Scientific Research Publishing

Open Journal of Applied Sciences, 2025, 15(1), 147-167
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

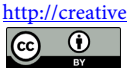
Efficient Resource Allocation in Cloud IaaS: A Multi-Objective Strategy for Minimizing Workflow Makespan and Cloud Resource Costs

Jean Edgard Gnimassoun[1], Dagou Dangui Augustin Sylvain Legrand Koffi[2], Akanza Konan Ricky N'dri[3]

[1]Département Informatique et Analyse de Données, Université de San Pedro, San Pedro, Côte d'Ivoire
[2]Département Informatique, Ecole Supérieure Africaine des Technologies d'Information et de Communication, Abidjan, Côte d'Ivoire
[3]Département Mathématiques et Informatique, Université Alassane Ouattara, Bouaké, Côte d'Ivoire
Email: gnimjean@gmail.com, dagousylvain@gmail.com, ricky.akanza@inphb.ci

How to cite this paper: Gnimassoun, J.E., Koffi, D.D.A.S.L. and N'dri, A.K.R. (2025) Efficient Resource Allocation in Cloud IaaS: A Multi-Objective Strategy for Minimizing Workflow Makespan and Cloud Resource Costs. *Open Journal of Applied Sciences*, **15**, 147-167.
https://doi.org/10.4236/ojapps.2025.151011

Received: December 6, 2024
Accepted: January 23, 2025
Published: January 26, 2025



Abstract

The ease of accessing a virtually unlimited pool of resources makes Infrastructure as a Service (IaaS) clouds an ideal platform for running data-intensive workflow applications comprising hundreds of computational tasks. However, executing scientific workflows in IaaS cloud environments poses significant challenges due to conflicting objectives, such as minimizing execution time (makespan) and reducing resource utilization costs. This study responds to the increasing need for efficient and adaptable optimization solutions in dynamic and complex environments, which are critical for meeting the evolving demands of modern users and applications. This study presents an innovative multi-objective approach for scheduling scientific workflows in IaaS cloud environments. The proposed algorithm, MOS-MWMC, aims to minimize total execution time (makespan) and resource utilization costs by leveraging key features of virtual machine instances, such as a high number of cores and fast local SSD storage. By integrating realistic simulations based on the WRENCH framework, the method effectively dimensions the cloud infrastructure and optimizes resource usage. Experimental results highlight the superiority of MOS-MWMC compared to benchmark algorithms HEFT and Max-Min. The Pareto fronts obtained for the CyberShake, Epigenomics, and Montage workflows demonstrate closer proximity to the optimal front, confirming the algorithm's ability to balance conflicting objectives. This study contributes to optimizing scientific workflows in complex environments by providing solutions tailored to specific user needs while minimizing costs and execution times.

Keywords

Cloud Infrastructure, Multi-Objective Scheduling, Resource Cost Optimization, Resource Utilization, Scientific Workflows

1. Introduction

Scientific workflows offer a compelling way to represent the complex orchestration of interdependent computations and have become widely adopted across various scientific fields [1]. They enable users to define the different steps required to process the large volumes of data typically generated by scientific experiments and to produce original scientific outcomes. The execution of such data-intensive applications, which often involve hundreds of computational tasks on large-scale distributed infrastructures, is generally managed by a Workflow Management System (WMS) [2]. Tasks such as resource selection, data management, and computation scheduling are handled by the WMS, thus shielding the end user from the complexity of these operations.

For a long time, commodity clusters and computing grids were the preferred infrastructures for running scientific workflows. Clusters, typically hosted and managed by the institution owning the workflow, facilitated resource access, while grids enabled scientists to scale their workflows by pooling resources from multiple institutions. However, with the rise of major providers like Amazon [3], Google [4], and Microsoft [5], Infrastructure as a Service (IaaS) clouds have emerged as strong competitors to clusters and grids. IaaS clouds combine the benefits of both by offering easy access to a virtually unlimited pool of resources. By carefully planning the execution of a workflow, a WMS can dynamically build a compute and storage infrastructure tailored to the workflow's specific needs, utilizing a customized set of virtual machine instances.

The description of a scientific workflow is typically independent of the characteristics of the infrastructure on which it will be executed. This provides users with greater flexibility, allowing them to run the same workflow on different infrastructures via the WMS without needing to modify their application. A direct consequence of this flexibility is that dependencies between computational tasks (where data produced by one task is consumed by another) are generally managed through files. Intermediate data is written to disk, and the file may then be transferred over the network to another storage device, where the consuming task will eventually read it.

In this paper, we propose to take advantage of two key features of a specific family of virtual machine instances provided by Amazon Web Services: a large number of cores and dedicated storage on high-speed SSD drives. By improving data locality, this approach aims to reduce the amount of data transferred over the network during workflow execution, which should have a direct and positive effect on the workflow's execution time. Additionally, we propose using realistic

simulations to design a cloud infrastructure that strikes a good balance between cost and performance. To achieve this, we developed a simulator built on the WRENCH framework [6]. WRENCH enables the construction of Workflow Management System (WMS) simulators that are accurate, fast, scalable on a single machine, and require minimal software development effort.

The main contributions of this article are therefore:

- An innovative data-aware planning algorithm designed to minimize the amount of data transferred over the network during workflow execution, based on a specific set of virtual machine instances.
- A simulation-driven approach to solve the cost-performance optimization problem and accurately size the virtual infrastructure required to execute a given workflow.

This paper is organized as follows. Section 2 reviews the related work on scheduling of scientific workflows on IaaS clouds. In Section 3, we describe the platform and application models used in this work. Then, in Section 4, we detail the proposed approach for Multi-Objective Scheduling while Section 5 explains results and discussion. Finally, we conclude this paper and present future work directions in Section 6.

2. Related Work

The rise of cloud computing has transformed the management of scientific workflows, enabling the processing of large volumes of data with flexible and scalable resources. However, task scheduling in a cloud environment remains a major challenge, particularly due to multi-objective constraints such as minimizing makespan (total execution time) and resource usage costs. This review explores recent approaches proposed to address this problem, focusing on heuristic, metaheuristic, and hybrid algorithms.

IaaS cloud environments offer flexible and scalable infrastructure but pose significant challenges for the efficient deployment of workflows. Shahid *et al*. [7] developed a multi-objective allocation strategy aimed at balancing execution time (makespan) and resource usage costs. Their approach relies on heuristic algorithms, demonstrating a significant improvement in efficiency compared to classical methods. However, despite its performance, this method does not account for dynamic workload variations, limiting its adaptability. Zhang *et al*. [8] introduced EHEFT-R, an enhancement of the Heterogeneous Earliest Finish Time (HEFT) algorithm designed for multi-objective scheduling in heterogeneous cloud environments. The algorithm incorporates an exploration strategy based on evolutionary techniques, achieving a trade-off between minimizing makespan and optimizing resource utilization. This method stands out for its efficiency in dynamic and complex environments often observed in cloud computing [8]. Hussain *et al*. [9] introduce a cost-aware and deadline-constrained scheduling algorithm in a hybrid cloud environment. The focus is on meeting deadlines while minimizing expenses, which is particularly useful for workflows with strict

temporal requirements. Simulations show that this approach outperforms traditional algorithms by balancing deadlines and costs. Mangalampalli *et al*. [10] introduced an innovative method leveraging deep reinforcement learning to prioritize and schedule workflows in cloud computing. Their approach combines dynamic modeling with task prioritization based on multi-objective goals. This method demonstrates significant improvements in scheduling accuracy and resource utilization, particularly in complex scenarios requiring rapid adaptability [10].

Hybrid algorithms combine multiple approaches to leverage their respective strengths. Malti *et al*. [11] proposed a hybrid optimization algorithm integrating evolutionary and metaheuristic techniques for task scheduling in cloud environments. Their solution demonstrated optimal task allocation while reducing operational costs. Similarly, Abualigah and Diabat [12] introduced a bio-inspired optimization algorithm based on ant-lion behavior, which excels in complex environments due to its efficient exploration of the search space. These approaches highlight the power of hybrid solutions for scheduling problems, though they often require manual parameter tuning, which may limit their generalizability. Kruekaew and Kimpan [13] propose a hybrid approach combining the Artificial Bee Colony (ABC) optimization algorithm and reinforcement learning. The objective is to address the load balancing problem in cloud environments while meeting the goals of minimizing costs and makespan. This hybrid method demonstrates increased efficiency by optimally distributing workloads. The study by Doostali *et al*. [14] presents the CP-PGWO algorithm, combining Critical Path concepts with the Grey Wolf Optimizer to address scheduling challenges. This approach underscores the ability of hybrid algorithms to exploit the structural features of workflows while adapting to workload variations in cloud environments [14]. The DE-GWO algorithm [15] combines Differential Evolution (DE) optimization and Grey Wolf Optimization (GWO) for scheduling in heterogeneous fog-cloud environments. It aims to simultaneously minimize makespan and cost while optimizing resource allocation between fog and cloud nodes. The results show significant improvements compared to existing algorithms, particularly in environments with limited resources. The Enhanced Artificial Bee Colony (ABC) algorithm, introduced by Zeedan *et al*. [16], exemplifies the effectiveness of nature-inspired algorithms in workflow scheduling problems. This hybrid approach combines the advantages of bee colony behavior with local search mechanisms, improving convergence toward optimal solutions. ABC is particularly suited for cloud computing environments due to its ability to balance conflicting objectives while ensuring efficient resource utilization [16]. Mohammadzadeh and Masdari [17] proposed a hybrid approach that combines multi-objective algorithms to optimize scientific workflows in multi-cloud environments. Their model leverages the synergy between local and global search heuristics, yielding robust solutions despite the complexity of tasks distributed across multiple clouds. Calzarossa *et al*. [18] tackled workflow planning under uncertainty, integrating

parameters such as deadlines and budgets into a multi-objective approach. Their algorithm showed substantial performance improvements in unpredictable resource conditions, emphasizing the importance of resilience in cloud computing environments. Konjaang and Xu proposed the MOWOS (Multi-Objective Workflow Optimization Strategy), which applies optimization based on probabilistic models [19]. Their study demonstrated that MOWOS effectively minimizes gaps between theoretical and practical workflow performance by better modeling task and resource uncertainties. Qin *et al*. [20] introduced a reliability-focused multi-objective memetic algorithm for multi-cloud systems. Their approach balances the trade-off between makespan minimization and reliability maximization by combining global exploration mechanisms with local adjustments to enhance solution quality. Belgacem and Beghdad-Bey [21] focused on the trade-off between makespan and execution costs of workflows in the cloud. Their model simultaneously optimizes these objectives using a heuristic method tailored for highly heterogeneous environments. Modified genetic algorithms integrate adaptive techniques to address uncertainties in cloud environments. Rizvi *et al*. [22] proposed an approach incorporating fuzzy logic to dynamically adjust the parameters of a genetic algorithm. This improves makespan and reduces costs under changing conditions. A key strength of this method lies in its ability to adapt to unforeseen workload variations, though its increased algorithmic complexity may pose challenges for implementation in large-scale systems.

The integration of multiple metaheuristics into a single optimization method represents an emerging trend. Thekkepuryil *et al*. [23] developed an approach that combines Genetic Algorithms (GA) with Particle Swarm Optimization (PSO). This hybrid method enhances the exploration and exploitation of the search space, leading to more efficient resource utilization and reduced workflow execution times. Despite these advantages, implementing such algorithms can require significant effort in terms of design and computational complexity. Cai *et al*. [24] introduce a bi-level evolutionary algorithm for data-intensive scientific workflows. This multitasking algorithm allocates resources while considering task priorities and optimizing costs and deadlines. The results demonstrate increased efficiency for complex workflows requiring intensive data management.

Workflow scheduling in cloud computing remains an active area of research, with diverse approaches ranging from heuristics to hybrid algorithms incorporating learning techniques. However, challenges related to the dynamic nature of cloud environments and multi-objective trade-offs demand new perspectives. The proposed work aims to address these gaps by leveraging innovative techniques and expanded criteria, thereby enhancing the capabilities of modern cloud systems.

The works analyzed in the literature review show a diversity of approaches to solve the multi-objective optimization problem in workflow scheduling. Each study proposes specific algorithms adapted to particular contexts, such as cloud, multi-cloud, or fog-cloud environments, focusing on various objectives, such as

minimizing makespan, costs, or load balancing. However, these contributions also present recurring limitations, particularly in terms of dynamic adaptability, scalability, or management of complex environments. To better understand the strengths and weaknesses of these methods, the following synthesis summarizes the main characteristics of the approaches studied in Table 1.

Table 1. Comparative multi-objective workflow scheduling in cloud environments.

Reference	Algorithm	Method	Limitations
[7]	Multi-Objective Workflow Allocation Strategy	Heuristic approach	Does not handle dynamic environments; lacks in-depth cost analysis
[8]	EHEFT-R	Extension of HEFT with multi-objective criteria	Limited scalability in complex multi-cloud environments
[9]	Deadline-Constrained Cost-Aware Algorithm	Hybrid optimization based on deadlines and cost constraints	Limited optimization for data-intensive workflows
[10]	Multi-Objective Prioritized Scheduling (A3C-based)	Deep reinforcement learning (A3C)	Performance depends heavily on the quality of training data
[11]	Hybrid Multi-Objective Algorithm	Hybrid metaheuristic	High algorithmic complexity; difficult to apply in real-time environments
[12]	Hybrid Antlion Optimization Algorithm	Bio-inspired algorithms (Antlion + heuristic strategies)	Lacks dynamic adaptation to resource changes
[13]	Hybrid ABC Algorithm with Reinforcement Learning	Bio-inspired (Artificial Bee Colony, ABC) + reinforcement learning	Limited efficiency for highly complex workflows
[14]	CP-PGWO	Critical path + Grey Wolf Optimization	Requires frequent manual adjustments for non-standard workflows
[15]	DE-GWO	Differential Evolution + Grey Wolf Optimization	Limited performance for unexpected workload surges
[16]	Enhanced Hybrid ABC Algorithm	Artificial Bee Colony + multi-objective optimization	Does not account for multi-cloud environments
[17]	Hybrid Multi-Objective Algorithm	Hybrid methodology for multi-cloud environments	Limited optimization for workflows with strict deadline constraints
[18]	Deadline-Budget Workflow Optimization	Bi-objective optimization (deadline + budget)	Highly sensitive to uncertainty in cloud resources
[19]	MOWOS (Multi-Objective Workflow Optimization Strategy)	Multi-objective approach for cloud environments	Simplified approach, lacks adaptability for real-time workflows
[20]	Reliability-Aware Multi-Objective Memetic Algorithm	Memetic metaheuristic	Moderate performance for data-intensive workflows
[21]	Trade-off Between Makespan and Cost	Multi-objective optimization	Does not handle dynamic workloads
[22]	Fuzzy Adaptive Genetic Algorithm	Genetic algorithm + fuzzy logic	Increased complexity as the number of tasks grows
[23]	Meta-Heuristic Based Hybrid Optimization	Hybrid metaheuristic	Suboptimal results for evolving multi-cloud environments
[24]	Bi-Level Evolutionary Algorithm	Bi-level multitask evolutionary optimization	Requires high computational resources

3. Platforms and Applications Models

In this study, we design our platform model based on a standard IaaS cloud setup, deploying various virtual machine (VM) instances across physical servers within a single datacenter. Our focus is on VMs comparable to Amazon EC2 M5 instances, particularly the M5d series, which, unlike standard M5 instances relying on Amazon Elastic Block Storage (EBS), feature local NVMe SSD storage. Table 2 provides the specifications of the M5d instances used.

The M5d instance series offers virtual cores (vCPUs) ranging from 2 to 96, with a consistent memory allocation of 4 GiB per core. These instances are typically deployed on nodes equipped with Intel Xeon Platinum 8000 series processors. A unique feature of M5d instances is the inclusion of high-speed, block-level SSD storage directly linked to the instance's lifecycle. Our study leverages this fast storage, shared among but exclusive to the instance's vCPUs, to store intermediate files generated during workflow execution. This approach minimizes data transfers across the network for tasks assigned to the same VM, with only the workflow's input and output files stored on an external storage node.

Network bandwidth between instances and EBS varies by instance size. We assume that only the largest instances (48, 64, and 96 vCPUs), which can occupy a full node, are assured bandwidths of 10, 20, and 25 Gbps, respectively. For smaller instances (2 to 32 cores), the bandwidth is proportional to the number of cores, allocated at 208.33 Mbps per core. All VMs deployed for a specific workflow are interconnected through a single switch.

For M5d instances, the VM-to-EBS connection is established via a dedicated network link, integrated into our simulation environment. We assume that the network bandwidth between a VM and EBS is proportional to the number of cores for VMs with up to 32 cores, estimated at 218.75 Mbps per core.

The costs indicated, in dollars per hour, correspond to on-demand Linux instances in the US-East (Ohio) region as of the time of writing this article.

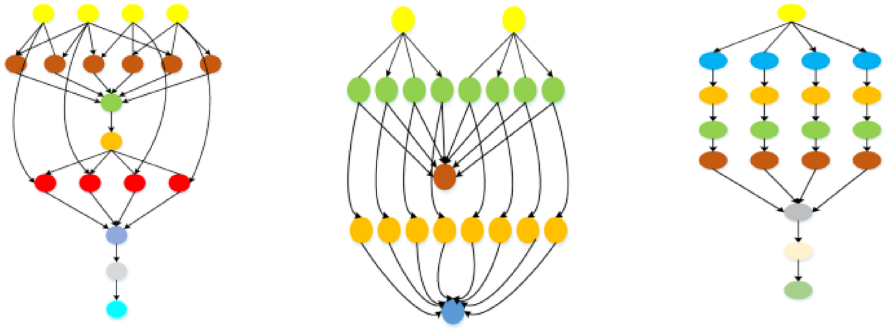
Table 2. Characteristics of the AWS M5d instances.

Model	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)	EBS Bandwidth (Mbps)	Cost ($ per Hour)
m5d.large	2	8	1 × 75 NVMe SSD	Up to 10	Up to 3,500	0.113
m5d.xlarge	4	16	1 × 150 NVMe SSD	Up to 10	Up to 3,500	0.226
m5d.2xlarge	8	32	1 × 300 NVMe SSD	Up to 10	Up to 3,500	0.452
m5d.4xlarge	16	64	2 × 300 NVMe SSD	Up to 10	3,500	0.904
m5d.8xlarge	32	128	2 × 600 NVMe SSD	10	5,000	1.808
m5d.12xlarge	48	192	2 × 900 NVMe SSD	10	7,000	2.712
m5d.16xlarge	64	256	4 × 600 NVMe SSD	20	10,000	3.616
m5d.24xlarge	96	384	4 × 900 NVMe SSD	25	14,000	5.424

The scientific workflows we aim to schedule are modeled as Directed Acyclic Graphs (DAGs), denoted as $G=\{T,\mathcal{E}\}$, where $T=\{t^i \mid i=1,\cdots,T\}$ represents the set of vertices corresponding to the computational tasks of the workflow, and $\mathcal{E}=\{e^{i,j} \mid (i,j)\in\{1,\cdots,T\}\times\{1,\cdots,T\}\}$ is the set of edges between these vertices, indicating either data dependencies (*i.e.*, file transfers) or control flows between tasks. We focus on workflows consisting of numerous sequential tasks that run on a single core, reflecting the structure of real-world scientific applications. Each task within the workflow has an estimated execution time, requires specific input files to start, and produces output files upon completion. To evaluate our contributions, we consider three workflow applications from the Pegasus Workflow Gallery [25]. Specifically, we use synthetic workflows generated by the Workflow Generator Toolkit [26], which resemble those used in real-world scientific applications but with a higher task count. The primary characteristics of these applications are presented in **Table 3**, and their structures are illustrated in **Figure 1**.

Table 3. Some characteristics of used workflows.

Workflow	Tasks	Input Files Size (GB)	Total Files Size (GB)
CyberShake	1000	150.76	400.39
Epigenomics	997	1217.72	1230.93
Montage	1000	0.65	17.32



(a) Montage (b) Cybershake (c)Epigenomics

Figure 1. Structure of the used workflows (http://pegasus.isi.edu).

- **Epigenomics**: A data processing pipeline designed to automate the execution of various genome sequencing operations.
- **CyberShake**: An application developed by the Southern California Earthquake Center to assess earthquake hazards.
- **Montage**: An astronomy application that creates customized mosaics of the sky by stitching together multiple images.

4. Proposed Approach for Multi-Objective Scheduling

To generate the different platforms and avoid wasting resources, *i.e.* avoid leasing resources that will not be used, the number of tasks that can be executed in parallel

for each of the workflows is determined. To do this, the platform is oversized, *i.e.* this platform has as many cores as there are tasks in the workflow. In the case of the workflows used in this study, there are 1000 tasks, so the platform with 1000 cores is considered. This allowed to determine the total number of cores that could be used in parallel for each of the workflows. Thus, for CyberShake, 374 cores are used in parallel while for Epigenomics and Montage it is respectively 246 and 662 cores used in parallel. This preliminary study will be used in following section in order to determine the limit of platforms to be used for each workflow in experiments. For each workflow, we consider infrastructures where we vary (*i.e.* increase) the maximum number of cores per VM (from 2 cores to 96 cores maximum). For a total number of cores to be used, we generate platforms with 2 cores per VM, 4 cores per VM, ..., 96 cores per VM. The platforms composed of 2 cores per VM provide poor execution times compared to platforms of 32 cores per VM and they themselves provide poor execution times compared to platforms of 96 cores per VM. The increasing of the total number of cores per VM provides good execution times. It is for this reason that this study favor large VMs (*i.e.* VMs with several cores) because these VMs can execute several tasks in parallel and considerably reducing the makespan.

4.1. Makespan Minimization Approach

Our primary focus is on workflows consisting of a substantial number of sequential tasks, each running on a single core (a pattern frequently observed in real-world scientific applications). Each task in the workflow has a predefined or estimated duration, requires specific input files to start, and generates output files upon completion.

To represent the input and output files for a given task t^i, we use the notation $Input_k^i$ for input files and $Output_k^i$ for output files, where k denotes the file index.

When an output file generated by one task t^i is needed as an input by another task t^j, this establishes a data dependency between t^i and t^j, represented by the $e^{i,j}$. Furthermore, there are input files that are not produced by any tasks within the workflow; these are referred to as the workflow's entry files and serve as the starting point for its execution.

Conversely, the output files that are not required by any subsequent tasks in the workflow are referred to as the exit files. To aid in the scheduling process, two important quantities are defined for each task in the workflow. These metrics are essential for making effective scheduling decisions: the Local Input Volume (LIV) of task t^i on machine vm_k, denoted as LIV_k^i, represents the total size of the input files required by task t^i that are locally available on vm_k; similarly, the Local Output Volume (LOV) of t^i on machine vm_k, denoted as LOV_k^i, represents the cumulative size of the output files generated by task t^i. These output files are needed by the successors of task t^i, and the successors are also scheduled on vm_k. If a file is used by multiple successors, its size is counted as many times

as there are successors. The LIV of an entry task and the LOV of an exit task are set to zero by definition.

During workflow execution, all intermediate files (those generated by one task and consumed by another) are stored locally on the SSD storage of one or more machines. In contrast, the workflow's entry and exit files are stored on the Elastic Block Store (EBS) service, accessible to all machines participating in the workflow. The time required to transfer a file between machines includes the time to read the file from the source machine's disk, the network transfer duration, and the time to write the file to the destination machine's disk.

The goal of this algorithm is to minimize the total execution time (makespan) by considering both the parallel execution of tasks and the reduction of data transfers across the network. Although the study does not explicitly account for data transfer time, the algorithm is designed to minimize network transfers as much as possible. However, completely avoiding these transfers is often unfeasible due to the complexity of task dependencies.

The algorithm starts by creating a sorted scheduling list that includes all tasks in the workflow (lines 1 - 2), ordered by their bottom-level values in descending order. The rank of a task t^i, denoted $Rank^i$, is the length of the longest path from t^i to the end of the workflow. This rank includes the estimated duration of all tasks along this path, including t^i. Following the approach in [27], we also account for the estimated data transfer costs when computing the rank values of tasks. This prioritization ensures that the most critical tasks are given higher priority, while also respecting the dependencies between tasks. Next, the algorithm assigns an initial mapping for each task t^i in the set T (lines 3-7). The chosen virtual machine vm from the set VM is the one that: (i) minimizes the start time of t^i and (ii) maximizes the volume of input files already stored locally for t^i. The reasoning behind this is that when multiple virtual machines can start t^i at the same time, the algorithm favors the machine that reduces data transfers over the network, thus improving overall efficiency. Since all the virtual machine instances in consideration are multi-core, scheduling a task t^i on a machine vm requires maintaining a local schedule within the virtual machine. To optimize the use of the available cores, each virtual machine is managed similarly to how a job and resource manager would handle tasks. Keeping track of the number of processors available on each virtual machine is essential for determining the earliest possible start time for a new task on that machine (*i.e.*, st_k^i). Once vm is selected for the execution of t^i, the number of processors available on vm is updated accordingly (line 6).

The workflow is processed level by level, from the bottom to the top (lines 8–40). This approach is motivated by the fact that, during the initial top-to-bottom placement, only the data volume from a task's direct predecessors can be considered. It is not feasible to account for the data locality required by a task's direct descendants, as their placements have not yet been established. Consequently, this can result in avoidable data transfers.

Level 0 is defined as the topmost level of the Directed Acyclic Graph (DAG), containing all entry tasks of the workflow. Each subsequent task's level is recursively calculated as the maximum level of its predecessors plus one. The total number of levels in the workflow is denoted by L.

We begin by saving the current start time $_{c}st^{i}$ and the current mapping vm^{i} for each task t^{i} in T^{l} (lines 11-12). Next, we calculate the local volume LV_{k}^{i} for task t^{i} on machine vm_{k} (lines 13-15). Afterward, we store the local volume for the current mapping of t^{i} (line 16) before canceling the current mapping (line 17).

Canceling the mapping of all the tasks at a given level creates idle processors of different machines. These processors can be leveraged to enhance data locality by "migrating" certain tasks from one machine to another. The conditions for migrating a task t^{i} from its previous mapping to a new mapping on vm_{k} are twofold: it must improve data locality, *i.e.*, $LV_{k}^{i} \geq {}_{c}LV^{i}$, and reduce the task's start time, *i.e.*, $st_{k}^{i} \leq {}_{c}st^{i}$ (lines 21 - 27).

At each step, the algorithm aims to find a better mapping for each task by prioritizing the machine that offers the largest increase in local data volume. If this machine also allows the task to start earlier, it is selected for a new tentative mapping. The first option (lines 28 - 31) is to execute the task and update the processor count, provided that its ready time aligns with its calculated start time (line 29), which does not include data transfer time.

The second option (lines 32 - 37) applies if the ready time of a task is strictly earlier than its calculated start time (line 32). In this case, if all predecessor tasks t^{j}, whose calculated finish time coincides with the calculated start time of the ready task t^{i}, have finished their execution (*i.e.*, they have released at least one processor) (line 33), then t^{i} is executed, and the count of available processors is updated (lines 34–35). Note that the calculated finish and start times do not factor in data transfer time but rather consider the data transfer volumes.

Equations (1) and (2) will be used subsequently to determine the non-dominated solutions that we will represent on the Pareto front. It should be noted that the proposed algorithm allows to minimize the makespan for a given total number of computing cores, *i.e.* for a given platform. Equation (2) allows to determine the cost corresponding to the use of this IaaS platform.

Algorithm: Minimizing makespan

1: Compute $Rank^{i}$ for each task t^{i}

2: Sort T by decreasing $Rank^{i}$ values

3: **for all** $t^{i} \in T$ **do**

4: $vm \leftarrow \{vm_{k} \in VM \mid st_{k}^{i}$ is minimal **and** LIV_{k}^{i} is maximal$\}$

5: Map t^{i} on vm

6: Update $proc_{k}$

7: **endfor**

Continued

```
for l = L to 0 do
    T^l ← tasks in level l sorted by decreasing Rank values
    for all t^i ∈ T^l do
        _c st^i ← current start time of t^i
        vm^i ← current mapping of t^i
        for all vm_k ∈ VM do
            LV_k^i ← LIV_k^i + LOV_k^i
        endfor
        _c LV^i ← current local volume of t^i
        Cancel the current mapping of t^i
    endfor
    for all t^i ∈ T^l do
        sort VM by decreasing LV_k^i values
        while LV_k^i ≥ _c LV^i do
            if st_k^i ≤ _c st^i then
                Map t^i on vm_k
                Update proc_k
                break
            endif
        endwhile
        if proc_k ≥ 0 then
            if rt_k^i = st_k^i then
                Execute t^i on vm_k
                Update proc_k
            else if rt_k^i < st_k^i then
                if all tasks t^j | ft_k^j = st_k^j are computed then
                    Execute t^i on vm_k
                    Update proc_k
                endif
            endif
        endif
    endfor
endfor
```

The makespan of a workflow, also referred to as its execution time, is calculated as the difference between the real start time of the first task and the real finish time of the last task. The real start time of a task t^i (denoted as $RST\left(t^i\right)$) is when the task begins transferring its input files, while the real finish time ($RFT\left(t^i\right)$) is when

the task completes transferring its output files. Therefore, the makespan of a workflow can be defined as:

$$Makespan = \max_{t^i \in T} \left\{ RFT\left(t^i\right) \right\} - \min_{t^i \in T} \left\{ RST\left(t^i\right) \right\} \quad (1)$$

4.2. Cost Optimization Strategy for Cloud Resource Utilization

We assume that each utilized virtual machine (VM) is billed on a per-second basis. Consequently, the cost associated with the execution of the workflow is defined as:

$$Cost = cost\_per\_core * total\_cores * Makespan \quad (2)$$

where $cost\_per\_core$ represents the unit cost per core per second, and $total\_cores$ denotes the total number of core utilized during the execution of the workflow.

5. Results and Discussion

Figure 2 highlights the inverse relationship between the two objectives under study: makespan (total execution time) and cost, as a function of the number of allocated cores. Initially, as the number of cores increases, the makespan decreases significantly. This is due to the enhanced parallelism capability, allowing more tasks to be executed simultaneously. However, this reduction in execution time is accompanied by a progressive increase in cost, as resource usage prices in a cloud environment are directly tied to the number of cores utilized. This inverse relationship becomes even more pronounced beyond a certain threshold (approximately 50 - 100 cores), where adding additional resources results in only marginal reductions in makespan, while costs continue to rise almost linearly. In other words, beyond a certain point, each additional core contributes little to performance improvement but significantly inflates costs. This observation underscores the challenge of achieving a perfect balance between these two objectives, as further reducing makespan entails exponential cost increases.

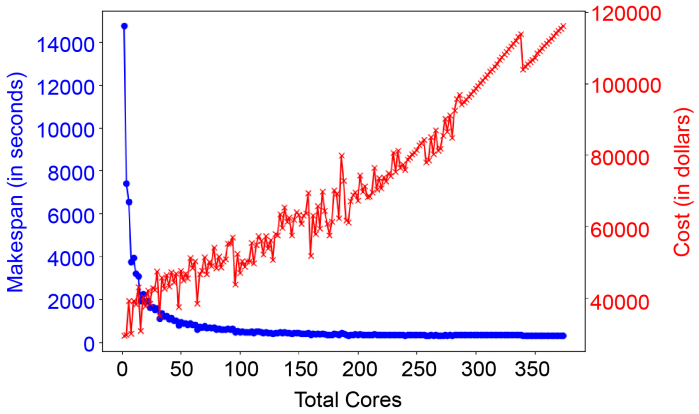


Figure 2. Makespan and cost evolution for the CyberShake scientific workflow.

In the Epigenomics workflow (Figure 3), a trend similar to that observed in other workflows such as CyberShake (Figure 2) becomes evident: the two objectives (makespan and cloud resource usage cost) exhibit opposing behaviors as the number of available cores increases. When the number of cores increases, the makespan (depicted by the blue curve) initially decreases drastically. This improvement is due to the enhanced ability to parallelize more tasks within the workflow, thereby reducing the total execution time. Simultaneously, the red curve representing the cost shows a progressive and non-linear increase with the number of cores. This is because the cost calculation formula depends on both the makespan and the total number of cores. Although the makespan decreases, the addition of more resources drives up expenses, particularly in configurations with a large number of cores. An important observation in such graphs is the presence of a zone where a trade-off between minimal makespan and acceptable cost is achievable. In the case of the Epigenomics workflow, this zone appears to be in the range of 40 to 100 cores, where the makespan reduction is significant, yet the cost increase remains moderate. Beyond a certain threshold (approximately 200 cores), the cost becomes excessively high without yielding a substantial reduction in makespan. This underscores the importance of an appropriate resource-sizing strategy to avoid unnecessary costs.

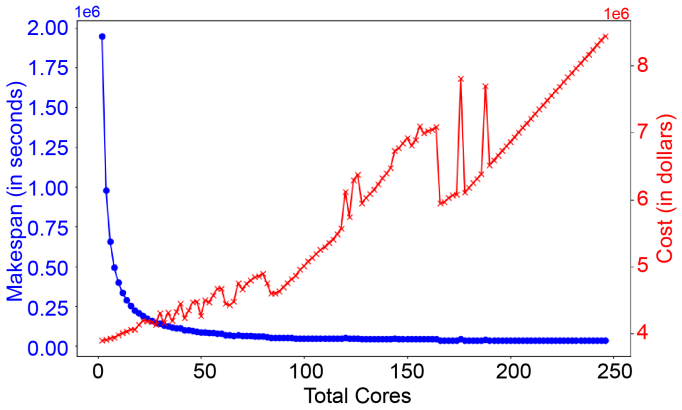


Figure 3. Makespan and cost evolution for the Epigenomics scientific workflow.

Figure 4 for the Montage workflow illustrates an inversely proportional relationship between the makespan and cost as a function of the total number of cores used. When the number of cores increases, the workflow execution time (makespan) decreases rapidly in the initial stages. This effect is particularly pronounced for smaller numbers of cores, where adding additional resources significantly enhances performance. However, as the number of cores continues to grow, the improvement becomes marginal, and the curve flattens, indicating a saturation effect where additional resources have a limited impact on the makespan. Conversely, the cost increases almost linearly with the total number of cores. This phenomenon is explained by the cost calculation formula, where

increased resource usage directly translates into higher expenses, even as the makespan improvements diminish. This highlights the inherent trade-off in cloud resource allocation between minimizing execution time and managing costs effectively.

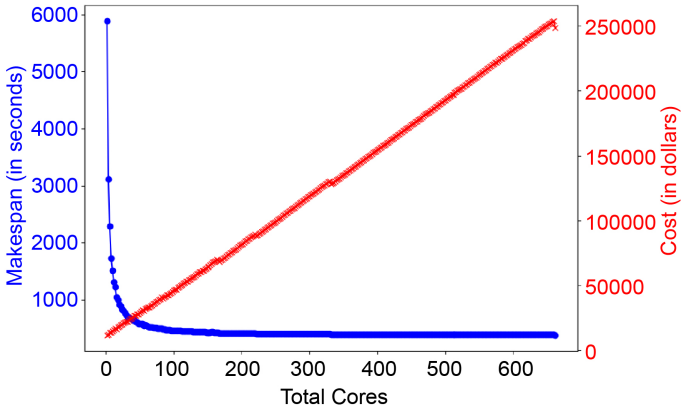


Figure 4. Makespan and cost evolution for the Montage scientific workflow.

The main purpose of such a study lies in identifying an optimal trade-off between the conflicting objectives of makespan and cost. This trade-off is represented by a Pareto front, which encompasses all non-dominated solutions, where an improvement in one objective can only be achieved at the expense of the other. Analyzing the behavior of these curves not only allows us to visualize the complex relationship between these two objectives but also to identify strategic solutions tailored to the specific needs of users or applications. Below, we present the Pareto fronts obtained for the different workflows studied, highlighting the optimal solutions in this multi-objective space.

The main objective of our algorithm is to help IaaS cloud users find a good trade-off between the execution time and cost of their workflow by selecting sets of VM instances on a Pareto front. In this section, we evaluate the quality of the Pareto front solutions produced by our algorithm. In the previous sections, we have shown that to minimize the execution time for a fixed number of cores, priority should be given to large VM instances. To obtain the Pareto front given by our algorithm, it is therefore sufficient to perform a simulation for each total number of cores and discard all dominated solutions. Two solutions, S_i and S_j, are considered equivalent if the same tasks are assigned to different but equivalent VMs with identical start times. During each iteration, only one representative of such equivalent solutions is retained. Furthermore, in all iterations except the final one, only strictly dominated solutions are eliminated. A solution S_i is strictly dominated by a solution S_j if both the execution time and cost of S_i are strictly higher than those of S_j. This approach ensures that intermediate solutions, which could lead to better schedules, are not prematurely discarded. In the final iteration, however, all simply dominated solutions (*i.e.*, solutions where one metric is worse) are

removed. The three algorithms exhibit similar behaviors, with a rapid decrease in cost as makespan increases from the initial points, followed by stabilization. The MOS-MWMC algorithm (red points) demonstrates superior dominance over the entire set of solutions. Its curve is generally closer to the origin, indicating a better trade-off between makespan and cost. Compared to HEFT and MAX-MIN, MOS-MWMC is able to provide solutions with slightly lower costs while maintaining competitive makespans. The HEFT algorithm (dashed green line) follows a similar trajectory but is slightly less efficient than MOS-MWMC for most trade-offs. HEFT solutions tend to have slightly higher costs, especially in the low makespan region. However, for very high makespans, HEFT reaches the performance of MOS-MWMC, indicating convergence. The MAX-MIN algorithm (dashed blue line) is generally less effective. Its curve is often above those of MOS-MWMC and HEFT, indicating higher costs for similar makespans. This difference is more pronounced in the region where the makespan is low, suggesting that MAX-MIN is not optimal for minimizing both objectives simultaneously. The superiority of MOS-MWMC is evident in the critical areas where the trade-off between makespan and cost is most important (low makespans). This demonstrates the algorithm's ability to effectively balance these two conflicting objectives for the scientific workflow CyberShake (Figure 5).

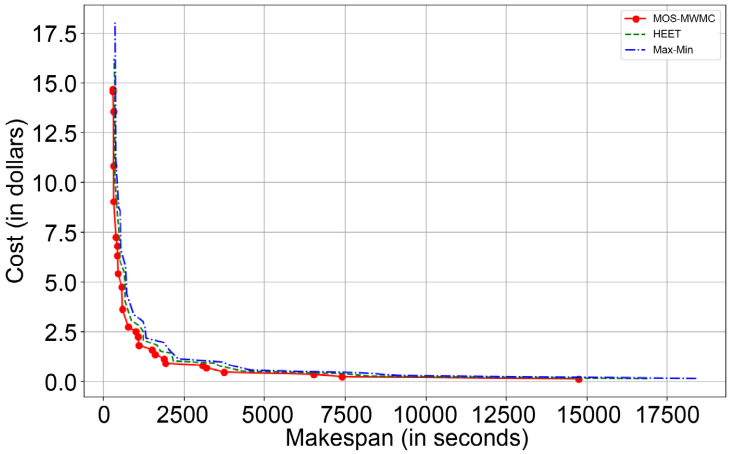


Figure 5. Pareto fronts for CyberShake.

The Epigenomics workflow (Figure 6) reveals a markedly superior performance of the MOS-MWMC algorithm compared to the HEFT and Max-Min methods. The points generated by MOS-MWMC stand out due to their proximity to the Pareto front, demonstrating an enhanced ability to provide optimal solutions by simultaneously minimizing both cost and makespan. This proximity reflects the effectiveness of MOS-MWMC in balancing the two objectives, a crucial aspect for complex workflows. In contrast, the HEFT and Max-Min algorithms, while following similar trends, struggle to achieve the same levels of performance. For equivalent makespan values, the costs associated with the solutions of HEFT and Max-Min are significantly higher, highlighting their relative inadequacy for

this type of workflow. Furthermore, the solutions from these two algorithms exhibit greater dispersion, suggesting variability in their ability to effectively optimize the objectives.

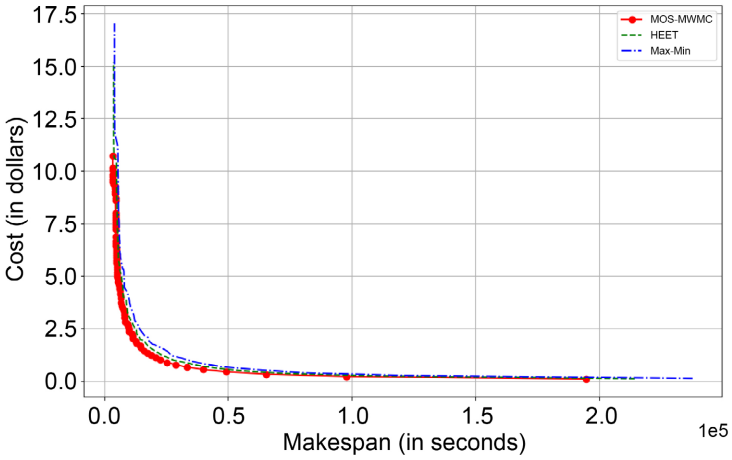


Figure 6. Pareto fronts for Epigenomics.

In the case of the Montage workflow (**Figure 7**), MOS-MWMC confirms its superiority by generating solutions that also approach the Pareto front, even in scenarios with more pronounced trade-offs between cost and makespan. The robustness of this algorithm is particularly evident in its ability to maintain an optimal balance between the two objectives, even for extreme points where cost or execution time becomes the priority. In contrast, the HEFT and Max-Min algorithms, while displaying similar trajectories to those observed for Epigenomics, remain inferior in terms of the quality of the solutions produced. HEFT, while slightly more effective than Max-Min in this case, fails to reach the optimal trade-offs proposed by MOS-MWMC. The increased complexity of the Montage workflow is reflected in the variability of the solutions obtained, further highlighting the adaptive capacity of MOS-MWMC in the face of diverse workflow profiles.

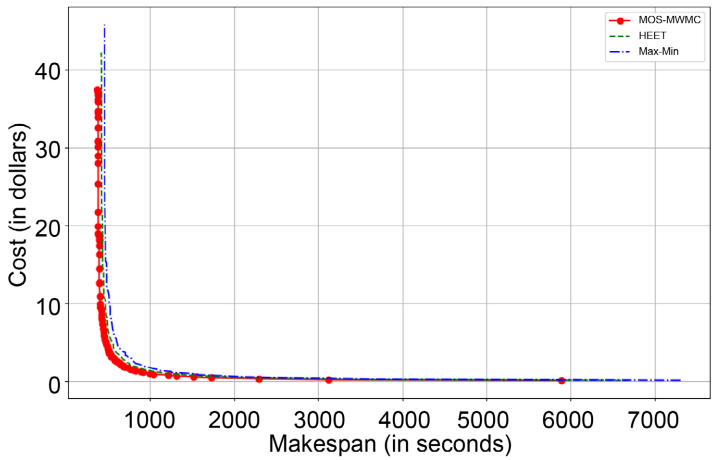


Figure 7. Pareto fronts for Montage.

The analysis of the results for the CyberShake, Epigenomics, and Montage workflows highlights the consistent superiority of the MOS-MWMC approach over the HEFT and Max-Min algorithms. MOS-MWMC excels in its ability to balance the conflicting objectives of cost and makespan, demonstrating remarkable robustness and adaptability in complex and varied workflow environments. The solutions generated by this algorithm are characterized by their proximity to the Pareto front, offering trade-offs that effectively meet the demands of cloud environments. In summary, MOS-MWMC emerges as the preferred solution for scheduling scientific workflows in cloud computing environments, especially in scenarios where multi-objective optimization is critical. Its consistent performance and flexibility make it an essential approach for addressing challenges related to resource efficiency and time constraints.

Table 4 represents a comparison of the performance parameters for the three algorithms MOS-MWMC, HEFT and Max-Min, based on typical results from our simulations. The MOS-MWMC algorithm dominates on most criteria, notably due to its ability to simultaneously optimize makespan and costs, while efficiently exploiting virtual machine (VM) resources. As for HEFT, this algorithm offers acceptable performances but remains inferior to MOS-MWMC on multi-objective trade-offs, particularly for complex workflows. On the other hand, the performances of the Max-Min algorithm are generally inferior, notably in terms of costs and makespan minimization, which makes it less suitable for demanding cloud environments.

Table 4. Performance criteria for MOS-MWMC, HEFT, and Max-Min.

Performance Criteria	MOS-MWMC	HEFT	Max-Min
Proximity to Pareto Front	Very high (optimal solutions close to the front): 98%	Medium (solutions slightly farther from the front): 85%	Low (solutions far from the front): 70%
Makespan Minimization	Very efficient (maximum reduction)	Efficient but less than MOS-MWMC	Less efficient (high makespan)
Cost Reduction	Excellent (optimized cost)	Moderate (cost moderately optimized)	Low (high cost)
Task Dependency Management	Very well managed	Adequate but can be improved	Less effective
Adaptability to Dynamic Environments	Very high (flexibility and robustness)	Medium (limited adaptability)	Low (not flexible)
Algorithmic Complexity	Moderate (well-balanced)	Medium (less complex than MOS-MWMC)	Low (simple algorithm)
Data Transfer Reduction	Very effective: 60%	Moderate: 40%	Less effective: 25%
Resource Utilization	Optimized (better VM utilization): 85%	Moderate: 75%	Suboptimal: 60%
Convergence Time (ms)	Fast: 500	Average: 700	Slow: 1200

6. Conclusions

Infrastructure as a Service (IaaS) clouds now enable scientists to run data-intensive workflows on customized infrastructures designed to meet the specific computing and storage needs of these applications. Selecting the appropriate set of virtual machine instances to form these infrastructures is a challenging task, typically handled by Workflow Management Systems. Maximizing performance hinges on effectively utilizing the unique characteristics of these virtual machine instances.

In this paper, we proposed an innovative multi-objective approach called MOS-MWMC. The MOS-MWMC algorithm establishes itself as a benchmark solution for multi-objective workflow scheduling in IaaS cloud environments. Unlike the HEFT and Max-Min algorithms, MOS-MWMC stands out for its ability to simultaneously optimize makespan and costs while maintaining robust and adaptable performance. This efficiency is attributed to several key strengths. Firstly, the algorithm generates solutions that are close to the Pareto front, ensuring an optimal trade-off between conflicting objectives. Secondly, it significantly reduces data transfers through a strategy of local storage for intermediate files, thereby minimizing network communication, a major advantage over HEFT and Max-Min.

Moreover, MOS-MWMC makes optimal use of virtual machine resources by leveraging their features, such as a high number of cores and fast storage, resulting in better task distribution and more efficient utilization of cloud infrastructures. Additionally, its robustness in handling complex workflows, as demonstrated by its performance on CyberShake, Epigenomics, and Montage, makes it particularly suitable for diverse and demanding scenarios. Experimental results show that MOS-MWMC consistently outperforms HEFT and Max-Min in minimizing makespan and optimizing costs, further validating its relevance in addressing the challenges of modern cloud environments.

Finally, this study opens up promising future prospects, including the evaluation of MOS-MWMC in real cloud environments and the integration of additional objectives, such as energy consumption reduction. These potential advancements would further enhance the relevance of this algorithm for industrial applications, where performance, flexibility, and cost control are critical.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Taylor, I.J. Deelman, E. Gannon, D.B. and Shields, M. (2007) Workflows for e-Science. Springer. https://doi.org/10.1007/978-1-84628-757-2

[2] Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., *et al.* (2015) Pegasus, a Workflow Management System for Science Automation. *Future Generation Computer Systems*, **46**, 17-35. https://doi.org/10.1016/j.future.2014.10.008

[3] Amazon EC2—Service d'hébergement cloud évolutif. https://aws.amazon.com/fr/ec2/

[4] des PVC de disque persistant. https://cloud.google.com/products/compute

[5] "Services de Cloud Computing. Microsoft Azure. https://azure.microsoft.com/fr-fr/

[6] Casanova, H., Tanaka, R., Koch, W. and Ferreira da Silva, R. (2021) Teaching Parallel and Distributed Computing Concepts in Simulation with Wrench. *Journal of Parallel and Distributed Computing*, **156**, 53-63. https://doi.org/10.1016/j.jpdc.2021.05.009

[7] Shahid, M., Ashraf, Z., Alam, M., Ahmad, F. and Imran, M. (2021) A Multi-Objective Workflow Allocation Strategyin IaaS Cloud Environment. 2021 *International Conference on Computing, Communication, and Intelligent Systems* (*ICCCIS*), Greater Noida, 19-20 February 2021, 308-313. https://doi.org/10.1109/icccis51004.2021.9397081

[8] Zhang, H., Wu, Y. and Sun, Z. (2021) EHEFT-R: Multi-Objective Task Scheduling Scheme in Cloud Computing. *Complex & Intelligent Systems*, **8**, 4475-4482. https://doi.org/10.1007/s40747-021-00479-7

[9] Hussain, M., Luo, M., Hussain, A., Javed, M.H., Abbas, Z. and Wei, L. (2023) Deadline-Constrained Cost-Aware Workflow Scheduling in Hybrid Cloud. *Simulation Modelling Practice and Theory*, **129**, Article 102819. https://doi.org/10.1016/j.simpat.2023.102819

[10] Mangalampalli, S., Hashmi, S.S., Gupta, A., Karri, G.R., Rajkumar, K.V., Chakrabarti, T., *et al.* (2024) Multi Objective Prioritized Workflow Scheduling Using Deep Reinforcement Based Learning in Cloud Computing. *IEEE Access*, **12**, 5373-5392. https://doi.org/10.1109/access.2024.3350741

[11] Malti, A.N., Hakem, M. and Benmammar, B. (2023) A New Hybrid Multi-Objective Optimization Algorithm for Task Scheduling in Cloud Systems. *Cluster Computing*, **27**, 2525-2548. https://doi.org/10.1007/s10586-023-04099-3

[12] Abualigah, L. and Diabat, A. (2020) A Novel Hybrid Antlion Optimization Algorithm for Multi-Objective Task Scheduling Problems in Cloud Computing Environments. *Cluster Computing*, **24**, 205-223. https://doi.org/10.1007/s10586-020-03075-5

[13] Kruekaew, B. and Kimpan, W. (2022) Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm with Reinforcement Learning. *IEEE Access*, **10**, 17803-17818. https://doi.org/10.1109/access.2022.3149955

[14] Doostali, S., Babamir, S.M. and Eini, M. (2021) CP-PGWO: Multi-Objective Workflow Scheduling for Cloud Computing Using Critical Path. *Cluster Computing*, **24**, 3607-3627. https://doi.org/10.1007/s10586-021-03351-y

[15] Shukla, P. and Pandey, S. (2023) DE-GWO: A Multi-Objective Workflow Scheduling Algorithm for Heterogeneous Fog-Cloud Environment. *Arabian Journal for Science and Engineering*, **49**, 4419-4444. https://doi.org/10.1007/s13369-023-08425-0

[16] Zeedan, M., Attiya, G. and El-Fishawy, N. (2022) Enhanced Hybrid Multi-Objective Workflow Scheduling Approach Based Artificial Bee Colony in Cloud Computing. *Computing*, **105**, 217-247. https://doi.org/10.1007/s00607-022-01116-y

[17] Mohammadzadeh, A. and Masdari, M. (2021) Scientific Workflow Scheduling in Multi-Cloud Computing Using a Hybrid Multi-Objective Optimization Algorithm. *Journal of Ambient Intelligence and Humanized Computing*, **14**, 3509-3529. https://doi.org/10.1007/s12652-021-03482-5

[18] Calzarossa, M.C., Vedova, M.L.D., Massari, L., Nebbione, G. and Tessera, D. (2021) Multi-Objective Optimization of Deadline and Budget-Aware Workflow Scheduling in Uncertain Clouds. *IEEE Access*, **9**, 89891-89905. https://doi.org/10.1109/access.2021.3091310

[19] Konjaang, J.K. and Xu, L. (2021) Multi-Objective Workflow Optimization Strategy (MOWOS) for Cloud Computing. *Journal of Cloud Computing*, **10**, Article No. 11. https://doi.org/10.1186/s13677-020-00219-1

[20] Qin, S., Pi, D., Shao, Z., Xu, Y. and Chen, Y. (2023) Reliability-Aware Multi-Objective Memetic Algorithm for Workflow Scheduling Problem in Multi-Cloud System. *IEEE Transactions on Parallel and Distributed Systems*, **34**, 1343-1361. https://doi.org/10.1109/tpds.2023.3245089

[21] Belgacem, A. and Beghdad-Bey, K. (2021) Multi-Objective Workflow Scheduling in Cloud Computing: Trade-off between Makespan and Cost. *Cluster Computing*, **25**, 579-595. https://doi.org/10.1007/s10586-021-03432-y

[22] Rizvi, N., Ramesh, D., Wang, L. and Basava, A. (2023) A Workflow Scheduling Approach with Modified Fuzzy Adaptive Genetic Algorithm in IaaS Clouds. *IEEE Transactions on Services Computing*, **16**, 872-885. https://doi.org/10.1109/tsc.2022.3174112

[23] Kakkottakath Valappil Thekkepuryil, J., Suseelan, D.P. and Keerikkattil, P.M. (2021) An Effective Meta-Heuristic Based Multi-Objective Hybrid Optimization Method for Workflow Scheduling in Cloud Computing Environment. *Cluster Computing*, **24**, 2367-2384. https://doi.org/10.1007/s10586-021-03269-5

[24] Cai, X., Li, M., Zhang, Y., Zhao, T., Zhang, W. and Chen, J. (2024) Multitasking Bi-Level Evolutionary Algorithm for Data-Intensive Scientific Workflows on Clouds. *Expert Systems with Applications*, **238**, Article 121833. https://doi.org/10.1016/j.eswa.2023.121833

[25] Automate, Recover, and Debug Scientific Computations. Pegasus WMS. https://pegasus.isi.edu/

[26] Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M. and Vahi, K. (2008) Characterization of Scientific Workflows. 2008 *Third Workshop on Workflows in Support of Large-Scale Science*, Austin, 17 November 2008, 1-10. https://doi.org/10.1109/works.2008.4723958

[27] Gerasoulis, A. and Yang, T. (1992) A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors. *Journal of Parallel and Distributed Computing*, **16**, 276-291. https://doi.org/10.1016/0743-7315(92)90012-c