], introduced an Improved Whale Optimization Algorithm for cloud task scheduling (IWC). Simulations indicate that the proposed IWC outperforms existing meta-heuristic algorithms in terms of utilization, convergence time and accuracy. Inspired by crow's food- gathering tendencies, Kumar, and Kousalya [28] proposed Crow Search Algorithm (CSA) for task scheduling. The crow keeps an eye on its other mates to find a better food source than the current one. In this way, CSA finds an appropriate VM for the tasks to minimize the makespan. The simulation results prove that the CSA algorithm outperforms the Min–Min and Ant algorithms. Huang et al. [20], proposed an algorithm with multiple discrete variants of the PSO algorithm for task scheduling in cloud computing. To evaluate the performance, these approaches were compared with three well-known heuristic algorithms. Experiment results demonstrate that the average makespan of the proposed PSO-based scheduler is effectively reduced compared to the gravitational search algorithm, artificial bee colony algorithm, and dragonfly algorithm.

Kashikolaei et al. [29], proposed an intelligent meta-heuristic algorithm based on the imperialist competitive algorithm (ICA) and firefly algorithm (FA), showing significant improvements in makespan, load balancing, CPU time, stability, and planning speed.

Singh et al., [30] proposed an algorithm that uses the honeybee load balancing and improvement detection operator to conclude which low-level heuristic is utilized to search for improved candidate solutions. The consequences of the proposed task scheduling algorithm are matched with existing heuristic-based scheduling procedures. The CloudSim toolkit simulation results demonstrated efficiency gains compared to existing heuristic-based scheduling approaches. Sreenivasulu and Paramasivam [18] developed a hybrid optimization model to allocate tasks to virtual machines in cloud computing efficiently. A hierarchical method is used to manage task priorities. The model's performance is compared with existing bandwidth-aware algorithms (BAT) and ant colony optimization (ACO). The hybrid model has shown to be efficient in terms of resource utilization, bandwidth utilization, memory usage, and response time.

Dakkak et al. [31], presented the Swift Gap (SG) mechanism for task scheduling, combining backfilling and metaheuristic local search optimization, with promising results in reducing latency and improving performance. The task is placed in the earliest available gap in the schedules of the local resources in the first step, and performance is optimized in the second stage by looking through all the gaps in the schedules of the resources to find a better gap to place the job in.

Max–Min and Min–Min stand out as widely recognized scheduling algorithms helping cloud service providers to address diverse user requirements. Nevertheless, their application often leads to challenges such as increased waiting time, resource starvation, and prolonged completion times. Consequently, numerous Advanced methodologies have emerged, building upon these foundational algorithms to mitigate these shortcomings and enhance overall efficiency in cloud task scheduling. Karuppan et al., in [32] proposed a Priority-Based Max–Min(PBMM) scheduling algorithm, leveraging fuzzy inference system to obtain a lower makespan while maximizing throughput. The CloudSim simulation results show that the PBMM reduces the time for scheduling tasks compared to other algorithms such as Shortest Job First (SJF) and Round Robin (RR). Amalarethinam & Kavitha in [33] proposed a rescheduling enhanced Min–Min algorithm (REMM). The research results of the REMM show evidence for its better performance than Min–Min and Balancing Min–Min (LBMM) in execution time and resource utilization.

Arif [22] proposed the hybrid Min–Min and RR scheduling algorithm (HMMRR), achieving improved resource utilization and system performance. The proposed algorithm is compared to other existing algorithms such as RR, Min–Min, and Max–Min, with experimental findings demonstrating that the HMMRR algorithm outperforms others. Ming and Li [34] introduced an Advanced Max–Min algorithm that, instead of picking the fastest available VM to perform tasks, chooses the one with the least waiting time. This algorithm reduces the application's makespan as well as the average waiting time for tasks. The absence of dependent task management is one of the most challenging issues with the proposed algorithm. The traditional Max–Min algorithm, which motivates this article, also does not have a policy for dealing with related tasks. To mitigate high waiting time and starvation of small tasks in Max–Min algorithm, Pradhan et al., [35] proposed an Improved Max–Min (IMM) algorithm which optimizes resource allocation, reduces waiting and completion times, and maximizes overall resource utilization. IMM prioritizes tasks based on execution time, preventing starvation of low size and ultimately outperforming MM in efficiency, as evidenced by experimental results.

The above-mentioned scheduling algorithms cannot distinguish dependent and independent tasks. Brar & Rao [36] implemented the Max–Min algorithm to schedule workflow tasks, prioritizing dependent tasks while

processing independent tasks concurrently to minimize computation time. The algorithm efficiently schedules multiple jobs on multiple machines, demonstrating improved execution time compared to existing methods.

Table 1 summarized the related works discussed in this section providing valuable information regarding their methods, evaluation metrics and simulation platforms. In summary, this paper aims to contribute to the body of knowledge on cloud task scheduling by proposing a simple, yet effective Max–Min based algorithm to manage related tasks and improve various quality features of cloud networks, including makespan, response time, VM utilization, and cost.

3 Problem description

In this study, we model applications as Directed Acyclic Graphs (DAGs) $= \{V, E\}$, where nodes represent individual application tasks labeled by their number of instructions (Million instructions) and edges denote task dependencies. The computational resources, represented as VM $= \{VM_1, VM_2, \ldots, VM_k\}$, are characterized by their processing power in MIPS (million instructions per second) and associated costs.

The Expected Execution Time (ETC) of each task on a VM signifies the time required for its completion, while waiting time denotes the duration a task remains in the queue for resources. Makespan, indicating the total application runtime, can be computed across all VMs:

$$MS = \max(MS(VM_1), MS(VM_2), \ldots, MS(VM_k)) \tag{1}$$

Resource utilization, a crucial metric, is calculated as the ratio of total makespan across all VMs to the overall makespan, normalized by the number of VMs:

$$\text{Resource Utilization} = \frac{\sum_{i=1}^{k} MS(VM_i)}{MS} \times \frac{1}{k} \tag{2}$$

Additionally, we introduce idleness cost to quantify the system's cost incurred when VMs remain idle, calculated by the difference between the total makespan and individual VM makespan, multiplied by their respective costs:

$$\text{Idealness Cost} = \sum_{i=1}^{k} (MS - MS(VM_i)) \times \text{Cost}(VM_i) \tag{3}$$

In this paper, we introduce an Advanced Max–Min Algorithm with the primary goal of reducing makespan (Eq. 1), enhancing system utilization (Eq. 2), and mitigating idleness cost (Eq. 3). By optimizing task scheduling within cloud computing environments, our algorithm aims to improve overall system efficiency and resource utilization.

Table 1 Comparison of scheduling methods in cloud computing

References	Algorithm	Evaluation factors					Analysis environment
		Time	Cost	Reliability	Makespan	Utilization	
[24]	Meta-heuristic	✔	✔	✔	✔	✗	Simulation (MATLAB)
[25]	Meta-heuristic	✔	✔	✗	✔	✗	Simulation (WorkflowSim)
[26]	Non-heuristic	✔	✔	✗	✔	✔	Simulation (CloudSim)
[14]	Meta-heuristic	✗	✔	✗	✔	✗	CloudSim/Java
[27]	Meta-heuristic	✔	✔	✗	✗	✔	Implementation (MATLAB)
[28]	Heuristic	✔	✗	✗	✔	✗	Simulation (CloudSim)
[20]	Hybrid	✗	✗	✗	✔	✗	Implementation (MATLAB)
[29]	Hybrid	✔	✗	✗	✔	✔	Implementation (DotNet)
[30]	Hyper meta-heuristic	✔	✗	✗	✔	✔	Simulation (CloudSim)
[18]	Hybrid	✔	✗	✗	✗	✔	Simulation (CloudSim)
[32]	Heuristic	✔	✗	✗	✗	✔	Simulation (CloudSim)
[33]	Heuristic	✗	✗	✗	✔	✔	(N/A)
[22]	Hybrid	✔	✗	✗	✔	✔	Implementation Boreland C++
[34]	Hybrid	✔	✔	✗	✔	✔	Simulation (CloudSim)
[35]	Heuristic	✔	✗	✗	✔	✔	(N/A)
[36]	Heuristic	✔	✗	✗	✔	✗	(N/A)
Our work	Heuristic	✔	✔	✗	✔	✔	Python implementation

4 Modern Max–Min algorithms

Task scheduling in cloud computing requires efficient algorithms to optimize key performance metrics such as makespan and cost. While traditional scheduling algorithms like First Come First Served (FCFS), Round-Robin (RR), Shortest Job First (SJF), Max–Min, and Min–Min provide solutions, their performance may fall short of meeting the evolving needs of cloud computing [31]. The Max–Min algorithm, a well-known scheduling algorithm, offers benefits such as ease of implementation and has seen various extensions. However, despite efforts to enhance its quality features limitations persist, particularly in handling related jobs and achieving the shortest makespan in all cases. Particularly, although [36], improves the classical algorithm's quality features by adding the possibility of handling related tasks, it eliminates the possibility of achieving the shortest makespan by executing tasks in sequence. Section 4.1 provides an overview of the algorithm proposed by Brar et al. [36], detailing the novel methodologies and strategies employed to achieve related tasks scheduling. Additionally, we illustrate how this approach inadvertently results in the sequential execution of tasks, leading to poor system performance.

4.1 Workflow Max–Min algorithm

The Max–Min task scheduling algorithm is widely used for minimizing makespan due to its simplicity. However, its conventional implementation lacks the capability to handle related tasks effectively. Brar [36] aimed to enhance the algorithm's performance by introducing the concept of using associated tasks, leading to the development of the Workflow Max–Min algorithm. Algorithm 1 illustrates the stages of the Workflow Max–Min algorithm, which addresses the challenge of scheduling related tasks in cloud computing environments. Each application is represented by a DAG, where vertices represent tasks, and edges signify dependencies between tasks [21]. The presence of an edge from node a to node b indicates that task a must execute before task b as shown in Fig. 1. Also, the number of instructions for each task is stored in the corresponding node. It is important to extract the transposition of tasks for each application's DAG to apply Max–Min algorithm. The parent(x) function extracts the tasks that must be executed before x, whereas the children(x) function finds the tasks that must execute after x. For example, in the application in Fig. 1, children$(a) = b.c.d.e$ and parent$(d) = a.b.c$.

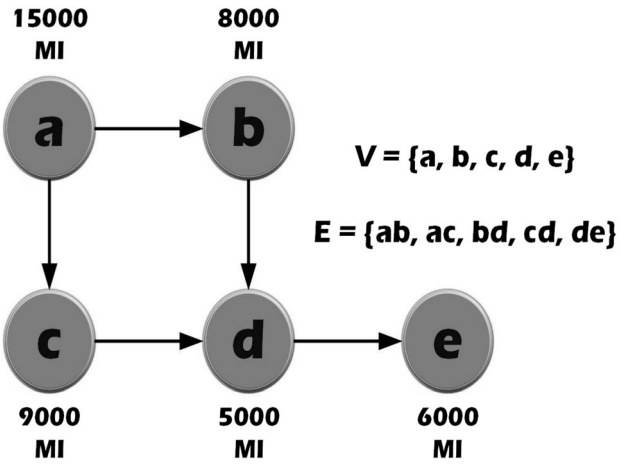


Fig. 1 A sample application

Algorithm 1 Max–Min algorithm proposed by [36]

Inputs: application Directed Acyclic Graph (DAG) and VM's characteristics
Outputs: tasks execution order, makespan, average response time, core utility, idleness cost
Start
Extract tasks dependencies by Child(x) and Parent(x) functions from DAG
while there is an unexecuted task **do**
    Schedule tasks
end while
while all virtual machines are busy **do**
    Wait until one VM gets idle
    if there is an available VM **then**
        Fetch a task with maximum execution time that has no parent
        Execute it on the virtual machine that completes the task faster
        Remove the task's node and all its associated edges from DAG
    end if
end while

The algorithm starts to work if there is an idle VM in the system. In this step, a task with no parent and the highest number of instructions is selected. The task is executed by an idle VM with the shortest completion time. The task's name is then removed from the parent list of children in the next phase. The loop will continue until all tasks have been accomplished. Running the job with the most instructions by the fastest virtual machine (i.e., shortest completion time) reduces makespan, balances the workload, and increases the VM's utility. However, a limitation of the Workflow Max–Min algorithm becomes apparent in certain scenarios where applications are executed sequentially, contrary to the goal of distributed systems to run applications concurrently. Figure 2 shows an example that will help to project the issue.

Running the job with the most instructions by the fastest virtual machine (i.e., shortest completion time) reduces makespan, balances the workload, and increases the VM's utility. However, a limitation of the Max–Min Algorithm propose by [36] becomes apparent in certain scenarios where applications are executed sequentially, contrary to the goal of distributed systems to run applications concurrently. Figure 2 shows an example that will help to project the issue.

The scheduling order will be determined by executing the algorithm on two applications using three virtual machines with the processing power of 5, 3, and 2 thousand million instructions per second. After the first application tasks (*a*, *b*, *c*, *d*, *e*), second application tasks (*f*.*g*.*h*.*i*) are executed, as shown in Fig. 3. Despite the intention to run applications concurrently, the Max–Min Algorithm leads to the sequential execution of applications. This sequential execution violates the fundamental principle of distributed systems and highlights the need for enhancements to the scheduling algorithm to ensure concurrent execution of applications.

A heuristic algorithm based on the Max–Min can be employed to overcome this limitation. This approach would enable concurrent execution while retaining the benefits of Max–Min, such as, minimizing makespan, and balancing the workload across VMs.

4.2 Proposed advanced cost aware Max–Min algorithm

The previous section found that the number of instructions is not a useful criterion for selecting tasks to execute. This subsection introduces a novel approach to address these limitations by gradually evolving the Max–Min algorithm. Our primary contribution lies in the development of a strategy for managing related tasks, thereby improving scheduling quality features. Furthermore, we introduce a cost function to enhance the quality of services, offering a comprehensive solution to optimize makespan, response time, resource utilization, and cost simultaneously.

To complete applications in parallel and reduce makespan, it is essential to consider the number of instructions for each child and the number of instructions for each task. Therefore, in the Advanced Cost-Aware Max–Min Algorithm, a task with the maximum number of instructions and the maximum number of children are selected. All steps of the proposed algorithm have shown in Algorithm 2. The procedure of the Advanced Cost-Aware Max–Min Algorithm is quite like the Workflow algorithm, except that the task selection criteria has changed in the fetching stage. The result of implementing this algorithm on the tasks of Fig. 2 illustrated in Fig. 4.

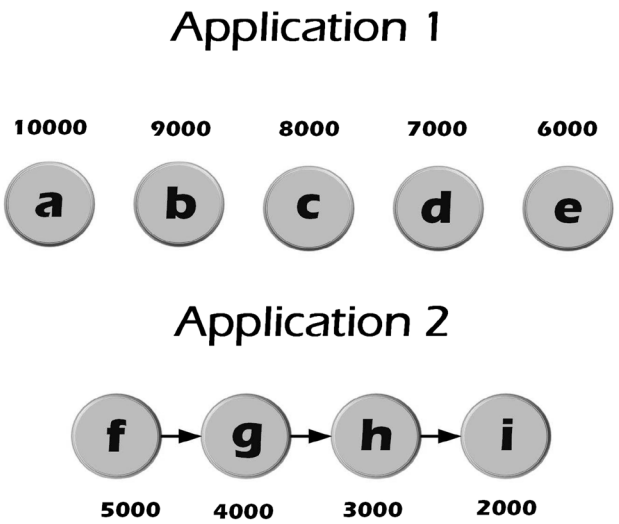


Fig. 2 Two sample applications demonstrating sequential scheduling with Workflow Max–Min algorithm

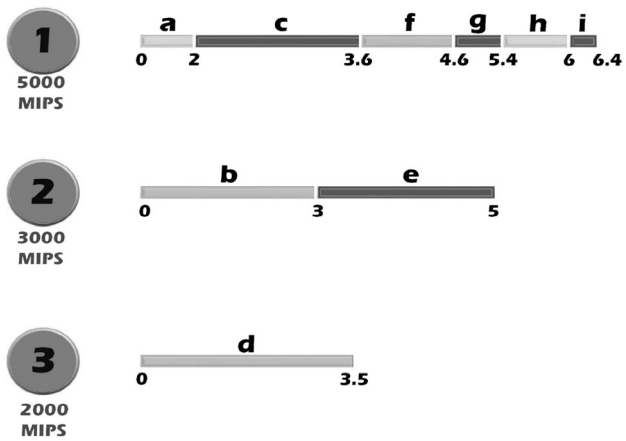


Fig. 3 Workflow Max–Min limitation—sequential execution of two sample applications of Fig. 2

Algorithm 2 Advanced Max–Min working steps

Inputs: Application Directed Acyclic Graph (DAG) and VM's characteristics
Outputs: Tasks execution order, makespan, average response time, core utility, idleness cost
Start
Extract tasks dependencies by Child(x) and Parent(x) functions from DAG
while there is an un-executed task **do**
    Schedule tasks
end while
while all virtual machines are busy **do**
    Wait until one VM gets idle
    if there is an available VM **then**
        Fetch a task with maximum "execution time + child's execution time" that has no parent
        if Cost Management Option = 0 **then**
            Execute it on the virtual machine that completes the task faster
        end if
    else

        Execute it on the virtual machine that minimizes ($\alpha \times$ Normalized Completion time + $(1-\alpha) \times$ Normalized cost)
        Remove the task's node and all its associated edges from DAG
    end if
end while

Although the algorithm demonstrated its capacity to reduce the makespan (6.4 to 5.9), it only focuses on running time. The algorithm disregards the cost required for executing each application on the VMs. Therefore, a specific cost is considered for executing each task on the VMs in the system. The cost for executing every 1000 million instructions is assumed as 15, 10 and five units by VMs 1,2 and 3, respectively. The cost aware part of the algorithm would be activated when the cost management is set to 1. Algorithm works like the previous part in terms of task selection. The difference is that a VM that provides the shortest completion time does not necessarily execute the selected task. Instead, a combination of time and cost is used to find a suitable VM. To combine time and cost with different units and intervals efficiently, normalization must be conducted. After selecting the appropriate task, the completion time and cost required for running a task on each VM are calculated. Finally, the values obtained are normalized (Minimum Maximum Normalization), and a VM with the lowest α Normalized Completion time + $(1-\alpha)$ Normalized cost is selected to run the application.

The α factor, defined between 0 and 1, decides whether a higher-speed or a lower-cost VM executes the task. This factor can be adjusted based on the characteristics and system requirements.

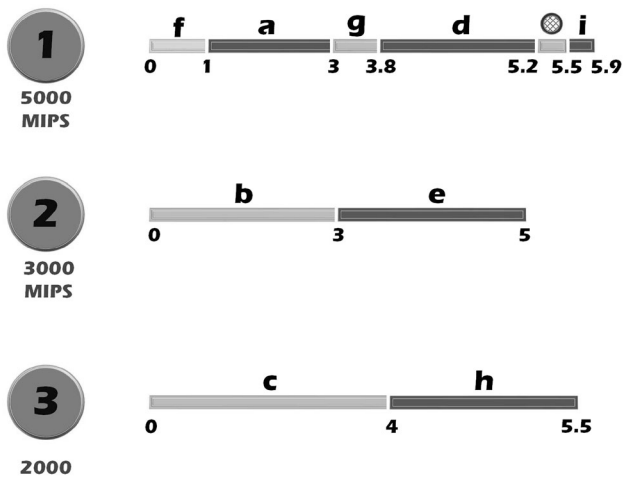


Fig. 4 Concurrent task execution achieved by the Advanced Max–Min Algorithm on the applications of Fig. 2

5 Experimental results

The proposed methods in the previous section have been implemented in a cloud environment with 3 VMs that execute 5, 3, and 2 thousand million instructions per second by the multi-threading feature in Python environment. The algorithm runs on five sets of random tasks shown in Fig. 5. Each of these sets includes four applications, and each application contains three tasks. The settings of all the parameters are presented in Table 2. The Advanced algorithm, Workflow Max–Min, Min–Min and SJF are

implemented on sets 1 to 5, and the results are reported in Figs. 6, 7, 8 and 9.

Figure 6 indicates that employing the Advanced Max–Min algorithm leads to a decrease in makespan in almost all cases compared to other approaches. In our algorithm, identifying the node with the highest cumulative weight of outgoing edges in the application graph is crucial. This cumulative weight captures the outgoing connections and the associated strength (weight). Such a node acts as a prominent source of outward influence, potentially creating bottlenecks in the network. By prioritizing this node and strategically assigning it to resources that minimize completion time, we have potentially achieved significant reductions in makespan.

Furthermore, as shown in Fig. 7, the Advanced algorithm effectively decreases the average waiting time for each task to receive suitable VM services. Specifically, the average waiting time of the 5 five random task sets has been reduced from 5.27 s per task in the Workflow Max–Min algorithm to 4.67 s in the Advanced algorithm. Although SJF and Min–Min algorithms yield lower waiting times, their reliance on slower VMs results in higher makespan.

In terms of resource utilization, Fig. 8 illustrates potential enhancement in core utility of up to 34% with the Advanced algorithm as opposed to other methods. This significant improvement stems from the algorithm's ability to prioritize tasks effectively. By focusing on tasks with the highest cumulative weight of outgoing edges, the Advanced algorithm identifies those that have the most significant impact in the application graph. Assigning these high-impact tasks strategically to VMs that offer the lowest completion time minimizes idle time within the system. This approach ensures that critical tasks are processed efficiently, leading to a more balanced workload distribution and ultimately, a significant improvement in overall core utilization.

Moreover, suppose that VM 1, 2, and 3 cost the system 15, 10, and 5 units, respectively; therefore, the total cost imposed on the system because of VM idleness is reduced from 122 units in the SJF algorithm to 10 units for the task set 3 in the Advanced Max–Min algorithm (as shown in Fig. 9), which translates to a substantial reduction in idleness costs of up to 92%.

This considerable cost reduction directly results from the algorithm's ability to minimize idle time, particularly for high-performance VMs. By prioritizing tasks with the highest cumulative weight of outgoing edges and strategically assigning them to the VMs that offer the lowest completion time, the Advanced Max–Min algorithm ensures that expensive, faster VMs are utilized efficiently. This approach reduces the instances where these high-cost VMs remain idle, leading to significant cost savings.

A Cost-Aware Advanced Max–Min algorithm has been presented in the previous section, which manages the cost of task execution under the supervision of cloud providers based on the client's requirements. The implementation of this algorithm on task set 1 shown in Fig. 5 with different α factors is presented in Table 3. The results demonstrate that by setting the α factor to 1, the algorithm performs similarly to the Advanced algorithm, and tasks execute on VMs, which provide the least completion time. As the α factor approaches zero, the tasks are sent to a VM with a lower cost to run. While setting the α value is set to zero, all tasks are assigned to the third VM, which has the lowest expenditure.

While our algorithm is derived from Max–Min, since it is not necessary to explore all the tasks/VMs combinations at the first step of our algorithm, the proposed Advanced Max–Min algorithm follows the time complexity of $O(mn^2)$ like the original Max–Min, where m is the number of resources and n is the number of tasks. While exhibiting a similar level of complexity to traditional Max–Min algorithms, the proposed mechanism yields superior makespan results along with a more dependable scheduling scheme.

6 Conclusion

In conclusion, this paper has presented a novel approach to addressing the intricate challenges of task scheduling within cloud computing environments. By introducing an Advanced Max–Min algorithm alongside a cost-aware scheduling framework, our methodology not only enhances traditional metrics such as makespan and resource utilization but also effectively manages task execution costs. The experimental results indicated a notable improvement over conventional Max–Min, Min–Min, and SJF algorithms, highlighting the efficacy of our proposed solution in optimizing cloud task scheduling. Moreover, our approach offers a dynamic and flexible mechanism for task allocation, ensuring alignment with user requirements and cloud service provider constraints. Overall, this research contributes significantly to advancing the field of cloud task scheduling, offering a promising avenue for improving efficiency, performance, and economic viability in cloud computing deployments. In future research, it would be beneficial to analyze the proposed algorithms on different benchmarks to test the algorithm's dynamic behavior.

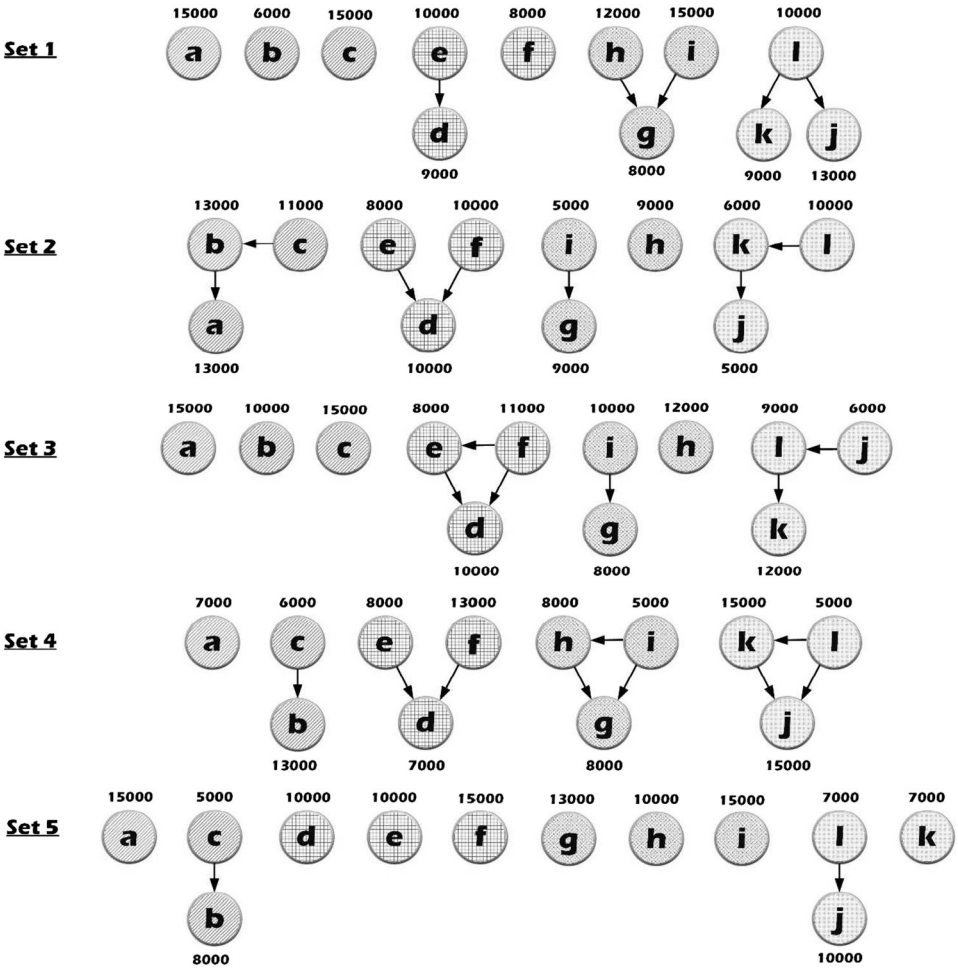


Fig. 5 Five random sets of tasks

Table 2 Setup of experiments

Number of VMs	3
Number of applications	5
Number of tasks	12 in each application
Tasks types	Dependent
Performance metrics	Makespan, waiting time, core utility, and idleness cost

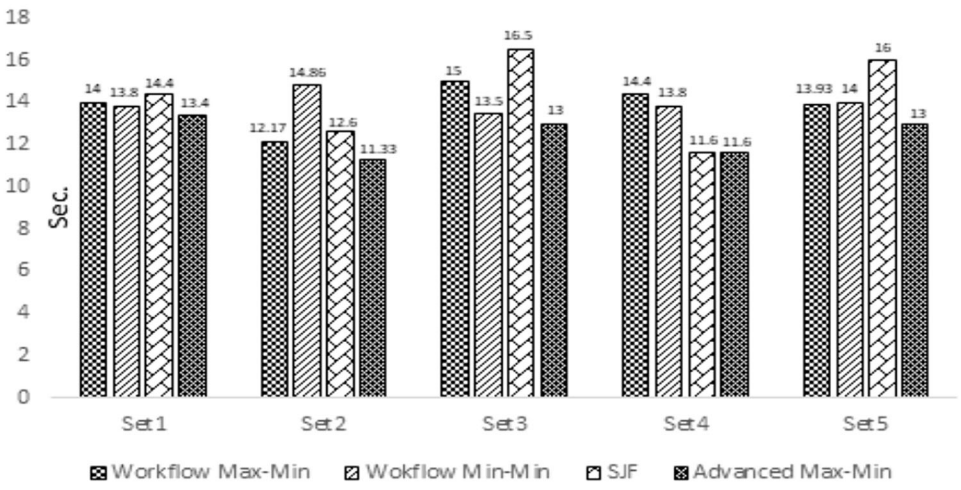


Fig. 6 Comparative analysis of makespan among task scheduling algorithms on random task sets

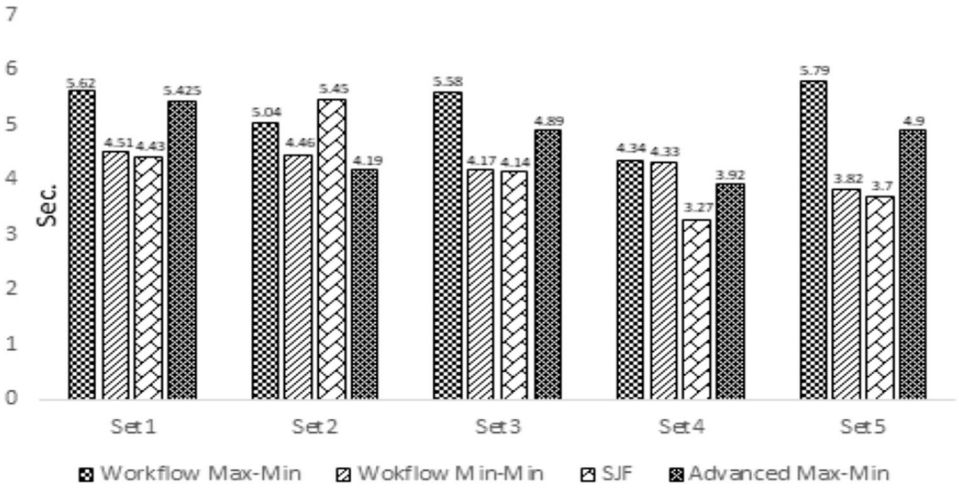


Fig. 7 Comparison of average waiting time across task scheduling algorithms on random task sets

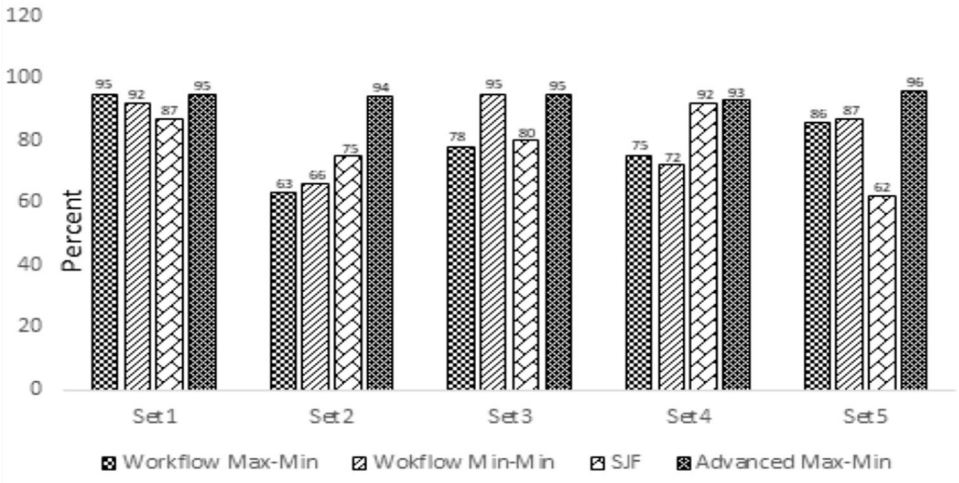


Fig. 8 Analysis of core utility for various task scheduling algorithms on random sets of tasks

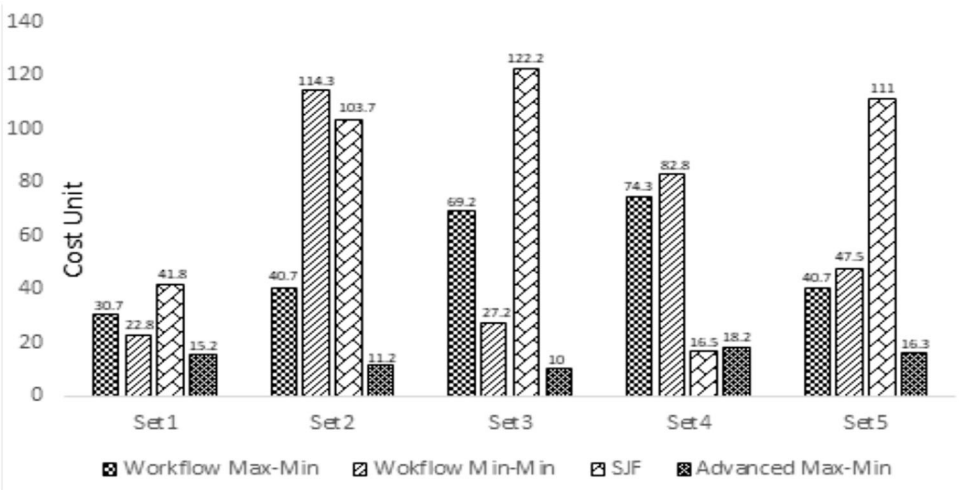


Fig. 9 Comparison of idleness costs among SJF, Max-min, Min–Min, and advanced algorithms on random task sets

Table 3 Running tasks set 1 with the advanced cost-aware algorithm with different alpha factors

α	Makespan	Cost
1	13.5	1500
0.9	13.5	1500
0.8	13.5	1500
0.7	14	1485
0.6	14.5	1445
0.5	15.5	1425
0.4	42	880
0.3	65	650
0.2	65	650
0.1	65	650
0	65	650

Author contributions Mostafa Raeisi: Implementing the research idea, Analyzing, writing the first draft and revising the paper. Omar Dakkak: Supervising, reviewing, editing, and coordinating and revising the paper. Yousef Fazea: Editing and proofreading the manuscript and data visualization. Mohammed Golam Kaosar: Supervising and Research Idea.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK).

Data availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

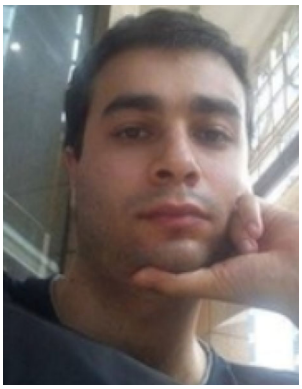
References

1. Cinar, B., Bharadiya, J.P.: Cloud computing forensics; challenges and future perspectives: a review. Asian J. Res. Comput. Sci. **16**(1), 1–14 (2023)
2. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. (CSUR) **47**(1), 1–47 (2014)
3. Varzaneh, M.R., Habbal, A., Dakkak, O.: Firewalls and internet of things security: a survey. Curr. Trends Comput. **1**(1), 22–43 (2024)
4. Raeisi-Varzaneh, M., Dakkak, O., Alaidaros, H., Avci, İ.: Internet of things: security, issues, threats, and assessment of different cryptographic technologies. J. Commun. **19**(2), 78–89 (2024)
5. Gao, J., Wang, H., Shen, H.: Task failure prediction in cloud data centers using deep learning. IEEE Trans. Serv. Comput. **15**(3), 1411–1422 (2020)
6. Arunarani, A., Manjula, D., Sugumaran, V.: Task scheduling techniques in cloud computing: a literature survey. Future Gener. Comput. Syst. **91**, 407–415 (2019)
7. Ibrahim, M., Nabi, S., Baz, A., Alhakami, H., Raza, M.S., Hussain, A., Salah, K., Djemame, K.: An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing. IEEE Access **8**, 128282–128294 (2020)
8. Raeisi-Varzaneh, M., Dakkak, O., Habbal, A., Kim, B.: Resource scheduling in edge computing: architecture, taxonomy, open issues and future research directions. IEEE Access **11**, 25329–25350 (2023)
9. Gohil, B.N., Gamit, S., Patel, D.R.: Fair fit-a load balance aware vm placement algorithm in cloud data centers. In: International Conference on Advanced Communication and Computational Technology, pp. 437–451. Springer (2019)
10. Agarwal, M., Srivastava, G.M.S.: A cuckoo search algorithm-based task scheduling in cloud computing. In: Advances in Computer and Computational Sciences: Proceedings of ICCCCS 2016, vol. 2, pp. 293–299. Springer (2018)
11. Sadiku, M.N., Musa, S.M., Momoh, O.D.: Cloud computing: opportunities and challenges. IEEE Potentials **33**(1), 34–36 (2014)
12. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y.C., Talbi, E.-G., Zomaya, A.Y., Tuyttens, D.: A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J. Parallel Distrib. Comput. **71**(11), 1497–1508 (2011)
13. Gavvala, S.K., Jatoth, C., Gangadharan, G., Buyya, R.: Qos-aware cloud service composition using eagle strategy. Future Gener. Comput. Syst. **90**, 273–290 (2019)
14. Sreenu, K., Sreelatha, M.: W-scheduler: whale optimization for task scheduling in cloud computing. Cluster Comput. **22**, 1087–1098 (2019)
15. Boveiri, H.R., Khayami, R., Elhoseny, M., Gunasekaran, M.: An efficient swarm-intelligence approach for task scheduling in cloud-based internet of things applications. J. Ambient Intell. Humaniz. Comput. **10**, 3469–3479 (2019)
16. Juarez, F., Ejarque, J., Badia, R.M.: Dynamic energy-aware scheduling for parallel task-based application in cloud computing. Future Gener. Comput. Syst. **78**, 257–271 (2018)
17. Alameen, A., Gupta, A.: Fitness rate-based rider optimization enabled for optimal task scheduling in cloud. Inf. Secur. J. Glob. Perspect. **29**(6), 310–326 (2020)
18. Sreenivasulu, G., Paramasivam, I.: Hybrid optimization algorithm for task scheduling and virtual machine allocation in cloud computing. Evol. Intell. **14**(2), 1015–1022 (2021)
19. Panda, S.K., Jana, P.K.: Load balanced task scheduling for cloud computing: a probabilistic approach. Knowl. Inf. Syst. **61**(3), 1607–1631 (2019)
20. Huang, X., Li, C., Chen, H., An, D.: Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. Cluster Comput. **23**(2), 1137–1147 (2020)
21. Khurma, R.A., Al Harahsheh, H., Sharieh, A.: Task scheduling algorithm in cloud computing based on modified round robin

algorithm. J. Theor. Appl. Inf. Technol. **96**(17), 5869–5888 (2018)
22. Arif, K.I.: A hybrid minmin & round robin approach for task scheduling in cloud computing. Int. J. Control Autom. **13**(1), 334–342 (2020)
23. Amini Motlagh, A., Movaghar, A., Rahmani, A.M.: Task scheduling mechanisms in cloud computing: a systematic review. Int. J. Commun. Syst. **33**(6), 4302 (2020)
24. Cui, H., Li, Y., Liu, X., Ansari, N., Liu, Y.: Cloud service reliability modelling and optimal task scheduling. IET Commun. **11**(2), 161–167 (2017)
25. Liu, L., Zhang, M., Buyya, R., Fan, Q.: Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurr. Comput. Pract. Exp. **29**(5), 3942 (2017)
26. Zhang, P., Zhou, M.: Dynamic cloud task scheduling based on a two-stage strategy. IEEE Trans. Autom. Sci. Eng. **15**(2), 772–783 (2017)
27. Chen, X., Cheng, L., Liu, C., Liu, Q., Liu, J., Mao, Y., Murphy, J.: A woa-based optimization approach for task scheduling in cloud computing systems. IEEE Syst. J. **14**(3), 3117–3128 (2020)
28. Prasanna Kumar, K., Kousalya, K.: Amelioration of task scheduling in cloud computing using crow search algorithm. Neural Comput. Appl. **32**(10), 5901–5907 (2020)
29. Kashikolaei, S.M.G., Hosseinabadi, A.A.R., Saemi, B., Shareh, M.B., Sangaiah, A.K., Bian, G.-B.: An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm. J. Supercomput. **76**(8), 6302–6329 (2020)
30. Gupta, A., Bhadauria, H., Singh, A.: Retracted article: Load balancing based hyper heuristic algorithm for cloud task scheduling. J. Ambient Intell. Humaniz. Comput. **12**(6), 5845–5852 (2021)
31. Dakkak, O., Fazea, Y., Nor, S.A., Arif, S.: Towards accommodating deadline driven jobs on high performance computing platforms in grid computing environment. J. Comput. Sci. **54**, 101439 (2021)
32. Sandana Karuppan, A., Meena Kumari, S., Sruthi, S.: A priority-based max–min scheduling algorithm for cloud environment using fuzzy approach. In: International Conference on Computer Networks and Communication Technologies: ICCNCT 2018, pp. 819–828. Springer (2019)
33. George Amalarethinam, D., Kavitha, S.: Rescheduling enhanced min–min (remm) algorithm for meta-task scheduling in cloud computing. In: International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018, pp. 895–902. Springer (2019)
34. Ming, G., Li, H.: An improved algorithm based on max–min for cloud task scheduling. Recent Adv. Comput. Sci. Inf. Eng. **2**, 217–223 (2012)
35. Pradhan, P., Behera, P.K., Ray, B.: Improved max-min algorithm for resource allocation in cloud computing. In: 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), pp. 22–24. IEEE (2020)
36. Brar, S.S., Rao, S.: Optimizing workflow scheduling using max–min algorithm in cloud environment. Int. J. Comput. Appl. **124**(4), 44–49 (2015)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Mostafa Raeisi-Varzaneh received the B.Sc. degree in computer science from the University of Isfahan, Isfahan, Iran, in 2014, and the M.S. degree in software engineering from the University of Kashan, Iran, in 2018. For the last five years, he has worked on resource allocation and workflow task scheduling in various computing environments, including Networks-on-Chips, cloud computing, and edge computing. His recent interest has been piqued by dynamic result validation and collusion detection, specifically in edge and IoT networks.



Omar Dakkak is an Assistant Professor at the Faculty of Engineering, Department of Computer Engineering, Karabük University, Türkiye. He received his B.E. degree in Telecommunication Engineering from Ittihad University (Syria). He completed his Ph.D. in Computer Science and M.Sc. from Universiti Utara Malaysia (UUM). In his Ph.D., he worked on scheduling problems in Grid Computing, analyzing the performance of the scheduling policies based on real workloads for better Quality of Service (QoS) criteria and building scheduling mechanisms considering High-Performance Computing (HPC) applications requirements through simulation approach using real workloads. His research interests include Scheduling Algorithms, Performance Evaluation, Optimization in Scheduling and Analyzing Datasets on HPC platforms and recently he carried out research in the field of Generative AI in Cybersecurity. He conducted several studies in other research areas such as Cloud Computing, IoT, WAN, IoV, energy efficiency for WSN, MANET and modeling and simulation for Electrical systems.

Yousef Fazea is an Assistant Professor in the Department of Computer Sciences and Electrical Engineering at Marshall University, Huntington, West Virginia, USA. He earned his Ph.D. in Computer Science from Universiti Utara Malaysia in 2017. He has published over 60 peer-reviewed scientific papers indexed in ISI and Scopus databases, authored two books, and contributed to five book chapters. His work has received numerous accolades, including best paper awards at IEEE ISCAIE-2017, IEEE ICCIS-

2020, and Springer IRICT-2023, as well as the John Marshall Research Award 2024. He serves as an editor for several esteemed journals and special issues and is a Senior Member of the IEEE. His research focuses on communications, high-performance computing, network security, next-generation computing, and emerging technologies.

Mohammed Golam Kaosar is currently working as a Senior Lecturer in the Discipline of Information Technology, Murdoch University, Australia. Prior to that he has worked in several universities including RMIT University, Victoria University, Effat University and Charles Sturt University. He has worked on a number of national and international research projects and grants. He has published many research papers in reputable journals and conferences including - IEEE Transaction on Knowledge and Data Engineering (TKDE), Data and Knowledge Engineering (DKE), IEEE International Conference on Data Engineering (ICDE), Computer Communication (ComCom), ACM SIGCOMM. He has been supervising many postgraduate students, mentoring junior colleagues, and collaborating with many national and international researchers. He is an active member of various professional organizations including – IEEE (Senior Member), Australian Computer Society (ACS), European Alliance for Innovation (EAI), The Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (ICST), South Pacific Competitive Programming Association, Industrial Engineering and Operation Management (IEOM). He contributes by organizing conferences and performing editorial responsibility in some journals. He can be contacted via Mohammed.Kaosar@murdoch.edu.au.