**ECTI Transactions on Computer and Information Technology**
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

# HyEcoSec: Hybrid Cloud Economic and Secure Workflow Scheduling System

Sarra Hammouti[1], Belabbas Yagoubi[2] and Sid Ahmed Makhlouf[3]

## ABSTRACT

Scheduling scientific workflows in cloud environments is challenging due to the need to balance multiple objectives while satisfying various constraints. In this paper, we propose a Hybrid cloud Economic and Secure workflow scheduling system (HyEcoSec) designed to minimize makespan and cost while ensuring security and meeting budget and deadline constraints. HyEcoSec comprises two modules: a Security Compliance Module to manage user-defined security requirements, and a Scheduling Module integrating a static scheduler based on the Multi-Portions Slime Mould Algorithm (MPSMA) and a dynamic re-scheduler to handle runtime failures. MPSMA, an enhanced version of the Slime Mould Algorithm (SMA), balances makespan and cost using a modified Pareto approach under budget and deadline constraints. Performance evaluation using the WorkflowSim demonstrates the effectiveness of HyEcoSec. Compared to baseline methods, MPSMA achieves a competitive execution time, reduces makespan by 26% and cost by 43%, and improves resource utilization by 22%. It ensures compliance with budget and deadline constraints in almost all cases, outperforming the compared algorithms. The integration of security services shows minimal cost and time impact, confirming HyEcoSec's suitability for secure environments. Furthermore, the dynamic re-scheduler enhances execution efficiency, reducing makespan by 61% and cost by 49% in failure scenarios.

*Article information:*

**Keywords:** Cloud Computing, Scientific Workflow, Scheduling, Cost, Makespan, Budget, Deadline, Security

**Article history:**
Received: May 20, 2025
Revised: July 4, 2025
Accepted: July 29, 2025
Published: August 9, 2025 (Online)

**DOI:** 10.37936/ecti-cit.2025193.261598

## 1. INTRODUCTION

Cloud Computing enables users to rent computing resources via the Internet on a pay-per-use basis. It offers scalability, rapid elasticity, and extensive network access [1]. These valuable characteristics make the cloud environment well-suited for hosting and running scientific workflows, which are complex sequences of computational tasks [2]. However, workflow scheduling in the cloud remains challenging due to the dynamic nature of the cloud environment, as well as the diverse user requirements, including multiple scheduling objectives (e.g., makespan, cost, energy consumption) and quality of service constraints (e.g., deadline, budget, and security) [3]. The problem is considered NP-hard [4] due to the difficulty of responding to the different scheduling objectives and quality of service constraints. In this area, numerous scheduling strategies have been developed, including static and dynamic approaches [5]. Static scheduling provides efficient execution plans before workflow execution, ensuring optimized resource allocation without runtime delays. However, it may suffer from inaccuracies due to missing accurate runtime information. Conversely, dynamic scheduling adapts to real-time conditions but may introduce scheduling overhead and runtime delays [6]. To address these inconsistencies, we propose HyEcoSec, a hybrid static-dynamic workflow scheduling approach that prioritizes security while optimizing makespan and cost. It consists of:

- Security Compliance Module: Ensures privacy and security through task classification, prioritization, and protection submodules.
- Scheduling Module: Consists of two submodules:

[1,2,3]The authors are with Oran 1 Ahmed Ben Bella University, Oran Computer Science Laboratory, Oran, Algeria, E-mail: s.hammouti95@gmail.com, byagoubi31@gmail.com, sidahmed.makhlouf@gmail.com
[3]Corresponding author: sidahmed.makhlouf@gmail.com

1. Static Scheduler (Multi-Portions Slime Mould Algorithm (MPSMA)): An improved version of Slime Mould Algorithm (SMA) leveraging a Pareto-based multi-objective approach. It provides an initial execution plan, balancing makespan and cost, and satisfies the user's deadline, budget, and security requirements.
2. Dynamic Re-scheduler: Handles unexpected failures during workflow execution.

We evaluate the efficiency of HyEcoSec using the WorkflowSim simulation tool [7], comparing MPSMA against Nondominated Sorting Genetic Algorithm II (NSGA-II), Multi-Objective Particle Swarm Optimization (MOPSO), and Multi-Objective Ant Colony Optimization (MOACO) [8–10]. Results show that MPSMA outperforms competitors, improving makespan by 26%, cost by 43%, and resource utilization by 22%. Additionally, the dynamic re-scheduler enhances system resilience, reducing makespan by 61% and cost by 49% in the event of failure scenarios. The following is a summary of the primary contributions of this work:

1. Introducing HyEcoSec, a Hybrid Cloud Workflow Scheduling (HCWS) approach that optimizes workflow execution time and cost and satisfies deadline, budget, and security needs.
2. The HyEcoSec approach prioritizes user security requirements and protects sensitive tasks and data.
3. The HyEcoSec approach merges static and dynamic strategies to maximize its benefits.
4. The components of the HyEcoSec approach are presented, explained in depth, and simulated using the WorkflowSim simulator.
5. Extensive simulations and experimental analysis are conducted using WorkflowSim to demonstrate the effectiveness of our suggested approach.

The remaining sections are organized as follows. Section 2 describes the related work, Section 3 defines the system model and main assumptions, Section 4 describes the proposed system, Section 5 outlines the performance evaluation and extensive simulations, and Section 6 concludes the paper.

## 2. RELATED WORK

Cloud Computing is an appealing choice for scientists to carry out their workflows. It provides powerful features, high performance, and significant processing power [3]. With the extensive usage of this technology, users continually seek the simplest solutions to achieve their goals at the lowest possible cost and time. Thus, scheduling scientific workflows in a cloud environment has become one of the most challenging problems facing researchers. The problem is NP-hard, multi-objective, and multi-constrained [5].

According to previous studies [5, 11], most workflow scheduling strategies focus on minimizing execution cost and makespan while adhering to deadline, budget, security, and quality of service constraints, along with optimizing energy consumption, resource utilization, and system reliability. Notable studies include [12], which proposed a scheduling algorithm to minimize cost within deadline constraints in a hybrid cloud, and [13], which introduced single-objective and multi-objective optimization algorithms for cost and makespan minimization. Similarly, [14] developed a static immune-based particle swarm optimization algorithm to optimize execution cost and makespan under a user-defined deadline, while [15] proposed a tri-objective hybrid meta-heuristic to enhance efficiency by minimizing makespan, cost, and energy consumption. Security-aware workflow scheduling has also been widely explored [11, 16], given its importance for workflows that handle confidential data or sensitive tasks. For instance, Zeng *et al.* [17] integrated static and dynamic scheduling strategies to reduce makespan under budget and security constraints, while Li *et al.* [2] proposed a static cost- and security-aware scheduling algorithm that minimizes cost while meeting deadline and risk constraints.

Cloud workflow scheduling is conducted using heuristic, meta-heuristic, and hybrid algorithms, employing either a static strategy with a predetermined execution plan or a dynamic strategy that adjusts at runtime [3, 5]. Heuristic algorithms provide approximate solutions with lower time complexity [18]. The most common heuristics used to address this issue include Heterogeneous Earliest Finish Time (HEFT), Min-Min, Max-Min, and Random [19]. Beyond these, several other heuristic-based approaches have been explored. For example, [20] proposed a general framework and a static multi-objective list scheduling algorithm to find a dominant solution using the Pareto approach. In addition, [21] introduced a security-aware static heuristic model. However, heuristics often suffer from the local optima problem with large search spaces [22], prompting researchers to adopt meta-heuristic algorithms, which typically offer better optimization performance [3]. For instance, [23] proposed a static Particle Swarm Optimisation (PSO)-based scheduling algorithm to minimize the execution cost, and [24] introduced a dynamic objective Genetic Algorithm (GA) to optimize either makespan or execution cost under a deadline. In addition, Ant Colony Optimisation (ACO) [25] was also applied for static cost and makespan optimization. To increase efficiency, researchers employ hybrid approaches to leverage the strengths of multiple algorithms at the same time. For example, [26] merged GA, Best-Fit, and Round Robin algorithms to optimize makespan and balance the load. Furthermore, [27] and [28] presented various hybrid approaches for static scheduling.

Moreover, in the context of hybrid cloud environments, workflow scheduling has been addressed through various approaches targeting different optimization objectives [5]. For instance, [29] introduced a static mixed-integer linear programming model to minimize the monetary cost of workflow execution while satisfying task security requirements and meeting a deadline, while [30] proposed a dynamic deep reinforcement learning approach for scheduling real-time jobs in a hybrid cloud context to optimize cost, quality of service, and response time. However, this approach cannot handle complex workflows involving interdependent tasks. Furthermore, [31] addressed the multi-workflow scheduling problem in a hybrid cloud by introducing a dynamic tri-objective optimization approach that integrates a multi-objective salp swarm algorithm and an iterative greedy algorithm.

Most existing studies focus on static scheduling strategies while overlooking dynamic and hybrid approaches, which can handle unexpected runtime events [6]. To fill this gap, we propose HyEcoSec, a scheduling approach that combines static efficiency with dynamic adaptability, while optimizing makespan, cost, resource utilization, and meeting the user's deadline, budget, and security requirements.

## 3. SYSTEM MODEL AND ASSUMPTIONS

This section outlines the models and key assumptions underlying our HCWS approach.

### 3.1 Workflow Model

A scientific workflow is a collection of computational tasks that exhibit inter-dependencies, primarily involving data. It is typically represented as a Directed Acyclic Graph (DAG), $W(T, E, D)$, where:

- $T = \{t_0, t_1, \ldots, t_n\}$ is the set of workflow tasks, and n is the number of tasks.
- $E$ is the set of edges representing the task's precedence constraints, with $E \subseteq T \times T \times \mathbb{R}$. For example, the edge $e(i, j) = (t_i, t_j, sd_{ij})$ indicates that the task $t_j$ must start its execution after the completion of task $t_i$, and the weight $sd_{ij}$ is the size of data transferred from task $t_i$ to task $t_j$. In this case, task $t_i$ is referred to as the predecessor of task $t_j$, while task $t_j$ is referred to as the successor of task $t_i$.
- $D = \{d_0, d_1, \ldots, d_m\}$ is the set of workflow datasets. Each task can be exploited by multiple datasets, and a dataset can be exploited by multiple tasks.
- The notations pred($t_i$) and succ($t_i$) denote the list of predecessors and successors of the task $t_i$. The workflow entry task $t_{entry}$ has no predecessors, and the exit task $t_{exit}$ has no successors.
- The notations $I(t_i)$ and $O(t_i)$ denote the list of input and output datasets of the task $t_i$, respectively.
- A task cannot start until all its predecessors finish and its input files are available.

### 3.2 Hybrid Cloud Model

A hybrid cloud integrates private and public cloud resources. In the present paper, we focus exclusively on Virtual Machines (VMs) as cloud resources, as they are the primary components responsible for executing workflow tasks. We assume that a VM can run only one task at a time. A private cloud has a fixed number of VMs with zero monetary cost (since the cloud customer owns it), higher security, but limited scalability, which can lead to increased execution time. In contrast, public cloud offers scalable VMs, but lower security, and a variable hourly cost model based on performance, where the better the machine performs, the higher the price. Private cloud VMs are denoted by: $VM^{prv} = \{vm_0^{prv}, vm_1^{prv}, \ldots, vm_p^{prv}\}$, where $p$ is the number of available VMs in the private cloud. Public cloud VMs are denoted by: $VM^{pub} = \{vm_0^{pub}, vm_1^{pub}, \ldots, vm_q^{pub}\}$, where $q$ is the number of VMs allocated from the public cloud.

### 3.3 Security Overhead Model

To ensure data privacy and meet user security requirements, we assume that each task or dataset in the workflow may require the following services: Confidentiality (C), to encrypt data and prevent unauthorized access; Integrity (I), to protect against unauthorized modifications; and Authentication (A), to verify user or system identities. Following the NIST data categorization model [32], we consider that cloud customers define the required security levels for each dataset based on its sensitivity. These levels are classified as Low (L), Moderate (M), or High (H) for each service, and are expressed as: $DSL(d_i) = \{dsl_i^C, dsl_i^I, dsl_i^A\}$. For example, $DSL(d_i) = \{H, M, M\}$ means that the dataset $d_i$ requires high confidentiality, moderate integrity, and moderate authentication. The security requirements of a task, denoted $SL(t_i) = \{sl_i^C, sl_i^I, sl_i^A\}$, are derived from the datasets it processes, as detailed in Section 4.1.1. To secure the workflow during execution, we ensure that each task is provided with adequate security services, defined by $PSL(t_i) = \{psl_i^C, psl_i^I, psl_i^A\}$, such that $SL(t_i) \leq PSL(t_i)$ for all $t_i \in T$. However, meeting these requirements introduces computational overhead, potentially increasing execution time. The security overhead $SO(t_i)$ of a task is computed using Equation (1), where confidentiality and integrity overheads depend on data size $(d_i)$. In contrast, authentication overhead depends solely on security level, as shown in Equation (2). The function $F^L$ determines the appropriate security overhead per unit of time based on the task's security level. As accurately measuring this overhead is challenging [33] and beyond the scope of this study, we estimate it using empirical values reported in previous works [2, 33, 34].

$$SO(t_i) = \sum_{L\in\{C,I,A\}} SO^L(t_i) \tag{1}$$

$$SO^L(t_i) = \begin{cases} F^L(sl_i^L, d_i) & if\ L\ \in \{C, L\}; \\ F^L(sl_i^L) & if\ L\ \in \{A\}; \end{cases} \tag{2}$$

Since the private cloud offers a more secure environment than the public cloud, we prioritize sensitive tasks for execution in the private cloud by assigning a priority rank to each task $t_i$ based on its security requirements. This rank, denoted as $Pr(t_i)$, ranges from low priority (tasks with minimal security needs, posing low risk in the public cloud) to high priority (tasks with high security needs, making execution in the public cloud very risky). The calculation of $Pr(t_i)$ is detailed in Section 4.1.1.

## 3.4 Makespan and Monetary Cost Models

The makespan $(M)$ is the elapsed time between the start of the execution of the first task in the workflow, $ST(t_{entry}, vm_k)$, until the completion of the last task in the workflow, $FT(t_{exit}, vm_m)$ (See Equation (3)), where $vm_k$ and $vm_m$ are the virtual machines executing the entry and exit tasks, respectively.

$$M = FT(t_{exit}, vm_m) - ST(t_{entry}, vm_k) \tag{3}$$

The makespan calculation depends on the following parameters:

- **Start Time** $(\boldsymbol{ST}(t_i, vm_j))$: Time at which task $t_i$ begins execution on $vm_j$, it is computed using Equation (4).

$$ST(t_i, vm_j) = \left\{ \begin{array}{ll} WT(vm_j) & , if(pred(t_i)) = \varnothing \\ \max(WT(vm_j), \max\limits_{t_k\in pred(t_i)} (FT(t_k, vm_l)) & , otherwise \end{array} \right. \tag{4}$$

where the work time $WT(vm_j)$ represents the time at which $vm_j$ becomes available for new task execution. Initially, all cloud VMs are idle, with $WT(vm_j) = 0$. After executing task $t_i$ on $vm_j$, the work time has to be updated as: $WT(vm_j) = FT(t_i, vm_j)$. Thus, the final work time of a VM is computed as:

$$WT(vm_j) = \max_{\forall t_i\ assigned\ to\ vm_j} (FT(t_i, vm_j))$$

- **Execution Time** $(ET(t_i, vm_j))$: The estimated time required to execute $t_i$ on $vm_j$ is computed using Equation (5).

$$ET(t_i, vm_j) = (length(t_i))/(mips(vm_j)) \tag{5}$$

where length$(t_i)$ denotes the task size (in instructions), and mips$(vm_j)$ is the VM's processing capacity (in millions of instructions per second (MIPS)).

- **Data Transfer Time** $(TT(t_i, vm_j))$: Time to transfer input data, as given in Equation (6).

$$TT(t_i, vm_j) = (SI(t_i))/(bw(vm_j)) \tag{6}$$

where SI$(t_i)$ is the size of the input data for task $t_i$, and $bw(vm_j)$ is the available bandwidth of $vm_j$.

- **Finish Time**$(FT(t_i, vm_j))$: The estimated time when task $t_i$ completes execution on VM $vm_j$, as defined in Equation (7).

$$\begin{aligned} FT(t_i, vm_j) =& ST(t_i, vm_j) + ET(t_i, vm_j) \\ & + TT(t_i, vm_j) + SO(t_i) \end{aligned} \tag{7}$$

Monetary Cost $(MC)$ represents the total cost of leasing public cloud VMs to complete the execution of the entire workflow, as shown in Equation (8).

$$MC = \sum_{j=0}^{q} \Big( WT(vm_j^{pub}) * C_U * price(vm_j^{pub}) \Big) \tag{8}$$

where WT$(vm_j^{pub})$ is the work time of a public VM in a given unit (e.g., milliseconds), and $C_U$ is a conversion factor that transforms this time into hours, and price$(vm_j^{pub})$ is the hourly leasing cost.

## 3.5 Problem Formulation

Our problem consists of scheduling workflow tasks while minimizing makespan $(M)$ and monetary cost $(MC)$ under deadline, budget, and security constraints. The objective function is defined by Equation (9).

$$\begin{array}{ll} minimize: & (M, MC) \\ & M \leq D \\ subject\ to: & MC \leq B \\ & \forall t_i \in T, SL(t_i) \leq PSL(t_i) \end{array} \tag{9}$$

where $D$ is the deadline constraint representing the maximum amount of time a user can wait to complete the execution of their workflow, $B$ is the budget constraint indicating the price a user is willing to pay for running their workflow, and $PSL(t_i)$ is the provided security level (ensuring it meets or exceeds required security levels $SL(t_i)$).

## 4. PROPOSED APPROACH

In this paper, we present a novel workflow scheduling approach called **Hy**brid Cloud **Eco**nomic and **Sec**ure Workflow Scheduling System (HyEcoSec), which is developed by enhancing our prior systems [35, 36]. It consists of two main components: a security compliance module and a scheduler module, as shown in Figure 1. The approach entails a secure and cost-effective distribution of tasks across hybrid cloud resources to provide clients with a safe schedule, minimized makespan, and reduced monetary cost, while adhering to specified deadline, budget, and security constraints.

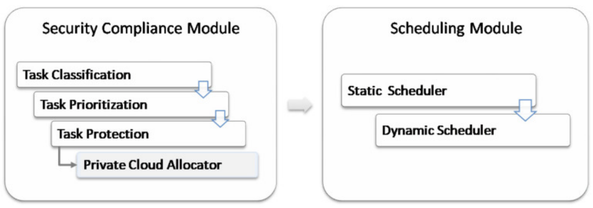

***Fig.1:*** *HyEcoSec Components.*

## 4.1 Security Compliance Module

The security compliance module ensures the privacy aspect of our HyEcoSec scheduling approach and satisfies the cloud customer's security requirements. It has three main tasks, including:

### 4.1.1 Task Classification

To secure sensitive data in scientific workflows, applying high-level security measures to all data and tasks can be effective but costly due to resource-intensive issues and the time-consuming nature of most security procedures. Therefore, according to the announcements made by the NIST [32] and Amazon Web Services best practices [37], one of the essential practices that needs to be followed to ensure appropriate levels of security for sensitive or critical data is data classification, which must be performed before establishing any protection mechanism. Thus, we assume that customers' data is inherently invisible, and it is the customer's responsibility to access and classify their data and thereby generate the tuple $DSL(d_i)$ for each dataset $d_i$. Meanwhile, the task classification module generates the tuple $SL(t_i) = \{sl_i^C, sl_i^I, sl_i^A\}$ for each task $t_i$ based on the level of sensitivity of the dataset it handles. We suppose that each task $t_i$ inherits the highest level of confidentiality and integrity services among its inputs and outputs, following the hierarchy $(L < M < H)$ as expressed in Equation (10). For authentication, if any input or output of the task $t_i$ needs high confidentiality, it is classified as highly sensitive and requires high-level authentication services. Therefore, each task adopts the maximum security level required by its inputs and outputs for both authentication and confidentiality, as shown in Equation (11).

$$For\ k\ \in \{C, I\}, sl_i^k = \begin{cases} H & , if\ \exists d_j \in \{I(t_i), O(t_i)\} \mid dsl_j^k = H \\ M\ , & if\ \nexists\ d_j \in \{I(t_i), O(t_i)\} \mid dsl_j^k = H \\ & and\ \exists d_j \in \{I(t_i), O(t_i)\} \mid dsl_j^k = M \\ L & , otherwise \end{cases} \tag{10}$$

$$sl_i^A = \begin{cases} H & , if\ \exists d_j \in \{I(t_i), O(t_i)\} \mid dsl_j^C = H \vee dsl_j^A = H \\ M\ , & if\ \nexists\ d_j \in \{I(t_i), O(t_i)\} \mid dsl_j^C = H \wedge dsl_j^A = H, \\ & and\ \exists d_j \in \{I(t_i), O(t_i)\} \mid dsl_j^C = M \vee dsl_j^A = M \\ L & , otherwise \end{cases} \tag{11}$$

### 4.1.2 Task Prioritization

Each task is assigned a priority rank, $Pr(t_i)$, based on its security requirements. Tasks with higher security demands are prioritized for execution on the private cloud, while lower-priority tasks may be assigned to the public cloud with reduced risk. The ranking function follows a hierarchical security classification: $Pr(t_i) = rank_{S_{sorted}}(sl_i^C + sl_i^I + sl_i^A)$, where assuming that $H = 3M$ and $M = 2L$, prioritizing tasks with High (H) security level over Medium (M) and Low (L), and discouraging their assignment to the public cloud. The sorted set $S_{sorted}$ contains unique values of all possible sums of $sl_i^C$, $sl_i^I$, and $sl_i^A$, sorted in ascending order. The function rank gives the priority rank based on the position of the corresponding sum $(sl_i^C + sl_i^I + sl_i^A)$ in the $S_{sorted}$ set. The priority ranking matrix (PRM) (Table 1) precomputes $Pr(t_i)$ for all security level combinations, allowing direct lookup: $Pr(t_i) = PRM[sl_i^C][sl_i^I][sl_i^A]$. For instance, tasks $t_1$, $t_2$, and $t_3$ with $SL(t_1) = \{H, H, H\}$, $SL(t_2) = \{M, H, L\}$, and $SL(t_3) = \{L, L, M\}$ have priorities 10, 6, and 2.

***Table 1:*** *Priority Rank Matrix (PRM).*

| $Sl_i^I = H$ | | $Sl_i^C$ | | |
|---|---|---|---|---|
| | | H | M | L |
| $Sl_i^A$ | H | 10 | 09 | 08 |
| | M | 09 | 07 | 06 |
| | L | 08 | 06 | 05 |

| $Sl_i^I = M$ | | $Sl_i^C$ | | |
|---|---|---|---|---|
| | | H | M | L |
| $Sl_i^A$ | H | 09 | 07 | 06 |
| | M | 07 | 04 | 03 |
| | L | 06 | 03 | 02 |

| $Sl_i^I = L$ | | $Sl_i^C$ | | |
|---|---|---|---|---|
| | | H | M | L |
| $Sl_i^A$ | H | 08 | 06 | 05 |
| | M | 06 | 03 | 02 |
| | L | 05 | 02 | 01 |

01 03 04 05 06 07 08 10

Low $\longrightarrow$ High

Priority order of task assignment to the private cloud

#### 4.1.3 Task protection

This module incorporates proactive measures and required security services at rest and in transit, thereby ensuring compliance with the constraint $SL(t_i) \leq PSL(t_i)$, as shown in Figure 2:

- At rest: Encrypting data before storage using suitable confidentiality services to protect it from unauthorized access and potential system breaches.
- In transit: Encrypting dataflows between tasks using the highest security level required by either the sender or receiver tasks, ensuring protection against intercepted communications across private and public clouds. This involves: (1) authenticating access before task initiation, (2) decrypting data for use while enforcing integrity checks during execution, and (3) encrypting data at the end of each task before transmission, as shown in Figure 2.

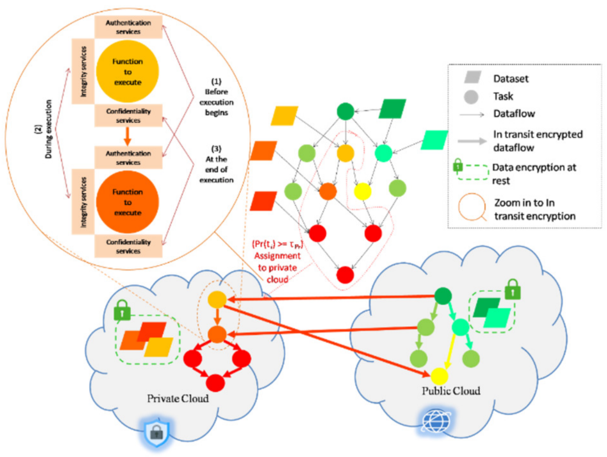

***Fig.2:*** *Protective Actions Offered by the Task Protection Module*[1].

Given that the private cloud offers higher security and incurs no additional costs, we prioritize maximizing workload allocation to it. However, this may increase execution time, potentially violating the deadline constraint. To address this, we introduce the Private Cloud Allocator (PCA) module, which balances privacy requirements with deadline adherence. PCA strategically assigns tasks to the private cloud based on priority and security needs while ensuring the makespan remains within the deadline. The PCA pseudo-code is presented in Algorithm 1. It employs a modified HEFT algorithm (Algorithm 2) to determine the priority threshold $\tau_{Pr}$, which classifies tasks for private or public cloud execution. Initially, $\tau_{Pr}$ is set to the lowest priority (01). HEFT schedules tasks by assuming all with priorities greater than or equal to $\tau_{Pr}$ run on the private cloud. If the resulting makespan meets the deadline, the algorithm returns $\tau_{Pr}$; otherwise, it increments $\tau_{Pr}$ until reaching the

[1]Note: data and tasks are color-coded based on their priorities, as defined in the PRM matrix (Table 1)

***Algorithm 1.*** *Private Cloud Allocator*

**Input:** $W = (T, E, D)$; $VM^{prv}$; $VM^{pub}$; $D$:deadline;
**Output:** $\tau_{Pr}$: Priority rank threshold

```
1. $\tau_{Pr} \leftarrow 1$;
2. while $\tau_{Pr} \leq 10$ do
3.     $M \leftarrow M - HEFT(W, VM^{prv}, VM^{pub}, \tau_{Pr})$;
4.     if ($M \leq D$) then return $\tau_{Pr}$;
5.     $\tau_{Pr} \leftarrow \tau_{Pr} + 1$;
6. end while
7. return $\tau_{Pr}$;
```

***Algorithm 2.*** *M-HEFT*

**Input:** $W = (T, E, D)$; $VM^{prv}, VM^{pub}$; $\tau_{Pr}$;
**Output:** $M$: Makespan;

```
1. Compute and sort task ranks in decreasing order using
   Equation (12);
2. for all $t_i$ in decreasing rank do
3.     if $Pr(t_i) < \tau_{Pr}$ then $L \leftarrow VM^{pub}$;
4.     else $L \leftarrow VM^{prv}$;
5.     Assign $t_i$ to $vm_k \in L$ with $min(FT(t_i, vm_k))$
       (Equation (7));
6. end for
7. Generate the schedule and calculate $M$ Equation (3);
8. return $M$;
```

highest priority (10). HEFT ranks tasks based on execution and data transfer times, as expressed in Equation (12), where AvgEt($t_i$) is the mean execution time across all VMs (private and public) (Equation (13)), and AvgTT($t_i$,$t_j$) is the average transfer time (Equation (14)):

$$rank(t_i) = \mathrm{AvgEt}(t_i) + \max_{\forall t_j \in \mathrm{succ}(t_i)}(\mathrm{AvgTT}(t_i, t_j),\ rank(t_j)) \quad (12)$$

$$\mathrm{AvgEt}(t_i) = \frac{\sum_{j=0}^{p} \mathrm{ET}(t_i, vm_j^{prv}) + \sum_{j=0}^{q} \mathrm{ET}(t_i, vm_j^{pub})}{p+q} \quad (13)$$

$$\mathrm{AvgTT}(t_i, t_j) = \frac{ds_{ij}}{\mathrm{AvgBw}} \quad (14)$$

$ds_{ij}$ is the data size transferred from $t_i$ to $t_j$, and AvgBw is the average bandwidth. M-HEFT assigns each task to the VM that minimizes the earliest finish time ($FT$), choosing from private or public cloud based on its priority rank ($Pr(t_i)$).

### 4.2 Scheduling Module

The scheduling module employs a hybrid strategy consisting of a static scheduler and a dynamic re-scheduler. The static scheduler generates an initial execution plan before workflow execution, minimizing makespan and cost while satisfying the user's deadline, budget, and security constraints. However, as it relies on predicted data, it cannot handle unexpected failures. Conversely, the dynamic re-scheduler adapts in real time to manage unforeseen failures but cannot optimize execution plans. To address these limitations, our hybrid strategy leverages both a static scheduler for initial planning and a dynamic re-scheduler for runtime adjustments.

### 4.2.1 Static Scheduler

This module provides an initial execution plan by mapping tasks to virtual machines while minimizing makespan and cost. Our analysis of scheduling methods indicates that bio-inspired algorithms yield strong performance in workflow scheduling. In particular, the Slime Mould Algorithm (SMA) [38] has shown promising results in solving complex optimization problems. However, as noted in prior studies [39, 40], the basic SMA tends to favor exploitation over exploration, making it susceptible to premature convergence and entrapment in local optima. These weaknesses are particularly problematic in complex, multi-objective environments such as hybrid cloud scheduling. Moreover, the original SMA was designed for single-objective problems and lacks the mechanisms for handling trade-offs between conflicting objectives like makespan and cost. To address these challenges, we introduce the MPSMA algorithm, an enhanced version of the SMA algorithm, which improves the exploration–exploitation balance by dividing the population into three fitness-based portions. In addition, MPSMA integrates a Pareto-based method to handle trade-offs between makespan and cost while satisfying deadline and budget constraints.

**The Slime Mould Algorithm (SMA)**, introduced by Li *et al.* [38], is a meta-heuristic optimization algorithm inspired by the foraging behaviour of Physarum polycephalum. These organisms efficiently locate food by generating propagation waves that influence cytoplasmic flow, forming intricate networks of protoplasmic tubes. SMA emulates this adaptive search mechanism to explore and refine solutions iteratively. The algorithm begins by generating an initial population of potential solutions, representing the slime's exploratory expansion. It then alternates between investigating unexplored solution domains and improving existing ones based on fitness values. The pseudo-code for the SMA is presented in Algorithm 3. First, parameters such as the maximum number of iterations $t_{max}$, population size popsize, and the positions $X_i$ of each search portion (individual) must be initialized, with $i = 1, 2, \ldots, popsize$. The population size refers to the number of slime's search portions or the number of veins. At each iteration, the algorithm computes the fitness of each search agent, denoted as $S_i$. The fitness values are then sorted and indexed in the SmellIndex list using Equation (16). Subsequently, the global best fitness bestFitness and its respective positions $X_b$ are updated. Afterward, the weight of slime, which models its adaptive search behaviour, denoted as $W$, is determined by Equation (15).

$$\overrightarrow{W(Smellindex(i))} = \begin{cases} 1+\text{r}\cdot\log\left(\frac{|bF-S(i)|}{|bF-wF|}+1\right) & ,the\ first\ half\ of\ the\ population \\ 1+\text{r}\cdot\log\left(\frac{|bF-S(i)|}{|bF-wF|}+1\right) & the\ second\ half\ of\ the\ population \end{cases} \tag{15}$$

$$SmellIndex = Sort(S) \tag{16}$$

where $bF$ and $wF$ indicate the local best and worst fitnesses found at the current iteration. They correspond to the first and last elements in the $SmellIndex$ list, respectively, and $r$ is a random value in the range of [0,1]. The weight formula's condition is similar to that of the original work, where it corresponds to the first and second halves of the population. Finally, an update of the slime's positions is performed using Equation (17).

$$\overrightarrow{X(t+1)} = \begin{cases} r_1\cdot(ub-lb)+lb & ,if\ r_1 < z \\ \overrightarrow{X_b(t)}+\overrightarrow{vb}\cdot(\overrightarrow{W}\cdot\overrightarrow{X_A(t)}-\overrightarrow{X_B(t)}) & ,if\ r_2 < p \\ \overrightarrow{vc}\cdot\overrightarrow{X(t)} & ,if\ r_2 \geq p \end{cases} \tag{17}$$

where $t$ refers to the current iteration, $r_1$ and $r_2$ are random values in [0,1], $z$ is a parameter in [0,0.1] (in the original paper $z = 0.03$), ub and lb are the upper and lower search range boundaries, $\overrightarrow{X_b(t)}$ is the best individual at iteration $t$, ($\overrightarrow{X_A(t)}$ and $\overrightarrow{X_B(t)}$ denote two individuals from the population who were picked at random at iteration $t$. $\overrightarrow{vb}$ is a parameter in $[-a, a]$ where $a$ is calculated using Equation (18). $\overrightarrow{vc}$ converges to zero after fluctuating within the interval [-1,1], p is a variable represented by Equation (19).

$$\text{a} = \operatorname{arctanh}\left(-\left(\frac{t}{t_{max}}\right)+1\right) \tag{18}$$

$$p = \tanh|S(i) - bestFitness| \tag{19}$$

For further details, readers are referred to the original paper introducing the SMA algorithm.

The SMA algorithm, like most meta-heuristic optimizers, relies on two key processes: exploration and exploitation. However, an imbalance between these processes can lead to premature convergence and trap the search in local optima [39, 40]. This behaviour often results from poor exploration due to limited population diversity, which narrows the search space and limits the discovery of better solutions. Additionally, the standard SMA was originally designed for single-objective optimization and lacks built-in mechanisms for handling conflicting objectives in multi-objective scenarios.

***Algorithm 3.*** *SMA Algorithm*

```
1.  Initialize population size $popsize$, max iterations $t_{max}$,
    and positions $X_i$ for ($i = 1, \dots, popsize$);
2.  $t \leftarrow 0$;
3.  **while** $t \leq t_{max}$ **do**
4.      Compute fitness, update $bestFitness$, $X_b$;
5.      Compute weight $W$ (Equation (15)), update
        parameter $a$ (Equation (18));
6.      **for all** search portions **do**
7.          Update the parameters $p$ (Equation (19)), $\overrightarrow{vb}$,
            $\overrightarrow{vc}$;
8.          Update positions $X_i$ (Equation (17));
9.      **end for**
10.     $t \leftarrow t + 1$;
11. **end while**
12. **return** $bestFitness$;
```

To overcome these limitations, we propose an enhanced version of SMA, called MPSMA. Inspired by [41], we divide the population into three fitness-based groups: best, moderate, and poor. This strategy helps maintain diversity and balances the search process by intensifying exploration in weaker solutions while refining high-quality candidates through exploitation.

1. The best search portion consists of individuals exhibiting superior fitness, located near the food source. Our goal is to enhance their exploitation capabilities to accelerate convergence and find the global optimum efficiently. The position update follows the original SMA equation (Equation (20)), maintaining its proven exploration-exploitation balance. To further improve exploitation, we introduce the Exploitation Factor (EF) (Equation (21)), which increases over iterations, gradually shifting the search from exploration to intensive local refinement.

$$\overrightarrow{X(t+1)} = \begin{cases} r_1 \cdot (ub - lb) + lb & , if\ r_1 < z \\ \overrightarrow{X_b(t)} + EF \cdot \overrightarrow{vb} \cdot (\overrightarrow{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}) & , if\ r_2 < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)} & , if\ r_2 \geq p \end{cases} \tag{20}$$

$$EF = 1 - \frac{t^{1/\alpha}}{t_{\max}^{t/\alpha}} \tag{21}$$

where $\alpha$ is a constant that represents the precision of exploitation, $t$ refers to the current iteration, and $t_{\max}$ is the maximum number of iterations.

2. The moderate search portion comprises individuals with mixed fitness. The objective is to balance exploration and exploitation, ensuring a diverse search while guiding solutions toward the global optimum. The position update follows Equation (22), incorporating EF (with ± sign) to enhance local search diversity. Additionally, the Exploration Exploitation Balancing (EEB) parameter facilitates a dynamic transition from exploration to exploitation. Its value increases linearly with each iteration, as defined in Equation (23). In early iterations, a random number $r_1 \in [0, 1]$ is likely to exceed the current EEB value, satisfying the condition described in Equation (23). This increases the likelihood that the algorithm applies Roulette Wheel Selection (RWS), promoting exploration by favoring less fit or diverse solutions. As iterations progress and EEB approaches 1, the condition becomes less likely to hold, and the algorithm naturally shifts toward exploiting high-quality individuals. This dynamic adjustment helps maintain diversity in early stages and improves convergence in later stages.

$$\overrightarrow{X(t+1)} = \begin{cases} \begin{rcases} \overrightarrow{X_b(t)} + EF \cdot \overrightarrow{vb} \cdot \left(\overrightarrow{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right) & , if\ r_2 < 0.5 \\ \overrightarrow{X_b(t)} - EF \cdot \overrightarrow{vb} \cdot \left(\overrightarrow{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right) & , if\ r_2 \geq 0.5 \end{rcases} & , if\ r_1 < EEB \\ RWS(\overrightarrow{W}) & , if\ r_1 \geq EEB \end{cases} \tag{22}$$

$$EEB(t) = \frac{t}{t_{\max}} \tag{23}$$

The use of $EEB$ and Exploitation Factor ($EF$) in this work is inspired by previous studies on nature-inspired and meta-heuristic algorithms [42, 43]. These parameters help to balance the exploration and exploitation processes. Furthermore, we employ the RWS method to strengthen the algorithm's exploration and increase population diversity. It enables the algorithm to reach new regions, improving its ability to approach the global optimum and avoid local optima. The RWS provides a higher probability of choosing slime portions with better fitness, while still allowing the selection of certain individuals with lower fitness due to the possibility of finding some advantageous portions in nearby positions. The selection probability is given by Equation (24), where the Roulette fitness parameter, $R_f$, is computed using Equation (25):

- It is proportional to the weight $W$, as higher weights indicate proximity to the optimal solution.
- It is inversely proportional to the objective functions $f_1$ and $f_2$, since we are solving a bi-objective minimization problem. For our HCWS problem: $f_1$ and $f_2$ correspond to the estimated execution time and monetary cost of running task $t_i$ on $vm_j$, respectively, with $f_1 = ET(t_i, vm_j)$ (Equation (5)), and $f_2 = ET(t_i, vm_j) \times price(vm_j)$.

$$Pr = \frac{Rf}{\sum_{j=1}^{m} Rf} \tag{24}$$

$$Pf = \frac{W}{f_1 \cdot f_2} \tag{25}$$

3. The poor search portion consists of low-fitness individuals located far from the optimal region.

To improve exploration in this group, we apply the Opposition-Based Learning (OBL) mechanism [44, 45], which assumes that if a solution X(t) is distant from the optimum, its opposite $\bar{X}(t) = lb + ub - X(t)$ may be closer. By evaluating the fitness values of both $\overrightarrow{X(t)}$ and $\overrightarrow{\bar{X}(t)}$\$, the algorithm increases its chances of escaping local optima and locating distinct, potentially better regions of the search space. This dual evaluation enhances population diversity by introducing solutions that are spatially distant from current individuals, effectively broadening the search space and reducing the risk of premature convergence. In addition, it accelerates convergence by increasing the likelihood of identifying high-quality solutions earlier in the optimization process, as the opposite point may lie closer to the global optimum. As a result, OBL improves the initial distribution and guides the search more effectively, especially when the current population is trapped in a sub-optimal region. For this sub-portion, the position update strategy incorporating this mechanism is presented in Equation (26).

$$\overrightarrow{X(t+1)} = \begin{cases} \overrightarrow{\bar{X}(t)} & ,if\ S \prec \bar{S} \\ \overrightarrow{\bar{X}(t)} & ,if\ S \nsucc \bar{S} \end{cases} ,if\ r_1 < z \qquad \begin{cases} \overrightarrow{X_b(t)} + EF \cdot \overrightarrow{vb} \cdot (\overrightarrow{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}) & ,if\ r_2 < 0.5 \\ \overrightarrow{X_b(t)} - EF \cdot \overrightarrow{vb} \cdot (\overrightarrow{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}) & ,if\ r_2 \geq 0.5 \end{cases} ,if\ r_1 \geq z \tag{26}$$

Given our multi-objective and multi-constrained HCWS problem (Minimize: $f_1$, $f_2$), we modify the Pareto optimality approach to rank fitness values and to update the best solution at each iteration. We also adjust weight $(W)$ and parameter $(p)$ computations, as shown in Equations (27) and (28), respectively, to account for two-dimensional Euclidean distances instead of one-dimensional measures used in single-objective problems, where $bF_1$, $S_1(i)$ correspond to makespan $(M)$ and $bF_2$, $S_2(i)$ to monetary cost $(MC)$.

$$\overrightarrow{W(Smellindex(i))} = \begin{cases} 1 + r \cdot \log\left(\dfrac{\sqrt{(bF_1 - S_1(i))^2 + (bF_2 - S_2(i))^2}}{\sqrt{(bF_1 - wF_1(i))^2 + (bF_2 - wF_2(i))^2}} + 1\right) & condition \\ 1 + r \cdot \log\left(\dfrac{\sqrt{(bF_1 - S_1(i))^2 + (bF_2 - S_2(i))^2}}{\sqrt{(bF_1 - wF_1(i))^2 + (bF_2 - wF_2(i))^2}} + 1\right) & others \end{cases} \tag{27}$$

$$p = \tanh(\sqrt{(s_1(i) - BF_1)^2 + (s_2(i) - BF_2)^2}) \tag{28}$$

The MPSMA algorithm's pseudo-code is outlined in Algorithm 5. It consists of three main steps:

1. **Initialization:** Define population size $(popsize)$, maximum number of iterations $(t_{\max})$, and search portion positions $(\overrightarrow{X_l})$ In our HCWS problem, $\vec{X}$ is a matrix where each value $X_{i,j}$ represents the VM assigned to task $t_j$. Each portion of the slime produces an execution plan denoted as $\overrightarrow{X_l}$. The assignment is randomly generated using Equation (29), where the upper and lower borders, $ub$ and $lb$, represent the maximum and minimum VM IDs across the hybrid cloud, respectively. Similarly, $ub_{prv}$ and $lb_{prv]}$ define the bounds for private cloud VMs. The condition $(\Pr(t_j) >= \tau_{Pr})$ restricts the assignment of each task $t_j$ with a priority $\Pr(t_j)$ greater than or equal to the priority threshold $\tau_{Pr}$ generated by the PCA module (Algorithm 1) to a private cloud VM.

***Algorithm 4.*** *Fitness Function*

**Input**: $S = \{VM, T, M\}$
**Output**: $M$: makespan; $MC$: monetary cost

```
1. for all $t_i \in T$ do
2.     Compute start time, execution time, transfer time, security overhead, and finish time of $t_i$ on $vm_i$ (Equations. (4–7));
3.     Update the work time of $vm_i$ ;
4. end for
5. Compute M and MC (Equations (3), (8));
6. return (M, MC);
```

$$X_{i,j} = \begin{cases} r * (ub_{prv} - lb_{prv}) + lb_{prv} & ,(Pr(t_j) \geq \tau_{Pr}) \\ r * (ub - lb) + lb & ,(Pr(t_j) < \tau_{Pr}) \end{cases} \tag{29}$$

2. **Fitness Evaluation & Best Fitness Update:** The fitness $S_i$ of each execution plan $\bar{X}_l$ is calculated utilizing Algorithm 4, which computes the makespan and monetary cost of a given scheduling scheme $S = \{VM^{prv}, VM^{pub}, T, M\}$, where $VM^{prv}$, $VM^{pub}$, and $T$ list cloud VMs and workflow tasks, and $M$ maps tasks to VMs. The fitness values $S_i$ are then sorted and indexed in the $SmellIndex$ list using the Pareto Optimality method outlined in Section 5, and the global best fitness is updated.
3. **Parameters & Location Update:** Search portions are categorized as best, moderate, or poor based on fitness values. The parameters $a, EF, EEB, p, vb, vc$, and weight $W$ are then updated accordingly. The location is adjusted using Equations (20), (22), and (26), depending on the classification. To align with our HCWS problem, if the assigned VM $X_{i,j}$ corresponds to a public VM and $\Pr(t_j) \geq \tau_{Pr}$, we replace $X_{i,})$ with a random value from $[lb_{prv},\ ub_{prv}]$. Additionally, for each generated position $X_{i,j}$, we apply absolute rounding, ensuring that $id_{vm}$ takes discrete positive values, where $X_{i,j} = round(|X_{i,j}|)$.

**The Pareto method** is a widely used approach in multi-objective optimization for identifying trade-offs between conflicting objectives [46, 47]. The first step is to partition the population into Pareto-dominated and non-dominated solutions to identify the set of optimal trade-offs. A solution $s_1$ dominates another solution $s_2$ if $s_1$ is better than or equal to $s_2$ in all objectives and strictly better in at least one. Two solutions, $s_3$ and $s_4$, are non-dominated if one is better in one objective and worse in another. In our case, each solution, s, has two objectives: makespan $(m)$ and monetary cost $(mc)$. A solution $S_1 = \{M_1, MC_1\}$ dominates S = M,MC if and only if either $M_1 < M_2$ and $MC_1 \leq MC_2$) or $(M_1 \leq M_2$ and $MC_1 < MC_2)$. Conversely, $S_3 = \{M_3, MC_3\}$ and $S_4 = \{M_4, MC_4\}$, are non-dominated if $(M_3 < M_4$ and $MC_3 > MC_4)$ or $(M_3 > M_4$ and $MC_3 < MC_4)$. The resulting set of non-dominated solutions forms the Pareto Optimal Front (POF), which represents the set of compromises between conflicting objectives. From this set, we select an optimal solution based on the minimum Euclidean distance to an ideal point, defined as the intersection of the minimum achievable values of makespan and cost [47, 48]. Figure 3 illustrates the distribution of dominated and non-dominated solutions, the POF, and the selection of the optimal point

Since the problem we tackled is multi-constrained, we introduce another solution set, called Constrained Pareto Optimal Front (CPOF), which encompasses the POF solutions that also satisfy the user's deadline and budget constraints. Next, the optimal solution is selected from the CPOF set as the one with the smallest Euclidean distance to the ideal point, as shown in Figure 4. To incorporate this concept into our MPSMA algorithm, we modify the original sort function of the SMA algorithm (Equation (16)) to rank the solutions in the $SmellIndex$ list according to Pareto optimality. We assume that CPOF solutions are better than POF solutions, which are preferred over dominated solutions in terms of makespan and monetary cost. Thus, the $SmellIndex$ list will include first the CPOF solutions, then the POF solutions, and finally the dominated solutions, where the elements of each set are arranged in ascending Euclidean distance order, with the first item representing the best fitness and the last item the worst fitness, as illustrated in Figure 5.

### 4.2.2 Dynamic Re-scheduler

Dynamic schedulers adapt to real-world systems by responding to real-time changes in cloud resources and workflow tasks. However, they may fail to optimize cost and time due to their reactive nature. In contrast, static schedulers generate efficient execution plans but rely only on predicted information and do not account for unexpected events at runtime. To bridge this gap, we propose a static scheduler that aims to provide a cost and time-optimized initial execution, followed by a dynamic re-scheduler module that can deal with unexpected events while

***Algorithm 5.*** *MPSMA Algorithm*

```
1.  Initialize population size $popsize$, max iterations $t_{max}$,
      and positions $X_i$ for $(i = 1, \ldots, popsize)$;
2.  $t \leftarrow 0$;
3.  **while** $t \leq t_{max}$ **do**
4.      Compute fitness using Algorithm 4 and sort using
        Constrained Pareto (CPOF) ;
5.      Update $bestFitness$, $X_b$, and classify search
        portions;
6.      Compute weight $W$ and update parameter $a$, $EF$,
        $EEB$, using Equations (27), (18), (21), and (23);
7.      **for all** search portions **do**
8.          Update the parameters $p$ (Equation (28)), $\overrightarrow{vb}$,
            and $\overrightarrow{vc}$;
9.          **switch** $search\ portion$ **do**
10.           **case** best **do** Update $X_i$ by Equation (20);
11.           **case** moderate **do** Update $X_i$ by Equation (22);
12.           **case** poor **do** Update $X_i$ by Equation (26);
13.         **end switch**
14.     **end for**
15.     $t = t + 1$;
16. **end while**
17. **return** $bestFitness$;
```

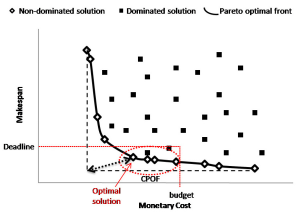

***Fig.3:*** *CPOF*

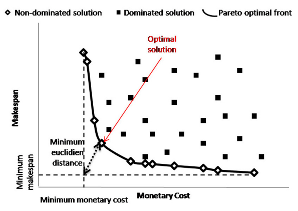

***Fig.4:*** *Pareto Method*

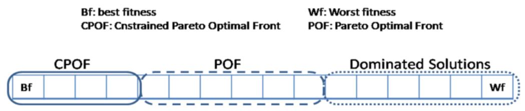

***Fig.5:*** *SmellIndex List.*

maintaining efficiency. The dynamic re-scheduler reassigns failed tasks to active VMs, assuming failures are primarily due to VM unavailability. It preserves the static scheduler's execution plan to ensure compliance with the targeted objectives and constraints while handling failures dynamically. The algorithm's pseudo-code is mentioned in Algorithm 6. It takes a task $(t)$ as input and returns the virtual machine $(vm)$ on which it will be executed. By default, the algorithm assigns task $t$ to the $VM$ initially planned by the static scheduler. If execution fails, it selects a new VM by computing the real start time $RST(t, vm_j)$ on all available VMs and choosing the one with the smallest deviation from the initially predicted start time $ST(t, vm)$. The real start time is determined as: $RST(t, vm_j) = \max(RWT(vm_j), PRFT(t))$, where $RWT(vm_j)$ is the real work time of virtual machine $vm_j$, indicating the time at which it becomes available, and PRFT(t) is the real time when the last pre-

***Algorithm 6.*** *Dynamic Re-scheduler*

**Input:** *Task t*
**Output:** *Virtual machine vm*

```
1.  $vm = VM(t)$;
2.  if $t$ is failed then
3.      if $pred(t) = \phi$ then
4.          $PRFT(t) = 0$;
5.      else
6.          $PRFT(t) = \max\left(RFT(t_i) \middle| t_i \in pred(t)\right)$;
7.      end if
8.      for all $vm_i \in VM$ do
9.          $RST(t, vm_i) = max(RWT(vm_i), PRFT(t))$;
10.     end for
11.     $vmList = VM$;
12.     if $(\Pr(t) >= \tau_{pr})$ then $vmList = VM^{prv}$;
13.     return $vm_i$ with $min(|RST(t, vmj) - ST(t, vm)|)$, where $vmj \in vmList$, $vmj$ is active and not busy;
14. else
15.     return $vm$;
16. end if
```

decessor of task t finishes its execution. Furthermore, it considers assigning task $t_i$ to the private cloud when its priority exceeds the threshold $\tau_{Pr}$ generated by the PCA module (Algorithm 1). Consequently, the dynamic re-scheduler ensures that the execution plan remains aligned with the static scheduler's initial plan, achieving efficient scheduling while dynamically handling runtime failures.

## 5. PERFORMANCE EVALUATION AND RESULTS

To assess the efficiency of HyEcoSec, we first compare our static scheduler, MPSMA, with the original SMA and three baseline algorithms for static cloud workflow scheduling: NSGA-II [8], MOPSO [9], and MOACO [10]. The comparison focuses on makespan, monetary cost, resource utilization, and adherence to deadline and budget constraints. We also assess the impact of a dynamic re-scheduler in scenarios involving system failures, which is crucial for understanding the robustness of our scheduling solution. Additionally, we examine the implications of security constraints on the proposed system, ensuring that our approach not only optimizes performance but also adheres to essential security requirements. We implemented HyEcoSec using WorkflowSim [7], an extension of CloudSim [49], and evaluated its performance across three synthetic workflow applications: LIGO (data-intensive), Montage (I/O-intensive), and Epigenomics (CPU-bound). For detailed descriptions of these workflows, refer to [50]. All results are derived from the average of 100 execution attempts to ensure accuracy.

### 5.1 Comparison with Baseline Methods & Deadline, and Budget Constraints

We evaluated MPSMA's performance against SMA and the baseline algorithms, focusing on makespan, monetary cost, resource utilization, execution time, and compliance with deadline and budget constraints. To ensure fairness, all algorithms employed the same fitness function and Pareto optimality method, with necessary adjustments to align with HyEcoSec's security constraints. Each algorithm was executed for 100 iterations with a population size of 25, adhering to original parameter settings. Evaluations were conducted under two virtual machine configurations: a small-scale setup with 20 VMs (10 public and 10 private) and a large-scale setup with 50 VMs (25 public and 25 private). Each VM was configured with:

- MIPS: randomly assigned between 100 and 1000.
- RAM: selected from {128, 256, 512, 1024, 2048} MB.
- Bandwidth: randomly assigned between 10 and 100 Mbps.
- Cost per hour: calculated based on a simplified model that increases proportionally with MIPS, RAM, and bandwidth. Such cost modelling is commonly employed in CloudSim-based simulations to approximate cloud provider billing policies [49].

These configurations reflect the diversity of VM offerings available from commercial cloud providers. They ensure a more comprehensive evaluation of the scheduling strategy under varying resource conditions, thereby enhancing generality and reducing biases associated with fixed configurations.
Deadlines ($D$) and budgets ($B$) were determined using:

$$D = L_D + k_1 * (U_D - L_D) \tag{30}$$

$$B = L_B + k_2 * (U_B - L_B) \tag{31}$$

where $L_D = M_{HEFT}$ (Makespan of HEFT), $U_D = 5 * M_{HEFT}$ and $k_1 \in [0, 1]$. $L_B$ represents the cost of assigning all tasks to the cheapest VM, $U_B$ the cost of assigning each task to the most expensive VM, and $k_2 \in [0, 1]$.

Results indicate that MPSMA consistently outperforms SMA, achieving improvements of 35% in makespan and 38% in cost (Figures 6, 7). Compared to baseline algorithms, MPSMA achieves average makespan reductions of 38% over NSGA-II, 21% over MOPSO, and 17% over MOACO, with an overall average improvement of 26% (Table 5). These results highlight MPSMA's efficiency in handling large-scale workflows. Additionally, increasing VM resources from 20 to 50 leads to reduced execution times across all algorithms, with MPSMA consistently delivering superior results, demonstrating better scalability. MPSMA also consistently meets deadline constraints, especially for medium and large workflows, whereas other algorithms struggle, particularly with the Montage workflow. In terms of monetary cost (Figure 7), MPSMA achieves reductions of 49% over NSGA-II, 47% over MOPSO, and 34% over MOACO, averaging a 43% improvement (Table 5). While other algorithms meet budget constraints in some cases, they often struggle with smaller workflows, notably Montage. Table 3 further shows that MPSMA improves deadline adherence by 32% and budget compliance by 50%, outperforming all compared algorithms. The superior performance of MPSMA is attributed to enhancements in the SMA algorithm, optimizing both exploration and exploitation processes, leading to more efficient workflow scheduling.

Additionally, we evaluated the resource utilization efficiency of MPSMA; the results indicate that MPSMA improves resource utilization by an average of 22% over the compared algorithms (Table 5). For a more precise comparison, Table 2 provides a detailed summary of the average makespan, monetary cost, and resource utilization achieved by each algorithm (MPSMA, NSGA-II, MOPSO, and MOACO) across the three workflows: LIGO, Montage, and Epigenomics. This comparison further confirms the superior performance of MPSMA across all metrics and workflows.

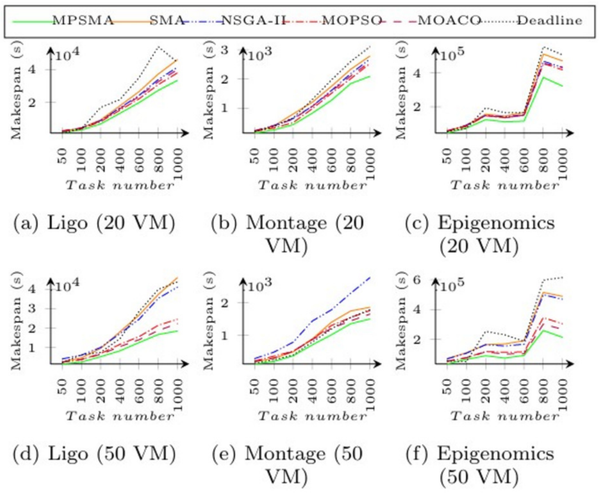

***Fig.6:*** *Makespan of MPSMA vs. Baselines & Deadline Constraint.*

Moreover, we assessed the execution time of MPSMA compared to baseline algorithms to evaluate its computational efficiency. Figure 8, which uses a logarithmic scale on the Y-axis to represent the wide variation, shows the execution time of MPSMA compared to the NSGA-II, MOPSO, and MOACO algorithms across different workflows and task sizes. Compared to MOACO and NSGA-II, MPSMA achieves significantly faster execution times across all workflows and task configurations. However, in comparison with MOPSO, MPSMA is slightly slower for small- and medium-scale workflows. This

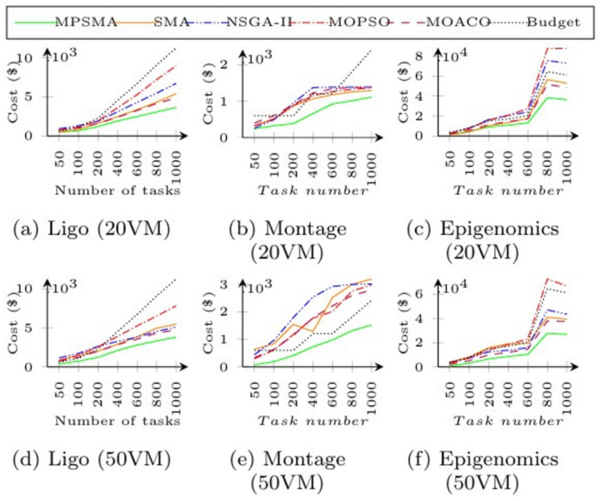

***Fig.7:*** *Monetary Cost of MPSMA vs. Baselines & Budget Constraint.*

minor performance gap is justified by the additional mechanisms integrated into MPSMA, such as the EBB parameter, OBL, and roulette wheel selection, which introduce computational overhead but effectively help prevent premature convergence and local optima, thereby yielding better results in terms of optimization objectives and adherence to constraints.

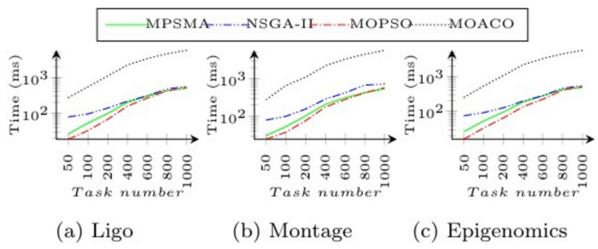

***Fig.8:*** *Execution Time of MPSMA vs. Baselines.*

Overall, MPSMA offers a favourable trade-off between execution time and solution quality. The detailed values achieved by each algorithm are illustrated in Table 4, which reports the average of 100 runs for each configuration under 20-VM and 50-VM setups.

## 5.2 Impact of Dynamic Re-scheduler

The dynamic re-scheduler heuristic proposed in this study addresses unexpected failures during real-time workflow execution. Its effectiveness was evaluated by comparing the average makespan and monetary costs across 100 runs of the HyEcoSec approach, both with and without the dynamic re-scheduler, while considering system failures. These failures were simulated using WorkflowSim's built-in fault injection framework. The failure model was configured to generate task-level failures following a normal distribution, with a fixed failure rate of 10% per workflow level. Failures were introduced across all virtual ma-

***Table 2:*** *Comparison of Average Makespan (M), Monetary Cost (MC), and Resource Utilization (RU) for MPSMA vs. Baseline Algorithms Across Workflows.*

| Algorithm | LIGO | | | Montage | | | Epigenomics | | |
|---|---|---|---|---|---|---|---|---|---|
| | M (s) | MC ($) | RU (%) | M (s) | MC ($) | RU (%) | M (s) | MC ($) | RU (%) |
| MPSMA | 10767.48 | 2006.20 | 37.81 | 809.38 | 770.04 | 26.86 | 128725.19 | 12965.18 | 34.00 |
| NSGA-II | 17853.19 | 3254.80 | 26.60 | 1265.10 | 1704.91 | 20.40 | 209712.29 | 28141.31 | 25.80 |
| MOPSO | 13521.90 | 3927.07 | 34.50 | 997.48 | 1385.22 | 25.95 | 169147.57 | 24363.64 | 29.10 |
| MOACO | 12817.48 | 2723.96 | 31.95 | 964.10 | 1488.99 | 22.70 | 158985.48 | 17890.46 | 28.05 |

***Table 3:*** *Performance Gains in Terms of Budget and Deadline Compliance.*

| | MPSMA | | NSGA-II | | MOPSO | | MOACO | |
|---|---|---|---|---|---|---|---|---|
| | Deadline | Budget | Deadline | Budget | Deadline | Budget | Deadline | Budget |
| Ligo | 41% | 52% | -4% | 6% | 24% | 6% | 24% | 26% |
| Montage | 17% | 43% | -44% | -56% | -9% | -16% | -7% | -26% |
| Epigenomics | 39% | 55% | 1% | 1% | 21% | 7% | 22% | 34% |
| Average | 32% | 50% | -16% | -16% | 12% | -1% | 13% | 11% |

***Table 4:*** *Execution Time (in milliseconds) of the Proposed MPSMA Compared to Baseline Methods.*

| | | Task number | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Workflow | Algorithm | 50 | 100 | 200 | 400 | 600 | 800 | 1000 |
| Ligo | MPSMA | 26 | 53 | 98 | 200 | 297 | 424 | 498 |
| | NSGA-II | 78 | 95 | 139 | 215 | 308 | 476 | 558 |
| | MOPSO | 19 | 33 | 67 | 159 | 263 | 423 | 525 |
| | MOACO | 271 | 545 | 1096 | 2211 | 3352 | 4576 | 5678 |
| Montage | MPSMA | 31 | 53 | 103 | 210 | 315 | 430 | 547 |
| | NSGA-II | 80 | 101 | 157 | 290 | 428 | 685 | 732 |
| | MOPSO | 25 | 38 | 77 | 178 | 284 | 429 | 584 |
| | MOACO | 278 | 665 | 1116 | 2236 | 3374 | 4560 | 5787 |
| Epigeno-Mics | MPSMA | 26 | 53 | 96 | 187 | 281 | 432 | 493 |
| | NSGA-II | 75 | 93 | 128 | 200 | 287 | 464 | 554 |
| | MOPSO | 16 | 34 | 66 | 138 | 225 | 412 | 516 |
| | MOACO | 253 | 541 | 1083 | 2207 | 3348 | 4545 | 5722 |

***Table 5:*** *Performance Gains of MPSMA Compared to Baseline Algorithms.*

| | | NSGA-II | | | MOPSO | | | MOACO | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | M | MC | RU | M | MC | RU | M | MC | RU |
| MPSMA | Ligo | 40% | 38% | 42% | 20% | 49% | 10% | 16% | 26% | 18% |
| | Montage | 36% | 55% | 32% | 19% | 44% | 3% | 16% | 48% | 17% |
| | Epigeno-mics | 39% | 54% | 37% | 24% | 47% | 17% | 19% | 28% | 21% |
| Average | | 38% | 49% | 37% | 21% | 47% | 10% | 17% | 34% | 19% |
| Average (M) = 26% Average (MC) = 43% Average (RU) = 22% | | | | | | | | | | |

chines and the 11 workflow levels used in the evaluation. These values follow WorkflowSim's default configuration, where 11 levels are used because most workflows supported by the simulator have a maximum of 11 levels. A 10% failure rate is applied for simplicity, meaning that 10% of submitted tasks at each level are randomly selected to fail. Results, illustrated in Figure 9, show a significant improvement when employing the dynamic re-scheduler, reducing makespan by 61% and cost by 49%, particularly for smaller task counts (50, 100). For medium and large workflows, the dynamic re-scheduler negatively impacted the execution costs of the Montage workflow, which is input/output bound. This can be explained by the fact that our suggested dynamic re-scheduler does not account for the VM's bandwidth.

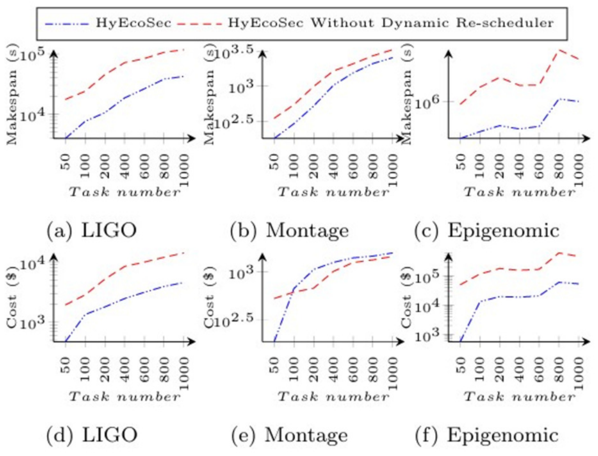

***Fig.9:*** *Impact of Dynamic Re-scheduler on Improvement of HyEcoSec Approach.*

## 5.3 Security Impact

We evaluated the impact of security constraints on workflow makespan and monetary cost by comparing our HyEcoSec approach with and without security constraints. The results, illustrated in Figure 10, indicate that the HyEcoSec technique experiences increased makespan when security constraints are applied, as shown in Figures 10a, 10b, and 10c. This is attributed to the security overhead from sensitive tasks. Conversely, Figures 10d, 10e, and 10f demonstrate that the monetary costs remain consistent regardless of the presence of security constraints. This is due to the billing model of our cloud resources, where virtual machines are charged for full hourly intervals, leading to no change in cost despite the increased makespan caused by security overhead.

## 6. CONCLUSION

This paper presents HyEcoSec, a secure, time- and cost-efficient workflow scheduling system for hybrid cloud environments. It combines a Security Compliance Module that meets user security requirements and a Scheduling Module that integrates a static

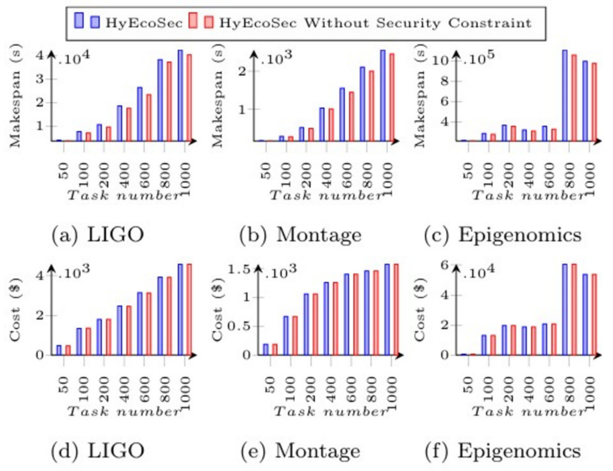

***Fig.10:*** *Impact of Security Constraint on Workflow Makespan and Monetary Cost.*

scheduler based on the MPSMA algorithm and a dynamic re-scheduler.

MPSMA efficiently balances makespan and cost while meeting deadline and budget constraints using a modified Pareto approach. Experimental results show that MPSMA outperforms SMA and baseline methods (NSGA-II, MOPSO, MOACO), achieving up to 43% cost reduction, 26% makespan improvement, and 22% better resource utilization. In terms of computational speed, MPSMA also achieves competitive execution times, significantly outperforming NSGA-II and MOACO. The dynamic re-scheduler further enhances execution in failure scenarios, reducing makespan and cost by 61% and 49%, respectively. HyEcoSec is particularly effective for data-intensive and CPU-bound workflows, with minor overhead from security integration.

However, this study has certain limitations as it was conducted in a simulated environment using synthetic workflows, which may not fully reflect the complexity and variability of real-world scenarios. In addition, the estimation of security overhead was simplified and assumed to be static. In contrast, in real-world deployments, this overhead may vary dynamically depending on task characteristics and cloud providers' security policies. Future studies can address these limitations by incorporating real workflow traces, deploying the system on actual cloud platforms, and accounting for dynamic, real-world variations in security overhead. Additionally, HyEcoSec can be extended to address energy consumption and explore additional quality of service parameters, such as load balancing and system reliability. Given its efficiency in cloud computing, HyEcoSec shows strong potential for adaptation to other environments, such as Fog computing.

## AUTHOR CONTRIBUTIONS

Conceptualization, S.H.; methodology, S.H.; software, S.H.; validation, S.H., B.Y., and S.M.; formal analysis, S.H.; investigation, S.H.; data curation, S.H.; resources, S.H.; writing—original draft preparation, S.H.; writing—review and editing, S.H. and S.M.; visualization, S.H.; supervision, B.Y. and S.M. All authors have read and agreed to the published version of the manuscript.

## References

[1] P. Mell and T. Grance, *The NIST definition of cloud computing*, 2011.

[2] Z. Li *et al.*, "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," *Future Generation Computer Systems*, vol. 65, pp. 140-152, 2016.

[3] S. Kaur, P. Bagga, R. Hans and H. Kaur, "Quality of service aware workflow scheduling in cloud computing: A systematic review," *Arabian Journal for Science and Engineering*, vol. 44, pp. 2867-2897, 2019.

[4] M. Masdari, F. Salehi, M. Jalali and M. Bidaki, "A survey of PSO-based scheduling algorithms in cloud computing," *Journal of Network and Systems Management*, vol. 25, pp. 122-158, 2017.

[5] M. Adhikari, T. Amgoth and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," ACM Computing Surveys (CSUR), vol. 52, no. 4, 2019.

[6] Y. Wang, Y. Guo, Z. Guo, W. Liu and C. Yang, "Securing the Intermediate Data of Scientific Workflows in Clouds With ACISO," in *IEEE Access*, vol. 7, pp. 126603-126617, 2019.

[7] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," *2012 IEEE 8th International Conference on E-Science, Chicago*, IL, USA, pp. 1-8, 2012.

[8] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, April 2002.

[9] C. A. Coello Coello and M. S. Lechuga, "MOPSO: a proposal for multiple objective particle swarm optimization," *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Honolulu, HI, USA, vol.2, pp. 1051-1056, 2002.

[10] D. Angus and C. Woodward, "Multiple objective ant colony optimisation," *Swarm Intelligence*, vol. 3, pp. 69-85, 2009.

[11] M. Masdari, S. ValiKardan, Z. Shahi and S. I. Azar, "Towards workflow scheduling in cloud computing: A comprehensive analysis," *Journal of Network and Computer Applications*, vol. 66, pp. 64-82, 2016.

[12] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, pp. 207-227, 2011.

[13] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *Journal of Systems Architecture*, vol. 100, p. 101631, 2019.

[14] P. Wang, Y. Lei, P. R. Agbedanu and Z. Zhang, "Makespan-Driven Workflow Scheduling in Clouds Using Immune-Based PSO Algorithm," in *IEEE Access*, vol. 8, pp. 29281-29290, 2020.

[15] A. Mohammadzadeh, M. Masdari, F. S. Gharehchopogh, and A. Jafarian, "A hybrid multi-objective metaheuristic optimization algorithm for scientific workflow scheduling," *Cluster Computing*, vol. 24, pp. 1479-1503, 2021.

[16] M. Alam, M. Shahid and S. Mustajab, "Security challenges for workflow allocation model in cloud computing environment: A comprehensive survey, framework, taxonomy, open issues, and future directions," *The Journal of Supercomputing*, vol. 80, pp. 11491-11555, 2024.

[17] L. Zeng, B. Veeravalli and X. Li, "SABA: A security-aware and budget-aware workflow scheduling strategy in clouds," Journal of Parallel and Distributed Computing, vol. 75 , pp. 141-151, 2015.

[18] R. Martí and G. Reinelt, "Heuristic methods," in *The Linear Ordering Problem*, Springer, pp. 17–40, 2011.

[19] M. M. Lopez, E. Heymann and M. A. Senar, "Analysis of Dynamic Heuristics for Workflow Scheduling on Grid Systems," *2006 Fifth International Symposium on Parallel and Distributed Computing*, Timisoara, Romania, pp. 199-207, 2006.

[20] H. M. Fard, R. Prodan, J. J. D. Barrionuevo and T. Fahringer, "A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments," *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, Ottawa, ON, Canada, pp. 300-309, 2012.

[21] F. Abazari, M. Analoui, H. Takabi, and S. Fu, "MOWS: Multi-objective workflow scheduling in cloud computing based on heuristic algorithm," *Simulation Modelling Practice and Theory*, vol. 93, pp. 119-132, 2019.

[22] D. G. Maringer, *Portfolio Management with Heuristic Optimization*, Boston, MA: Springer US, 2005.

[23] S. Pandey, L. Wu, S. M. Guru and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, Perth, WA, Australia, pp. 400-407, 2010.

[24] Z. -G. Chen *et al.*, "Deadline Constrained Cloud Computing Resources Scheduling through an Ant Colony System Approach," *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, Singapore, pp. 112-119, 2015.

[25] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia and J. Wen, "Deadline-Constrained Cost Optimization Approaches for Workflow Scheduling in Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3401-3412, 1 Dec. 2017.

[26] A. G. Delavar and Y. Aryan, "HSGA: A hybrid heuristic algorithm for workflow scheduling in cloud systems," *Cluster Computing*, vol. 17, pp. 129-137, 2014.

[27] A. Belgacem and K. Beghdad-Bey, "Multi-objective workflow scheduling in cloud computing: Trade-off between makespan and cost," *Cluster Computing*, vol. 25, pp.579-595, 2022.

[28] N. Arora and R. K. Banyal, "A particle grey wolf hybrid algorithm for workflow scheduling in cloud computing," *Wireless Personal Communications*, vol. 122, pp. 3313-3345, 2022.

[29] S. Abdi, M. Ashjaei and S. Mubeen, "Deadline-constrained security-aware workflow scheduling in hybrid cloud architecture," *Future Generation Computer Systems*, vol. 162, p. 107466, 2025.

[30] L. Cheng *et al.*, "Cost-aware real-time job scheduling for hybrid cloud using deep reinforcement learning," *Neural Computing and Applications*, vol. 34, pp. 18579-18593, 2022.

[31] Z. Sun, H. Huang, Z. Li, C. Gu, R. Xie and B. Qian, "Efficient, economical and energy saving multi-workflow scheduling in hybrid cloud," *Expert Systems with Applications*, vol. 228, p.120401, 2023.

[32] National Institute of Standards and Technology (U.S.), "Standards for security categorization of federal information and information systems," Washington, D.C., Tech. Rep. 199, 2004.

[33] H. Chen, X. Zhu, D. Qiu, L. Liu and Z. Du, "Scheduling for Workflows with Security-Sensitive Intermediate Data by Selective Tasks Duplication in Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2674-2688, 1 Sept. 2017.

[34] T. Xie and X. Qin, "Scheduling security-critical real-time applications on clusters," in *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 864-879, July 2006.

[35] S. Hammouti, B. Yagoubi and S. A. Makhlouf, "Workflow security scheduling strategy in cloud computing," in *International Symposium on*

*Modelling and Implementation of Complex Systems*, Springer, pp. 48–61, 2020.

[36] S. Hammouti, B. Yagoubi and S. Ahmed Makhlouf, "Parametric Scientific Workflow Scheduling Algorithm in Cloud Computing," *2022 International Symposium on iNnovative Informatics of Biskra (ISNIB)*, Biskra, Algeria, pp. 1-6, 2022.

[37] Amazon Web Services, "Data protection security pillar." [Online]. Available: `https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/data-protection.html`. [Accessed: Dec. 9, 2024].

[38] S. Li, H. Chen, M. Wang, A. A. Heidari and S. Mirjalili, "Slime mould algorithm: A new method for stochastic optimization," *Future Generation Computer Systems*, vol. 111, pp. 300-323, 2020.

[39] H. Chen, C. Li, M. Mafarja, A. A. Heidari, Y. Chen and Z. Cai, "Slime mould algorithm: A comprehensive review of recent variants and applications," *International Journal of Systems Science*, vol. 54, no. 1, pp. 204–235, 2023.

[40] F. S. Gharehchopogh, A. Ucan, T. Ibrikci, B. Arasteh and G. Isik, "Slime mould algorithm: A comprehensive survey of its variants and applications," *Archives of Computational Methods in Engineering*, vol. 30, no. 4, pp. 2683–2723, 2023.

[41] Y. Shen, C. Zhang, F. S. Gharehchopogh and S. Mirjalili, "An improved whale optimization algorithm based on multi-population evolution for global optimization and engineering design problems," *Expert Systems with Applications*, vol. 215, p. 119269, 2023.

[42] S. Mirjalili, S. M. Mirjalili and A. Hatamlou, "Multi-verse optimizer: A nature-inspired algorithm for global optimization," *Neural Computing and Applications*, vol. 27, pp. 495-513, 2016.

[43] L. Abualigah, A. Diabat, S. Mirjalili, M. Abd Elaziz and A. H. Gandomi, "The arithmetic optimization algorithm," *Computer Methods in Applied Mechanics and Engineering*, vol. 376, p. 113609, 2021.

[44] H. R. Tizhoosh, "Opposition-Based Learning: A New Scheme for Machine Intelligence," *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vienna, Austria, pp. 695-701, 2005.

[45] S. Mahdavi, S. Rahnamayan and K. Deb, "Opposition based learning: A literature review," *Swarm and Evolutionary Computation*, vol. 39, pp. 1-23, 2018.

[46] S. Yassa, R. Chelouah, H. Kadima and B. Granado, "Multi-objective approach for energy-aware workflow scheduling in cloud computing environments," *The Scientific World Journal*, vol. 2013, no. 350934, 2013.

[47] N. Gunantara, "A review of multi-objective optimization: Methods and its applications," *Cogent Engineering*, vol. 5, no. 1, 2018.

[48] T. Ozcelebi, "Multi-objective optimization for video streaming," Ph.D. dissertation, Graduate School of Sciences and Engineering, Koc University, 2006.

[49] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[50] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682-92, 2013.

**Sarra Hammouti** is a Ph.D. candidate in Computer Science at the University of Oran 1 Ahmed Ben Bella, Algeria. Her research interests include cloud computing, workflow scheduling, distributed systems, and multi-objective optimization. She is also a member of the Oran Computer Science Laboratory.

**Belabbas Yagoubi** holds a Ph.D. in Computer Science on 2007 and is a Full Professor at the Oran1 University Ahmed Ben Bella, Algeria. His research focuses on parallel and distributed systems, including resources management in cloud computing (security, fault tolerance, tasks scheduling, load balancing, etc.). He has held several academic and administrative roles, including Dean of the Faculty of Exact and Applied Sciences, Rector of the University of Mostaganem, and Rector of the University Center of Tindouf. He is also research team leader at the Oran Computer Science Laboratory.

**Sid Ahmed Makhlouf** holds a Ph.D in computer science on 2019. His main research interests include distributed system, cluster, grid and cloud computing, load balancing, task and workflow scheduling, and machine learning. He is currently an associate professor at the University of Oran1 Ahmed Ben Bella (Algeria) and a member of the Oran Computer Science Laboratory.