

Scheduling Latency-Sensitive Scientific Workflows on Cloud Servers and Serverless Containers

Jinquan Zhang, Xiaoping Li, *Senior Member, IEEE*, Long Chen, Zhicheng Cai, *Member, IEEE*, Rubén Ruiz, and Rajkumar Buyya, *Fellow, IEEE*

Abstract—Many latency-sensitive scientific workflows consist of numerous serial and parallel tasks with diverse workloads. Common cloud computing resources include SCs (serverless containers) and VMs (virtual machines), which have different characteristics. The large number of topological task orders resulting from complex precedence constraints, unbalanced resource requirements, and the unique characteristics of different resources makes effective workflow scheduling challenging. In this paper, we consider the scheduling problem in a hybrid configuration of servers and serverless containers to minimize the rental cost of executing workflows with deadline constraints. An algorithmic framework is proposed that comprehensively takes into account deadlines, task workloads, and dependency relationships among tasks. The framework integrates several key components, including task sequencing, determining the number of SCs and VMs, task-resource mapping with fragment minimization, and task migration across SCs for rental cost minimization. The proposed algorithm is evaluated over a comprehensive set of instances. Experimental results show that WSHC achieves the lowest mean RPD and rental cost under both homogeneous and heterogeneous resource settings.

Index Terms—Workflow scheduling, Deadline, Cost, Serverless container, Hybrid configuration.

1 INTRODUCTION

SCIENTIFIC workflows consist of many dependent tasks with different workloads (i.e., CPU instructions required for execution) and are widely used to analyze data from scientific experiments, e.g., weather forecasting, climate modeling, storm landfall prediction [1]. These scientific workflows are latency-sensitive, i.e., deadline constrained. A deadline violation could lead to serious consequences, e.g., missing the deadline for storm landfall prediction might cause serious loss of life and property. In a scientific workflow, some tasks must be executed in serial while others can be executed in parallel. In the workflow shown in Fig. 1(a), tasks t_2 and t_3 can start only after t_1 is completed. Tasks t_1 and t_2 must be executed in serial, while t_2 and t_3

can be executed in parallel. The dependency relationships and differentiated workloads among tasks result in different resource requirements at different stages of workflow execution. Satisfying the unbalanced resource requirements on traditional resource-limited clusters may result in deadline violation.

Cloud computing provides elastic resources for scientific workflow execution, including virtual machines (VMs) and serverless containers (SCs). In this paper, SCs refer to container-based serverless services [2] such as AWS Fargate. VMs and SCs follow different provisioning and billing models. VMs are provision-based resources, where users pay for instance-level CPU and memory capacity over billing time units (BTUs), regardless of actual utilization. This model can be cost-efficient for long-running and steady tasks, especially under loose deadlines where tasks can be consolidated on fewer VM instances. However, it may create idle fragments when task execution times do not align with BTU boundaries, leaving residual paid capacity unused. In contrast, SCs are task-granularity usage-based resources, where charges depend on task execution times and requested vCPU and memory resources. This finer-grained model can reduce resource waste for short-lived, bursty, or deadline-urgent tasks requiring temporary parallel capacity. Thus, the relative cost advantage of VMs and SCs depends on workload characteristics, deadline tightness, and resource configuration, motivating the hybrid VM-SC configuration studied in this paper.

The cost advantage of hybrid configurations is illustrated in Fig. 1. For example, an EC2 a1.xlarge instance (4 vCPUs, 8 GB memory) was priced at 0.102\$ per hour¹. For

- This work is supported by the National Key Research and Development Program of China (No. 2022YFB3305500), the National Natural Science Foundation of China (No. 62273089, 62363032), and the Fundamental Research Funds for the Central Universities (No.2242022R10017).
- Jinquan Zhang and Xiaoping Li are with the School of Computer Science, Guangdong University of Technology, Guangzhou, 510006, China. E-mail: zjq8860@gdut.edu.cn.
- Long Chen is with the School of Computer Science and Engineering, Southeast University, Nanjing, 211189, China, and also with the Key Laboratory of New Generation Artificial Intelligence Technology and Its Interdisciplinary Applications, Southeast University, Ministry of Education, Nanjing 211189, China. E-mail: chen_long@seu.edu.cn.
- Rubén Ruiz is at the Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain. (rruiz@eio.upv.es)
- Zhicheng Cai is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. (caizhicheng@njust.edu.cn)
- Rajkumar Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, the University of Melbourne, Australia. (rbuyya@unimelb.edu.au)

Manuscript received ; revised .

1. https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls.

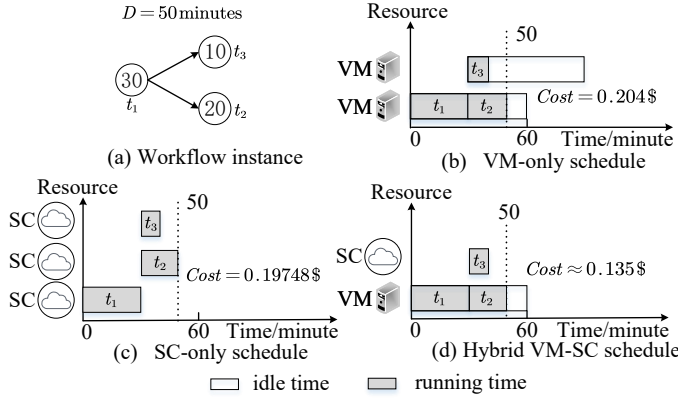


Fig. 1. Schedules of the workflow instance in (a) under different configurations: (b) VM-only; (c) SC-only; (d) hybrid.

AWS Fargate, the SC cost is computed from its usage- and resource-based pricing model, i.e., 0.04048\$ per vCPU-hour and 0.004445\$ per GB-hour. Under the selected 4-vCPU and 8-GB configuration, this results in an equivalent running cost of $4 \times 0.04048 + 8 \times 0.004445 = 0.19748\$/h$ for an SC instance². Suppose that SCs and VMs have the same processing speed, and consider a computation-intensive workflow with three tasks of 30, 20, and 10 minutes, respectively, under a deadline of 50 minutes. In the server-only case, the total execution time is 60 minutes, requiring two billing time units and incurring a cost of $2 \times 0.102 = 0.204\$$. In the SC-only case, the cost is 0.19748\$. In the hybrid configuration, one VM billing unit and one SC instance are sufficient, yielding a lower total cost of $0.102 + \frac{10}{60} \times 0.19748 \approx 0.135\$$. The corresponding Gantt charts are shown in Fig. 1(b)-(d), demonstrating that the hybrid configuration achieves the minimum rental cost. This example illustrates that the non-trivial benefit of hybridization arises from the interplay among workload characteristics, deadline tightness, and billing granularity: VMs provide cost-efficient base capacity for consolidated tasks, while SCs absorb deadline-driven temporary parallelism or fine-grained workloads without introducing additional VM BTUs.

This paper investigates the scientific workflow scheduling problem in a hybrid environment of VMs and SCs, aiming to minimize the total rental cost under hard deadline constraints. Since workflow scheduling with server-only resources is NP-hard [3], it is natural that the considered hybrid VM-SC scheduling problem is also NP-hard with the following additional challenges: i) The workflow is commonly scheduled in a topological order to meet the dependency constraints among tasks. Different topological task orders may change the dependency release order of tasks, which determines when downstream tasks become ready and thus affects the degree of parallelism and resource requirements during workflow execution. However, there is a large number of topological orders. How to determine a topological order to balance resource requirements is a challenge. ii) For a given topological task order, the cost benefit of hybridization depends on the interaction among workload characteristics, deadline tightness, and BTU

boundaries. Assigning too many tasks to VMs may increase BTU-induced idle fragments under tight deadlines, whereas assigning too many tasks to SCs may lose the cost benefit of VM consolidation under loose deadlines. Therefore, how to assign tasks between VM and SC configurations to exploit deadline-driven cost complementarity is challenging. iii) For the tasks assigned to the VM configuration, different numbers of rented VMs and different task-to-VM mappings lead to different idle fragments under the BTU-based VM billing model. Determining the number of rented VMs and mapping workflow tasks to them to reduce fragments and improve paid-capacity utilization is also complicated.

In this paper, we propose a heuristic scheduling framework, Workflow Scheduling Algorithm in Hybrid Configuration (WSHC), which aims to minimize the total rental cost by exploiting the cost complementarity between VMs and SCs under deadline constraints. Specific methods for the heuristic components in WSHC are proposed for task sequencing, determining the number of SCs and VMs, task-resource mapping with fragment minimization, and task migration from SCs to VMs for rental cost minimization. The main contributions are the following:

- 1) The WSHC algorithm framework is proposed for scheduling deadline-constrained workflows on hybrid cloud resources of VMs and SCs. Different from existing hybrid cloud resource provisioning with a fixed number of local VMs, the considered environment supports elastic VM provisioning.
- 2) Besides the total rental cost of SCs, that of on-demand VMs is also minimized, since tasks are more commonly executed by cloud VMs in practice than only by SCs and local VMs in existing studies. In addition, a shared storage is adopted to enable data exchange between SCs and on-demand VMs.
- 3) To address the unbalanced resource requirements of latency-sensitive workflows, the proposed WSHC algorithm jointly considers task distribution between VMs and SCs, task-resource mapping, and SC-to-VM task migration to improve VM paid-capacity utilization and reduce unnecessary usage-based SC executions, thereby reducing the rental cost under deadline constraints.

The rest of this paper is organized as follows: Section 2 reviews the related literature. The problem is described and formulated in Section 3. Section 4 introduces the proposed system architecture and algorithms. Experimental results are shown in Section 5, followed by conclusions and future research in Section 6.

2 RELATED WORK

Workflow scheduling has been widely studied. In the following, we briefly review related work on server-based, serverless, and hybrid resource-based workflow scheduling.

2.1 Server-Based Workflow Scheduling

For server-based workflow scheduling, deadlines are commonly treated as constraints, while monetary cost is commonly optimized as the objective. Abrishami et al. [4] proposed IC-PCP and IC-PCPD2, representing critical-path

2. <https://aws.amazon.com/fargate/pricing/>

and deadline-decomposition heuristics for cost minimization. Rodriguez et al. [5] used Particle Swarm Optimization (PSO) to search task-VM mappings after estimating the VM pool from workflow parallelism. Haidri et al. [6] proposed the Cost-Effective Deadline-Aware (CEDA) algorithm, which emphasizes rental-cost reduction under VM availability and billing-interval constraints. Faragardi et al. [7] proposed GRP-HEFT to jointly address workflow scheduling and resource provisioning in IaaS clouds.

In addition, many studies extended cloud workflow scheduling to more dynamic and heterogeneous settings. Fan et al. [8] proposed a bidirectional workflow scheduling approach with a feedback mechanism for deadline-constrained cost optimization. Cai et al. [9] investigated dynamic deadline-constrained multi-objective workflow scheduling in multi-cloud environments. Abdi et al. [10] formulated a mixed-integer linear programming model for minimizing workflow execution cost under deadline and security constraints in hybrid private-public clouds. Abdelatif [11] proposed an energy-efficient workflow task scheduler for DVFS-enabled cloud systems under deadline and budget constraints. However, these studies mainly focus on server-based resources rather than joint VM-serverless-container scheduling.

2.2 Serverless Workflow Scheduling

Serverless functions and serverless containers provide fine-grained elasticity for workflow execution, but workflow-level performance, cost, and deadline requirements still need to be modeled and optimized. Lin et al. [12] developed a performance and billing model for serverless workflows and designed greedy heuristics for performance-cost optimization. Majewski et al. [13] studied scientific workflow scheduling on serverless architectures under budget and deadline constraints. Zhang et al. [14] proposed Jolteon for stage-level resource configuration of serverless application workflows, selecting the size and number of function instances for workflow stages under user-specified cost or latency bounds.

Beyond scheduling models, serverless workflow execution is shaped by system-level factors such as execution overhead, cold starts, and container-level scheduling. Li et al. [15] proposed FaaSFlow to reduce scheduling and intermediate-data transfer overheads in serverless workflow execution. Joosen et al. [16] characterized serverless cold starts at large scale across triggers, runtimes, resource configurations, and regions. Zhu et al. [2] developed RHINO, a serverless container system for small-scale HPC applications, distinguishing serverless containers from traditional Function-as-a-Service models. Birajdar et al. [17] proposed DCRDA for function-to-container and container-to-VM scheduling under function deadlines and heterogeneous resource demands. These studies improve serverless-only workflow scheduling or execution, but they do not exploit the cost complementarity between VM paid-capacity consolidation and SC task-granularity elasticity.

2.3 Hybrid Resource-Based Workflow Scheduling

Only a limited number of studies have investigated scheduling under hybrid configurations of server-based and

TABLE 1
Comparison of representative workflow scheduling studies.

Reference	Workflow	Deadline	Cost Min	On-demand VM	Hybrid VM+SC
[4]–[6], [8], [9], [10]	✓	✓	✓	✓	×
[7]	✓	×	×	✓	×
[11]	✓	✓	×	✓	×
[12]	✓	×	✓	×	×
[13]	✓	✓	×	×	×
[14]	✓	×	✓	×	×
[15]	✓	×	×	×	×
[17]	×	✓	×	×	×
[18], [19]	×	×	✓	✓	✓
[20]	✓	×	✓	×	✓
[21]	✓	×	×	✓	✓
This work	✓	✓	✓	✓	✓

serverless resources, where heterogeneous provisioning and billing models further complicate scheduling decisions. Mahajan et al. [18] analyzed the cost and performance trade-offs between serverless and traditional cloud computing, but their formulation focuses on independent tasks rather than workflows. Mampage et al. [19] used deep reinforcement learning to schedule application workload requests in serverless and serverful hybrid environments, but did not model DAG-based workflows or hard workflow deadlines. Jiang et al. [20] combined serverless functions with local clusters to execute scientific workflows, but the local-cluster setting does not capture on-demand VM provisioning. Zhang et al. [21] proposed a hybrid VM-serverless workflow scheduling framework under a limited budget, but it does not model hard workflow deadlines.

Table 1 compares existing studies in terms of constraints, objectives, and resource characteristics. The comparison shows that existing studies usually cover only part of the problem space: VM-based and serverless-only studies do not jointly model VM-SC resources, while hybrid studies miss either DAG workflow dependencies, hard deadlines, or on-demand VM provisioning. As a result, dependency-aware and deadline-driven cost optimization over the complementary capabilities of VM paid-capacity consolidation and SC task-granularity elasticity remains underexplored.

3 PROBLEM FORMULATION

3.1 Assumptions

Since the problem under study is complex, we make the following assumptions:

- Each task is non-preemptive and assigned to only one resource instance, and each resource instance executes at most one task at a time.
- The workflow structure, task workloads, data sizes, and deadline are known before scheduling.
- Each VM or SC instance has a resource-specific start-up time, which is considered before the first assigned task can start and is included in the corresponding rental duration.

3.2 Problem Description

A workflow can be described as a directed acyclic graph (DAG) $G = (T, E)$. $T = \{t_1, \dots, t_N\}$ is the task set, and E is the edge set. Each task t_i has a workload w_i . In particular, t_1 and t_N are the source and the sink tasks, respectively. They are dummy tasks with 0 instructions. Edge $e_{ij} = (t_i, t_j)$

indicates that task t_j cannot be executed until the data d_{ij} transferred from t_i has been received. In addition, the workflow G is constrained by a deadline D . All tasks should be completed before D .

Two types of resources are considered in this paper: SCs and VMs. For any resource instance r , which can be an SC or a VM, p_r denotes its processing speed measured in instructions per second, and δ_r denotes its startup time. Different resource instances may have heterogeneous execution characteristics. The unit cost c_r is resource-specific. For an SC, it is computed according to its usage- and resource-based pricing model. For a VM, it denotes the unit rental cost under the BTU-based billing model. The billing time interval of VMs is BTU . Besides, a shared cloud object storage (COS) is used for data exchange among resource instances, with bandwidth b between the COS and each instance.

Suppose that R is the set of rented resource instances, and r_k is the k^{th} resource instance in R . The first m_{SC} resource instances in R are SCs and the remaining m_V resource instances are VMs. $m_{SC} + m_V = |R|$. Let x_{ik} be a binary assignment variable, where $x_{ik} = 1$ if task t_i is executed on resource instance r_k , and $x_{ik} = 0$ otherwise. Each task is assigned to exactly one resource instance, i.e.,

$$\sum_{k=1}^{|R|} x_{ik} = 1, \quad \forall t_i \in T. \quad (1)$$

When task t_i is processed on resource instance r_k , its candidate computation time is calculated as

$$CT(t_i, r_k) = w_i / p_{r_k}. \quad (2)$$

Thus, the actual computation time of task t_i is determined by the assignment variable x_{ik} , i.e.,

$$CT(t_i) = \sum_{k=1}^{|R|} CT(t_i, r_k) x_{ik}. \quad (3)$$

This resource-dependent formulation captures the heterogeneous execution performance of VMs and SCs.

Let $d_{j,i}$ denote the data size transferred from task t_j to t_i . For task t_i , its total input data consist of both the data received from its predecessor tasks and the additional external inputs $IP(t_i)$ stored in the shared COS. Each task reads its input data from the COS and writes its output data back after execution. If two tasks are deployed on the same VM instance, their intermediate data are reused locally and thus excluded from COS transfers. Otherwise, intermediate data are transferred through the COS. Therefore, the data input time $In(t_i)$ can be calculated via

$$In(t_i) = IP(t_i)/b + \sum_{t_j \in pr(t_i)} d_{ji}/b \cdot \mathbb{I}_{ji}, \quad (4)$$

where $pr(t_i)$ is the set of immediate predecessor tasks of t_i , i.e., those tasks t_j for which a directed edge $(t_j, t_i) \in E$ exists. $\mathbb{I}_{ji}$ is an indicator function that equals 0 when t_j and t_i are co-located on the same instance, and 1 otherwise.

$$\mathbb{I}_{ji} = \begin{cases} 0, & \text{if } \exists k \in \{1, \dots, |R|\} \text{ s.t. } x_{jk} = x_{ik} = 1, \\ 1, & \text{otherwise.} \end{cases} \quad (5)$$

For data persistence and cross-instance accessibility, each task writes its output data to the shared COS after execution. Therefore, the data output time of t_i can be calculated via

$$Out(t_i) = \sum_{t_j \in su(t_i)} d_{ij}/b, \quad (6)$$

where $su(t_i)$ is the immediate successor set of t_i . The total execution time of task t_i on resource instance r_k is the sum of computation time and data transmission time, i.e.,

$$ET(t_i) = CT(t_i) + In(t_i) + Out(t_i). \quad (7)$$

For each resource instance r_k , $Av(r_k)$ denotes its earliest available time. It is initialized to the startup time δ_{r_k} rather than 0, because a resource instance cannot execute its first assigned task until its startup process is completed. It is dynamically updated during task assignment: after a task t_j finishes on r_k , $Av(r_k)$ is set to the finish time of t_j . When scheduling a new task, $Av(r_k)$ ensures that resource conflicts are avoided and tasks start only when the assigned instance becomes idle. The start time $ST(t_i)$ and finish time $FT(t_i)$ of task t_i satisfy

$$ST(t_i) \geq \max \left\{ \max_{t_j \in pr(t_i)} \{FT(t_j)\}, \sum_{k=1}^{|R|} Av(r_k) x_{ik} \right\}, \quad (8)$$

$$FT(t_i) = ST(t_i) + ET(t_i). \quad (9)$$

Considering the deadline constraint, the following inequality should be satisfied, i.e.,

$$\max_{t_i \in T} \{FT(t_i)\} \leq D. \quad (10)$$

The objective of the considered problem is to minimize the total cost C , which can be described as follows:

$$\min C = C_{SC} + C_V, \quad (11)$$

C_{SC} is the total cost of all rented SC instances under the usage-based billing model, i.e.,

$$C_{SC} = \sum_{k=1}^{m_{SC}} c_{r_k} \times (FT(r_k) - ST(r_k)), \quad (12)$$

C_V is the total cost of all rented VM instances taking billing time intervals BTU into account, i.e.,

$$C_V = \sum_{k=m_{SC}+1}^{|R|} c_{r_k} \times BTU \times \left\lceil \frac{FT(r_k) - ST(r_k)}{BTU} \right\rceil, \quad (13)$$

where c_{r_k} is the unit cost of instance r_k , $FT(r_k)$ and $ST(r_k)$ are the finish time and start time of r_k , respectively. They can be calculated by

$$FT(r_k) = \max \{FT(t_i) \mid t_i \in T, x_{ik} = 1\}, \quad (14)$$

$$ST(r_k) = \min \{ST(t_i) \mid t_i \in T, x_{ik} = 1\} - \delta_{r_k}. \quad (15)$$

4 PROPOSED ALGORITHM

4.1 System Architecture

To support deadline-constrained workflow execution on hybrid VM-SC resources, a system architecture is designed, as illustrated in Fig. 2. The hybrid system architecture consists of a control plane and an execution plane. The control plane contains a request manager, a resource monitor, and a scheduler, while the execution plane contains the unified resource management platform, VM and SC resource pools, and the shared cloud object storage (COS).

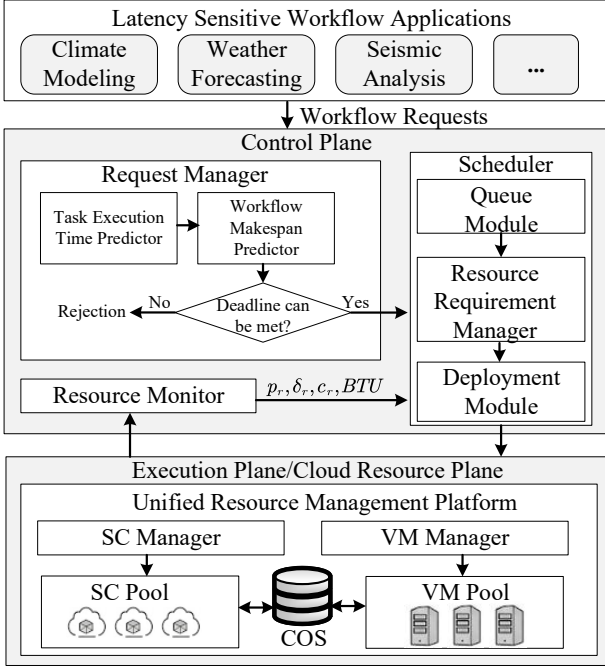


Fig. 2. Proposed VM-SC hybrid workflow scheduling architecture.

Latency-sensitive scientific workflow requests from different users are first submitted to the request manager. For each request, the request manager predicts resource-dependent task execution times using existing methods, e.g., the learning-based approach in [22], and estimates whether the workflow deadline can be satisfied. Requests whose deadlines cannot be satisfied are rejected, while the remaining ones are forwarded to the scheduler. Based on workflow information, including dependencies, deadlines, and estimated execution times, and resource characteristics provided by the resource monitor, including processing speed, startup time, billing unit, and unit cost, the scheduler generates a workflow schedule. The schedule is then submitted to the unified resource management platform to provision the required VM and SC resources. The COS is used for data exchange among VM and SC instances, as described in Section 3.

The scheduler consists of three modules: a queue module, a resource requirement manager, and a deployment module. They are responsible for task sequencing based on topological order, assigning tasks between VM and SC configurations, and mapping tasks to resource instances, respectively. To address the considered problem, a workflow scheduling framework, termed Workflow Scheduling

in Hybrid Configuration (WSHC), is proposed. WSHC minimizes the total rental cost by exploiting the dependency-aware and deadline-driven cost complementarity between VMs and SCs, rather than relying on a fixed preference for either resource type. Four algorithmic components are included in WSHC: Task Sequencing (TS), Resource Requirement Distribution (RRD), Task-Resource Mapping (TRM), and Task Migration for Improvement (TMI). The TS, RRD, and TRM components correspond to the queue module, resource requirement manager, and deployment module, respectively. In TMI, the total rental cost is further reduced by migrating selected tasks from SCs to VMs, which requires cooperation between the resource requirement manager and the deployment module. An outline of WSHC is shown in Algorithm 1. The details of each component are described in the following section.

Algorithm 1: Workflow Scheduling in a Hybrid Configuration (WSHC)

Input: $G = (T, E)$; D

Output: Total cost C

```

1 begin
2    $Q \leftarrow$  Task Sequencing (TS) ;
3   Resource Requirement Distribution (RRD) ;
4   foreach task  $t_i \in Q$  do
5     Task-Resource Mapping (TRM) ;
6   Task Migration for Improvement (TMI);
7   return  $C$  ;
```

4.2 Task Sequencing (TS)

In this section, tasks are sequenced according to their priorities. Three priority strategies are commonly used in list scheduling algorithms: S-Level (static level), T-Level (top level), and B-Level (bottom level). The principle behind S-Level and B-Level is to prioritize tasks that have a greater impact on the overall workflow completion time. Both S-Level and B-Level search for the longest path from the current task to the sink task, differing in whether communication time is considered, and schedule tasks with higher priorities first. In contrast, T-Level computes the longest path from the source task to the current task while considering communication time, and schedules tasks with smaller priority values first.

In heterogeneous VM-SC environments, the actual execution time of a task depends on the resource instance to which it is assigned. However, the concrete resource assignment is not determined in the TS stage. Therefore, we introduce a deadline-aware reference VM speed p^{ref} to estimate task execution times before resource assignment. To determine p^{ref} , we assume that each task is executed on an independent reference VM instance and that all reference VM instances have the same processing speed. Among the candidate VM processing speeds provided by the resource monitor, p^{ref} is selected as the slowest VM speed under which the workflow can satisfy the deadline in the reference execution model. This setting makes the reference execution-time estimate consistent with the workflow deadline, rather than optimistically assuming the fastest available

VM resources. Specifically, the reference computation time of task t_i is calculated as

$$\widehat{ET}(t_i) = w_i / p^{ref}. \quad (16)$$

The specific implementations of these strategies are defined as follows:

- 1) B-Level: Suppose that all tasks are allocated to different instances, the data transmission time among tasks cannot be ignored. $MET(t_i)$ denotes the estimated execution time of t_i under the reference VM speed when all input data are read from the shared COS. It is calculated as $MET(t_i) = IP(t_i)/b + \widehat{ET}(t_i) + Out(t_i) + \sum_{t_j \in pr(t_i)} d_{ji}/b$. The priority of t_i can be calculated by $prio_i = \max_{t_j \in su(t_i)} \{prio_j\} + MET(t_i)$, $prio_N = 0$.
- 2) S-Level: Suppose that some tasks may be allocated to the same instance, the communication time among these tasks is not considered. $MET(t_i)^*$ denotes the estimated execution time of t_i under the reference VM speed when the transmission time of all intermediate data is ignored. It is calculated as $MET(t_i)^* = IP(t_i)/b + \widehat{ET}(t_i) + Out(t_i)$. The priority of task t_i can be calculated by $prio_i = \max_{t_j \in su(t_i)} \{prio_j\} + MET(t_i)^*$, $prio_N = 0$.
- 3) T-Level: Similar to B-Level, the priority of t_i defined by T-Level is also calculated based on $MET(t_i)$. It is calculated as $prio_i = \max_{t_j \in pr(t_i)} \{prio_j\} + MET(t_i)$, $prio_1 = 0$.

4.3 Resource Requirement Distribution (RRD)

The Resource Requirement Distribution (RRD) component is performed before instance-level scheduling. It partitions tasks into the VM pool and the SC pool according to deadline tightness, reference execution-time estimates, and workflow dependencies. Because concrete heterogeneous instances have not yet been selected at this stage, RRD uses the deadline-aware reference VM speed introduced in Section 4.2 as a pool-level estimation basis. Let c_{vm}^{ref} denote the unit price of the reference VM that provides p^{ref} , and let c_{sc}^{ref} denote the unit price of the SC with the same or comparable CPU-memory configuration.

First, if the deadline of a workflow is too tight, distributing tasks across multiple VM instances tends to increase fragmentation [4], i.e., the unutilized portion within each BTU, because a tight deadline forces many VM instances to be launched while each task occupies only a small fraction of a BTU. Let C_{SC} be the total rental cost of the schedule in the SC pool, and C_H be the total rental cost of the schedule in the hybrid VM-SC configuration. If D falls into the interval $(0, \min_{t_i \in T} \{MET(t_i)^* / MET(t_i) \cdot c_{vm}^{ref} / c_{sc}^{ref} \cdot BTU\}]$, then $C_{SC} \leq C_H$ under the reference execution model. Therefore, RRD assigns all tasks to the SC pool in this case. The detailed proof is provided in the Appendix. Note that this conclusion is applicable to a DAG-based workflow rather than isolated tasks. It provides a reference-level analytical condition for RRD rather than a complete scheduling rule, while actual startup times are considered later in TRM and in the final cost calculation. In addition, the implication of this conclusion is too strict. A small number of tasks with small $MET(t_i)^* / MET(t_i)$ could be ignored because an approximate optimal schedule can

be obtained. Therefore, we set the interval η_1 as $\eta_1 = (0, \frac{1}{N} \sum_{t_i \in T} \{MET(t_i)^* / MET(t_i) \cdot c_{vm}^{ref} / c_{sc}^{ref} \cdot BTU\}]$. If $D \in \eta_1$, all tasks are assigned to the SC pool.

Second, tasks are assigned to resource pools according to their reference execution-time estimates. For the task t_i to be scheduled, let ΔC_i^{SC} be the cost increment if t_i is assigned to the SC pool, and let ΔC_i^{VM} be the cost increment if t_i is assigned to the VM pool. Let $ub = \lceil c_{vm}^{ref} / (c_{sc}^{ref} - c_{vm}^{ref}) \rceil$. $\Delta C_i^{SC} \geq \Delta C_i^{VM}$ if $MET(t_i)$ falls into the interval, $\eta_2 = \bigcup_{k=1}^{ub} [k \cdot BTU \cdot c_{vm}^{ref} / c_{sc}^{ref}, k \cdot BTU] \cup (ub \cdot BTU, +\infty)$, which is proved in the Appendix, under the reference execution model. The VM pool is selected for each task t_i with $MET(t_i) \in \eta_2$ to greedily reduce the cost increment. Specifically, if $MET(t_i) \in \eta_2$ for all $t_i \in T$, RRD assigns all tasks to the VM pool.

Finally, the remaining tasks are assigned to reduce the imbalance of resource requirements for VMs. After the above two steps, there are still some tasks left unassigned to any resource pool. The idle fragments on VMs result from unbalanced resource requirements for VM instances. Therefore, reducing the imbalance of resource requirements for VMs is crucial to reduce rental costs. In this section, the imbalance of resource requirements is reduced by distributing part of the remaining tasks of each level to the VM pool.

Definition 1. The level $l(t_i)$ of task t_i is the length from the source task to the current task. It is calculated by

$$l(t_i) = \begin{cases} 1, & \text{if } t_i = t_1, \\ \max_{t_j \in pr(t_i)} \{l(t_j) + 1\}, & \text{otherwise.} \end{cases} \quad (17)$$

Different levels contain different numbers of tasks. If all tasks are deployed to the VM pool, the number of tasks at different levels may cause severe imbalance in resource requirements and increase the total rental cost. Intuitively, resource requirements are more balanced if the total number of running tasks and ready tasks is stable in the workflow execution process. Therefore, we propose two heuristic rules for assigning the remaining tasks.

- 1) Average Task Number Limitation (ATNL): The average task number of different levels is used to set a limit when assigning tasks to the VM pool. L is the maximal level of the tasks, i.e., $L = \max_{t_i \in T} \{l(t_i)\}$. The average task number $\bar{n}$ can be calculated by: $\bar{n} = N / (L - 2)$. $L - 2$ is adopted as a denominator instead of L , because t_1 and t_N are dummy tasks. If the number of scheduled tasks in the $l(t_i)^{th}$ level is larger than $\bar{n}$, t_i is assigned to the SC pool, and the VM pool is chosen otherwise.
- 2) Average Workload Limitation (AWL): The average reference workload of tasks in different levels is used to set a limit when assigning tasks to the VM pool, which can be calculated as $\bar{w} = \sum_{t_i \in T} MET(t_i) / (L - 2)$. For the task t_i to be scheduled, if the total reference workload of scheduled tasks in the $l(t_i)^{th}$ level is larger than $\bar{w}$, t_i is assigned to the SC pool. Otherwise, it is assigned to the VM pool.

4.4 Task-Resource Mapping (TRM)

In TRM, tasks assigned to the VM pool and the SC pool are mapped to concrete heterogeneous resource instances.

For tasks assigned to the VM pool, TRM selects suitable VM instances to reduce BTU-induced fragments. For tasks assigned to the SC pool, TRM selects suitable SC configurations according to their resource-dependent execution times, startup times, and usage-based costs.

To avoid violating the workflow deadline, we set a group of time constraints for tasks mapped to resources. Here, $EFT(t_i)$ and $LFT(t_i)$ denote the earliest and latest finish times of task t_i , respectively. The reference execution-time estimate $MET(t_i)$, which includes both the reference computation time and the data transmission time of task t_i as described in Section 4.2, is adopted to calculate these time constraints before concrete heterogeneous instances are selected.

$$EFT(t_1) = 0, \quad (18)$$

$$EFT(t_i) = \max_{t_j \in pr(t_i)} \{EFT(t_j)\} + MET(t_i), \quad (19)$$

$$LFT(t_N) = D, \quad (20)$$

$$LFT(t_i) = \min_{t_j \in su(t_i)} \{LFT(t_j) - MET(t_j)\}. \quad (21)$$

Two types of time constraints are considered in this section:

- 1) Sub-deadline Constraints (SDC): Sub-deadlines are adopted to make the deadline more manageable. For each task t_i , its sub-deadline sub_i can be calculated by:

$$sub_i = EFT(t_i)/EFT(t_N) \cdot D, \quad (22)$$

- 2) Latest Finish Time Constraints (LFTC): The latest finish time $LFT(t_i)$ of t_i cannot be violated. Otherwise, the deadline is missed.

To uniformly describe the two time constraints, we define

$$\theta_i = \begin{cases} sub_i, & \text{under SDC,} \\ LFT(t_i), & \text{under LFTC.} \end{cases} \quad (23)$$

Thus, a candidate resource is regarded as feasible only if the finish time of t_i on this resource does not exceed θ_i .

For tasks assigned to the SC pool, TRM selects a concrete SC configuration before execution. Since SCs are charged in a task-granularity usage-based manner, SC pool tasks do not introduce BTU-induced fragments. Suppose that R_{sc} is the set of candidate SC configurations. For each task t_i assigned to the SC pool and each SC configuration $r_k \in R_{sc}$, TRM calculates $ET(t_i, r_k)$, $ST(t_i, r_k)$, and $FT(t_i, r_k)$ using the actual processing speed, startup time, and unit cost of r_k . The candidate SC set is constructed as

$$Av_{sc}(t_i) = \{r_k \mid r_k \in R_{sc}, FT(t_i, r_k) \leq \theta_i\}.$$

Among the candidate SC configurations in $Av_{sc}(t_i)$, the one with the minimum usage-based cost is selected. If multiple SC configurations incur the same cost, the one with the earliest finish time is selected. In our experimental setting, candidate VM and SC configurations are provided with comparable resource capacities, and the generated workflow instances are feasible under the available configurations.

The most suitable VM is chosen by jointly considering cost increment, task finish time, and lease remaining time (the gap between the last task's finish time and the

VM's release time). We determine it through the following steps. First, the set of available VMs is determined. Suppose that t_i is the task to be scheduled, R_{vm} is the set of currently rented VMs. For each heterogeneous VM instance $r_k \in R_{vm}$, $k = 1, 2, \dots, |R_{vm}|$, calculate $ET(t_i, r_k)$, $ST(t_i, r_k)$ and $FT(t_i, r_k)$ using the actual processing speed, startup time, and availability of r_k . When evaluating a candidate VM instance, the input transfer time of t_i is recalculated according to whether its predecessor tasks have been mapped to the same instance. The set of available VMs $Av_{vm} = \{r_k \mid r_k \in R_{vm}, FT(t_i, r_k) \leq \theta_i, rel(r_k) > ST(t_i, r_k)\}$ is constructed, where $rel(r_k)$ is the release time of VM r_k . Second, the VM instance that incurs the minimum cost increment is selected from Av_{vm} . If multiple candidates exist, the VM with the minimum earliest finish time is chosen to reduce the gap between t_i and the previous task and to maximize data reuse. If ties still remain, the VM with the smallest residual lease time after accommodating t_i is selected to further utilize the leased time. Finally, if $Av_{vm} = \emptyset$, TRM evaluates each candidate VM type as a newly provisioned VM instance. For each candidate type, it computes the start time, finish time, and BTU-based cost increment under the selected time constraint θ_i . The VM type with the minimum cost increment among feasible candidates is selected. If multiple VM types incur the same cost increment, the one with the earliest finish time is selected.

The reference resource requirements for the VM pool can be regarded as approximately balanced after RRD, although VM fragments are still difficult to avoid. Based on the above reference-based estimation, the number of provided VM instances should be around a reference threshold $Thre$ during the workflow execution.

$$Thre = \lceil \sum_{t_i \in T_{vm}} MET(t_i)/D \rceil \quad (24)$$

T_{vm} is the set of tasks assigned to the VM pool. Fewer VM instances might cause the deadline to be missed, whereas more VM instances result in more fragments. However, the proposed algorithm prefers to select a VM instance from the rented ones, which results in fewer rented VMs in the initial phase of workflow execution and more rented VMs in later stages. Therefore, at the beginning phase of workflow execution, if the number of rented VM instances is less than $Thre$, t_i is allocated to a new VM instance. Since $Thre$ is derived from the reference execution-time estimates, it is used only as a reference threshold for early-stage VM provisioning. The initially provisioned VMs follow the reference VM configuration that provides p^{ref} , while subsequent task mappings may select concrete heterogeneous VM types according to actual execution time, startup time, and BTU-based cost increment.

4.5 Task Migration for Improvement (TMI)

After TRM, there are still many tasks allocated to SCs. When consecutive SC tasks can be consolidated within a single VM's BTU, migrating them to a newly provisioned VM may further reduce the total rental cost. Therefore, in TMI, tasks deployed on SCs are first sorted according to their start times in ascending order. Let $Q_{SC} = (q_1, q_2, \dots, q_m)$

be the sorted list of SC tasks. For each task q_i in Q_{SC} , the migration-specific latest finish time is recalculated as

$$\widehat{LFT}(q_i) = \begin{cases} \min_{t_j \in su(q_i)} \{ST(t_j)\}, & su(q_i) \neq \emptyset, \\ D, & su(q_i) = \emptyset. \end{cases} \quad (25)$$

This constraint ensures that migrating q_i does not delay its successor tasks in the current schedule.

TMI regards q_i as an anchor task and evaluates each candidate VM type r as a newly provisioned VM instance. For candidate VM type r , a fixed rental window is determined by the BTU demand of q_i :

$$H_i^r = BTU \cdot \left\lceil \frac{\delta_r + ET(q_i, r)}{BTU} \right\rceil. \quad (26)$$

Then, q_i and its subsequent SC tasks are greedily inserted into this fixed rental window without introducing additional BTUs. Let $P_i^r = \{q_i, q_{i+1}, \dots, q_{i+S}\}$ be the maximal consecutive task set that can be placed on candidate VM type r . When tentatively inserting tasks into the fixed rental window, their start and finish times are recalculated according to both the VM availability and the finish times of their predecessor tasks. For each task $q_{i+s} \in P_i^r$, its tentative finish time must satisfy

$$FT(q_{i+s}, r) \leq \widehat{LFT}(q_{i+s}), \quad FT(q_{i+s}, r) \leq rel(r), \quad (27)$$

where $rel(r)$ is the release time determined by the fixed rental window H_i^r .

Let $C_{SC}(q_{i+s})$ denote the usage-based cost of task q_{i+s} under its currently selected SC configuration. If the total rental cost is reduced, i.e.,

$$\sum_{s=0}^S C_{SC}(q_{i+s}) > c_r \cdot H_i^r, \quad (28)$$

the candidate VM type and the migrated task set with the largest positive cost decrement are selected. These tasks are then moved to the selected VM instance.

4.6 Computational Time Complexity Analysis

The computational time complexity of TS is $O(|E| + N)$, since task priorities are calculated by traversing the workflow dependencies and tasks. In RRD, calculating the reference-based intervals, assigning tasks to resource pools, and computing task levels require $O(N)$, $O(N)$, and $O(|E| + N)$ time, respectively. Therefore, the computational time complexity of RRD is $O(|E| + N)$. Let M_{vm} and M_{sc} denote the numbers of candidate VM types and SC configurations, respectively. For SC pool tasks, TRM checks candidate SC configurations, resulting in $O(NM_{sc})$ time. For VM pool tasks, TRM checks currently rented VM instances, whose number is at most $O(N)$ in the worst case, resulting in $O(N^2)$ time. When a new VM instance is provisioned, candidate VM types are checked, adding $O(NM_{vm})$ time. Therefore, the computational time complexity of TRM is $O(N^2 + NM_{sc} + NM_{vm})$. In TMI, SC tasks are first sorted according to their start times, requiring $O(N \log N)$ time. For each anchor task, subsequent SC tasks may be greedily checked within the fixed VM rental window. Thus, TMI requires $O(N^2)$ time when the VM type is fixed, and $O(N^2 M_{vm})$ time when all candidate VM types are

evaluated. Combining the above components, the overall computational time complexity of WSHC can be written as $O(N^2 M_{vm} + NM_{sc})$. Since the numbers of VM types and SC configurations provided by a cloud platform are usually limited and can be treated as constants, i.e., $M_{vm} = O(1)$ and $M_{sc} = O(1)$, the overall worst-case complexity of WSHC can be simplified to $O(N^2)$.

5 EXPERIMENTAL RESULTS

To evaluate the proposed WSHC framework in hybrid VM-SC environments, we first calibrate the key parameters of WSHC under homogeneous settings to identify the optimal parameter combination. In addition, the schedules generated by several existing methods for similar problems are compared under both homogeneous and heterogeneous settings. All evaluated algorithms are implemented in Python and run on an AMD Ryzen 9 7940HX with Radeon Graphics CPU @ 2.40 GHz with 16 GB of RAM. The source code and the dataset are available in the anonymous repository at: <https://anonymous.4open.science/r/WSHC>.

All algorithms are compared using the RPD (Relative Percentage Deviation) measure, which is defined by:

$$RPD(\%) = \frac{C(A) - C^*}{C^*} \times 100\% \quad (29)$$

$C(A)$ is the total rental cost obtained by algorithm A , while C^* is the minimal cost obtained among all compared algorithms on the same workflow instance under the same deadline factor and resource setting. The ANOVA (multi-factor analysis of variance) technique is adopted to analyze results. The three main hypotheses (normality, homoscedasticity, independence of the residuals) are checked from the residuals of the experiments. All three hypotheses are acceptable from this analysis.

In the experiments, both VM types and SC configurations are assigned 1, 2, 4, and 8 vCPUs, with 2 GiB of memory per vCPU. The 4-vCPU configurations are used as the baseline configurations in homogeneous cases. It should be noted that these baseline configurations are not fixed as the algorithmic reference speed p^{ref} used in WSHC. The resource configurations are summarized in Table 2. The VM types are configured according to the AWS EC2 a1 family, with hourly billing granularity. The SC configurations are configured according to AWS Fargate Linux/X86 pricing, where SCs are charged at task granularity based on the requested vCPU, memory size, and charged usage duration. The vCPU price is 0.04048\$/vCPU-hour, and the memory price is 0.004445\$/GB-hour. The equivalent hourly cost of each SC configuration is calculated as the sum of its vCPU and memory costs. The relative processing speed is proportional to the number of vCPUs and normalized by the 4-vCPU baseline configuration. Thus, the relative speeds of the 1, 2, 4, and 8 vCPU configurations are 0.25, 0.50, 1.00, and 2.00, respectively. The startup times of VM types are set to {60, 90, 120, 180} seconds for a1.medium, a1.large, a1.xlarge, and a1.2xlarge, respectively. The startup times of SC configurations are set to {5, 8, 12, 20} seconds for SC-1, SC-2, SC-3, and SC-4, respectively. In the cost calculation, startup time is also included in the billable duration.

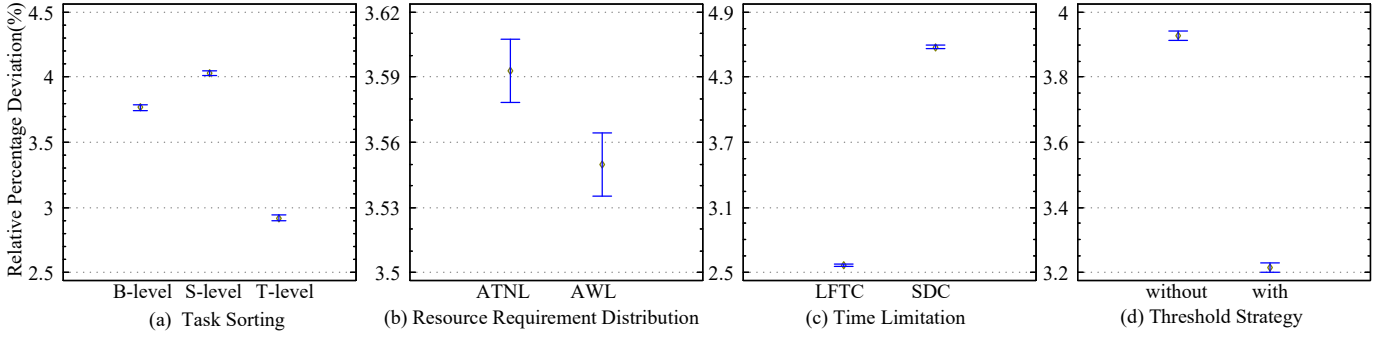


Fig. 3. Mean plots of different WSHC strategies and parameters with 95.0% Tukey HSD confidence level intervals under homogeneous VM-SC resource settings: a) Task sorting, b) Resource requirement distribution, c) Time limitation, d) Threshold strategy.

TABLE 2
Resource configurations used in the experiments.

Type	vCPU (cores)	Memory (GB)	Relative Speed	Price (\$/h)	Startup (s)
VM types					
a1.medium	1	2	0.25	0.0255	60
a1.large	2	4	0.50	0.051	90
a1.xlarge	4	8	1.00	0.102	120
a1.2xlarge	8	16	2.00	0.204	180
SC configurations					
SC-1	1	2	0.25	0.04937	5
SC-2	2	4	0.50	0.09874	8
SC-3	4	8	1.00	0.19748	12
SC-4	8	16	2.00	0.39496	20

5.1 Parameter Calibration

The proposed WSHC has four parameters that need to be calibrated. In TS, three strategies (B-Level, S-Level, and T-Level) are used to calculate task priority. In RRD, two strategies (ATNL, AWL) are adopted to reduce the imbalance of resource requirements for VMs based on average task number limitation and average workload limitation, respectively. In TRM, two task-resource mapping strategies are compared: one that applies the threshold $Thre$ (with) and one that does not (without). In addition, two kinds of time constraints are considered (SDC, LFTC) when allocating tasks. Therefore, there are $3 \times 2 \times 2 \times 2 = 24$ combinations of strategies for the proposed WSHC.

To calibrate the parameters, a large set of workflow instances are randomly generated considering various cases. The number of tasks N for each workflow is tested at the following values: $\{100, 200, 300, 400, 500\}$. The link density LD is defined as the ratio of the number of edges to the number of tasks, which is selected from $\{2, 3, 4, 5\}$. The base computation time of each task follows a uniform distribution $U(1, 3600)$ on the 4-vCPU baseline configuration. The computation-to-communication ratio (CCR) is tested at $\{0.5, 0.67, 1.0, 1.5, 2.0\}$. The deadline for workflow instances is defined as:

$$D = (1 + \alpha) \times EFT(t_N),$$

where $EFT(t_N)$ is calculated using the base execution times on the 4-vCPU baseline configuration. The deadline factor

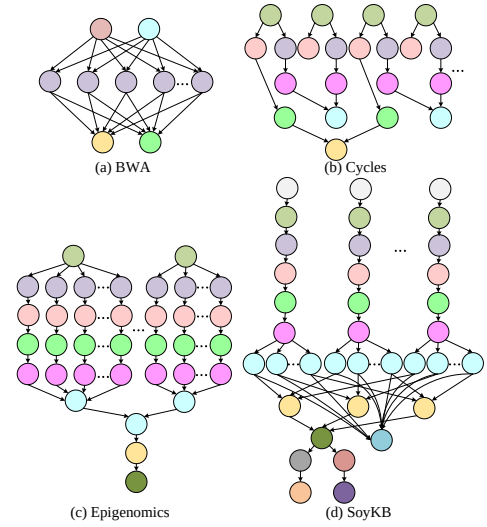


Fig. 4. Structures of four workflows: a) BWA, b) Cycles, c) Epigenomics, d) SoyKB.

α has five values $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ that reflect different deadline tightness levels. 10 instances are generated for each combination of N , LD , CCR and α . Therefore, $5 \times 4 \times 5 \times 5 \times 10 = 5000$ instances are generated in total. The total number of treatments is therefore $5000 \times 24 = 120000$.

The mean plots of the four parameters with 95.0% Tukey HSD intervals are shown in Fig. 3. Since the plots only show marginal effects, the final configuration is selected based on the mean RPD of the complete parameter combinations. The combination T-Level + AWL + LFTC + Thre achieves the lowest mean RPD among all tested settings. Therefore, WSHC adopts T-Level for task sequencing, AWL for resource requirement distribution, LFTC for time limitation, and enables Thre in the following comparisons.

5.2 Algorithm Comparison under Homogeneous Resources

To evaluate the performance of WSHC, we compare it with several representative methods designed for related settings. Specifically, SMOHEFT [13], originally proposed for heterogeneous serverless scheduling under budget and deadline constraints, is included with the budget set to infinity. In addition, VM-based scheduling algorithms are

considered for comparison, including IC-PCP [4], PSO [5], and CEDA [6]. Finally, GRP-HEFT proposed in [7] and BCWS proposed in [21] are adopted as baselines in a hybrid configuration. Although both GRP-HEFT and BCWS are designed to minimize the workflow makespan under budget constraints, we extend these two algorithms by inverting their objectives and constraints and allowing them to rent SC and VM resources. This modification enables the algorithms to greedily optimize the cost while meeting the deadline constraint. In this subsection, all baselines are first compared under the homogeneous VM-SC resource setting with startup times considered. For VM-only baselines, tasks can be assigned only to VMs. For SC-only baselines, tasks can be assigned only to SCs. For hybrid baselines, both VMs and SCs are available.

Four scientific workflows, namely BWA, Cycles, Epigenomics, and SoyKB, are used to evaluate the performance of the proposed WSHC. These workflows are widely adopted benchmarks in workflow scheduling research [6] and cover both computation-intensive and data-intensive applications. Specifically, BWA and Cycles are computation-intensive workflows, whereas Epigenomics and SoyKB are data-intensive workflows. Their different workflow structures are illustrated in Fig. 4, where nodes with different colors in each subfigure denote different task types.

WfGen³ [23] is adopted to generate a large number of synthetic realistic workflow instances by analyzing actual workflow executions. The number of tasks N of each workflow takes values from $\{100, 200, 300, 400, 500\}$. The deadline factor α is selected from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each combination of N and α of each workflow, 10 workflow instances are generated. Therefore, $4 \times 5 \times 5 \times 10 = 1000$ instances are generated for each resource setting. Thus, each homogeneous or heterogeneous comparison involves $7 \times 1000 = 7000$ algorithm runs. Most tasks of BWA, Cycles, and Epigenomics generated by WfGen using default parameters are fine-grained. This causes the deadline D of all workflow instances to be significantly less than one hour and to fall into the interval η_1 , which is not suitable for the problem under study [24]. Therefore, for a fair comparison, the execution time of tasks in these workflows is enlarged ten times to keep a similar magnitude to SoyKB. Some statistical indicators of the generated dataset are shown in Table 3, including ALD (average link density), AD (average depth), ATCT (average total computation time), and ATDIT (average total data transmission time).

TABLE 3
Statistical indicators of workflows.

Workflow	ALD	AD	ATCT	ATDIT
BWA	3.95	3	52113.66s	775.63s
Cycles	1.20	4	42830.99s	893.81s
Epigenomics	1.21	9	109443.86s	11794.86s
SoyKB	1.93	11	104076.08s	62430.08s

To demonstrate the effectiveness of the compared algorithms, the influence of two parameters, i.e., the number of tasks N and the deadline factor α , is evaluated over the four

workflows. The RPD variations of different algorithms under homogeneous VM-SC resources with 95% Tukey HSD intervals are shown in Figures 5 and 6. As shown in Figure 5, WSHC consistently achieves the lowest or near-lowest RPD across different task numbers, and its RPD remains relatively stable as N increases from 100 to 500. In contrast, IC-PCP and CEDA show much larger RPD values, indicating that their schedules are less cost-efficient in the homogeneous VM-SC setting. The other baseline algorithms, including BCWS, GRPHEFT, PSO, and SMOHEFT, generally obtain moderate RPD values, but their average performance is still worse than that of WSHC. Figure 6 further shows the effect of the deadline factor α . WSHC maintains a low RPD under all deadline settings, demonstrating stable cost performance under both tight and loose deadlines. By comparison, IC-PCP and CEDA become increasingly less competitive as α increases, while the remaining baseline algorithms exhibit only moderate changes. These trends indicate that WSHC is less sensitive to workflow size and deadline tightness in the homogeneous VM-SC setting.

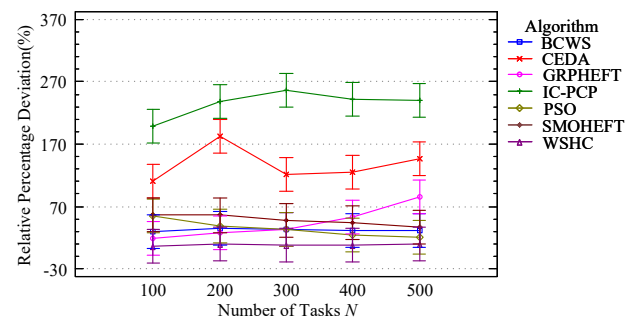


Fig. 5. RPD variation under different task numbers in the homogeneous resource setting.

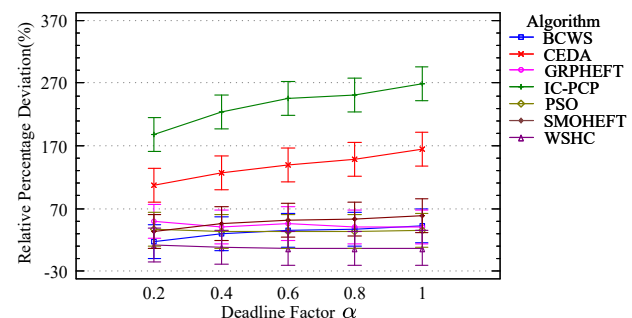


Fig. 6. RPD variation under different deadline factors in the homogeneous resource setting.

Table 4 reports the mean RPD, mean rental cost, win ratio, and deadline satisfaction ratio under homogeneous resources. The win ratio denotes the percentage of workflow instances for which an algorithm achieves the lowest rental cost, with ties counted for all tied algorithms. WSHC achieves the lowest mean RPD of 7.996%, the lowest mean cost of 3.906, and the highest win ratio of 50.8%. Compared with BCWS, the second-best algorithm in terms of mean RPD, WSHC reduces the mean RPD from 32.376% to 7.996%. Moreover, WSHC satisfies all deadline constraints, while PSO and IC-PCP show a small number of deadline violations.

3. <https://wfcommons.org/generator>

These results confirm that WSHC effectively reduces rental cost while maintaining stable deadline satisfaction under homogeneous VM-SC resources.

TABLE 4
Cost and RPD comparison under homogeneous resources

Algorithm	Mean RPD(%)	Mean Cost	Win Ratio(%)	Deadline Satisfaction(%)
WSHC	7.996	3.906	50.8	100.0
BCWS	32.376	4.825	13.3	100.0
PSO	34.521	4.670	6.6	93.7
GRP-HEFT	43.549	5.769	24.6	100.0
SMOHEFT	48.161	5.462	10.3	100.0
CEDA	137.483	8.726	3.9	100.0
IC-PCP	235.209	9.551	8.3	99.9

5.3 Algorithm Comparison under Heterogeneous Resources

Table 5 reports the mean RPD, mean rental cost, win ratio, and deadline satisfaction ratio of all compared algorithms in the heterogeneous resource setting. As shown in Table 5, WSHC achieves the lowest mean RPD of 9.002% and the lowest mean cost of 3.657. It also obtains the highest win ratio of 50.4%, indicating that WSHC most frequently finds the minimum-cost schedule under heterogeneous VM-SC resources. Compared with BCWS, which has the second-lowest mean RPD, WSHC reduces the mean RPD from 19.696% to 9.002%. All algorithms satisfy the deadline constraints in the heterogeneous setting. PSO performs considerably worse under heterogeneous resources (mean RPD 164.460% vs. 34.521%), as the enlarged search space in this setting makes it more difficult to converge to near-optimal solutions. These results show that WSHC can effectively exploit heterogeneous VM-SC resource options and maintain a clear cost advantage over the baseline algorithms.

TABLE 5
Cost and RPD comparison under heterogeneous resources

Algorithm	Mean RPD(%)	Mean Cost	Win Ratio(%)	Deadline Satisfaction(%)
WSHC	9.002	3.657	50.4	100.0
BCWS	19.696	3.908	24.7	100.0
GRP-HEFT	26.401	4.185	15.5	100.0
SMOHEFT	36.403	4.568	12.0	100.0
CEDA	54.315	5.420	13.6	100.0
IC-PCP	55.876	5.525	17.5	100.0
PSO	164.460	7.847	0.1	100.0

5.4 Runtime Performance and Scalability

To further evaluate computational scalability, Table 6 reports the average runtime of all compared algorithms under heterogeneous resources. The results are grouped by the number of tasks and averaged over all workflow applications, deadline factors, and generated instances. As shown in Table 6, the runtime of most deterministic heuristic algorithms increases smoothly with the workflow size. For WSHC, the average runtime increases from 0.0052 s to 0.0759 s as the number of tasks grows from 100 to 500, which is consistent with the computational time complexity analysis in Section 4.6. The average runtime of WSHC is

only 0.0351 s over all tested cases, indicating that its cost advantage does not introduce excessive scheduling overhead. In contrast, PSO incurs much higher runtime because it repeatedly evaluates candidate schedules during iterative search.

TABLE 6
Average runtime comparison under heterogeneous resources

Algorithm	100	200	300	400	500	Mean
WSHC	0.0052	0.0154	0.0316	0.0475	0.0759	0.0351
BCWS	0.0070	0.0216	0.0438	0.0769	0.1189	0.0536
CEDA	0.0062	0.0296	0.0758	0.1136	0.2140	0.0879
GRP-HEFT	0.0047	0.0134	0.0273	0.0450	0.0700	0.0321
IC-PCP	0.0068	0.0312	0.0796	0.1175	0.2162	0.0903
PSO	4.0927	9.5939	16.4993	24.1889	34.4954	17.7741
SMOHEFT	0.0034	0.0103	0.0208	0.0350	0.0544	0.0248

6 CONCLUSIONS AND FUTURE WORK

This paper investigates latency-sensitive workflow scheduling in a hybrid VM-serverless container environment with startup delays and both homogeneous and heterogeneous resource settings. We formulate the cost-aware scheduling problem and propose WSHC, a hybrid scheduling algorithm that integrates task prioritization, resource requirement distribution, time limitation assignment, and threshold-based scheduling decisions. Experimental results show that WSHC consistently achieves lower RPD and rental cost than the compared algorithms under both homogeneous and heterogeneous settings, demonstrating its robustness to startup-delay modeling and its effectiveness in exploiting hybrid VM-SC resources. In future work, we will study more dynamic scenarios, including online workflow arrivals, multi-tenant resource contention, uncertain task execution times, energy-aware scheduling, and validation on real cloud platforms.

REFERENCES

- [1] S.-M. Park and M. Humphrey, "Predictable high-performance computing using feedback control and admission control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 396–411, 2011.
- [2] H. Zhu, M. Li, and H. You, "Rhino: An efficient serverless container system for small-scale hpc applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 8, pp. 1560–1573, 2025.
- [3] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.
- [4] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.
- [5] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.
- [6] R. A. Haidri, C. P. Katti, and P. C. Saxena, "Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 6, pp. 666–683, 2020.
- [7] H. R. Faragardi, M. R. Saleh Sedghpour, S. Fazlihamadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1239–1254, 2020.
- [8] M. Fan, L. Ye, X. Zuo, and X. Zhao, "A bidirectional workflow scheduling approach with feedback mechanism in clouds," *Expert Systems with Applications*, vol. 249, p. 123494, 2024.

- [9] X. Cai, Y. Zhang, M. Li, L. Wu, W. Zhang, and J. Chen, "Dynamic deadline constrained multi-objective workflow scheduling in multi-cloud environments," *Expert Systems with Applications*, vol. 258, p. 125168, 2024.
- [10] 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.
- [11] W. Abdellatif, "Energy-efficient workflow task scheduling with deadline and budget constraints on dvfs-enabled cloud systems," *Future Generation Computer Systems*, p. 108527, 2026.
- [12] C. Lin and H. Khazaee, "Modeling and optimization of performance and cost of serverless applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 615–632, 2020.
- [13] M. Majewski, M. Pawlik, and M. Malawski, "Algorithms for scheduling scientific workflows on serverless architecture," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 782–789.
- [14] Z. Zhang, C. Jin, and X. Jin, "Jolteon: Unleashing the promise of serverless for serverless workflows," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 167–183.
- [15] Z. Li, Y. Liu, L. Guo, Q. Chen, J. Cheng, W. Zheng, and M. Guo, "Faasflow: Enable efficient workflow execution for function-as-a-service," in *Proceedings of the 27th acm international conference on architectural support for programming languages and operating systems*, 2022, pp. 782–796.
- [16] A. Joosen, A. Hassan, M. Asenov, R. Singh, L. Darlow, J. Wang, Q. Deng, and A. Barker, "Serverless cold starts and where to find them," in *Proceedings of the Twentieth European Conference on Computer Systems*, 2025, pp. 938–953.
- [17] P. A. Birajdar, D. Meena, A. Satpathy, and S. K. Addya, "Dcrda: deadline-constrained function scheduling in serverless-cloud platform," *Cluster Computing*, vol. 28, no. 6, p. 355, 2025.
- [18] K. Mahajan, D. Figueiredo, V. Misra, and D. Rubenstein, "Optimal pricing for serverless computing," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
- [19] A. Mampage, S. Karunasekera, and R. Buyya, "Deep reinforcement learning for scheduling applications in serverless and serverful hybrid computing environments," *IEEE Transactions on Services Computing*, vol. 18, no. 2, pp. 718–728, 2025.
- [20] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 706–721.
- [21] J. Zhang, X. Li, L. Chen, and R. Ruiz, "Scheduling workflows with limited budget to cloud server and serverless resources," *IEEE Transactions on Services Computing*, vol. 17, no. 4, pp. 1766–1779, 2024.
- [22] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2017.
- [23] T. a. Coleman, H. Casanova, L. Pottier, M. Kaushik, E. Deelman, and R. Ferreira da Silva, "WfCommons: A Framework for Enabling Scientific Workflow Research and Development," *Future Generation Computer Systems*, vol. 128, pp. 16–27, 2022.
- [24] R. F. Da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, "Online task resource consumption prediction for scientific workflows," *Parallel Processing Letters*, vol. 25, no. 03, p. 1541003, 2015.
- [25] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.



Jinquan Zhang is currently a Lecturer with the School of Computer Science, Guangdong University of Technology, China. His research interests include cloud computing resource management, serverless computing, and scheduling optimization. He has published more than ten research papers in well-known international journals and conferences, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Services Computing*, and *Future Generation Computer Systems*.



Xiaoping Li (M'09–SM'12) received his B.Sc. and M.Sc. degrees in Applied Computer Science from the Harbin University of Science and Technology in 1993 and 1999, respectively, and the Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology in 2002. He is the dean of the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China. He has authored more than 100 academic papers in journals such as *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Services Computing*. His research interests include scheduling in cloud computing, scheduling in cloud manufacturing, and service computing.



Long Chen received his B.Sc. and Ph.D. degrees in Computer Science and Engineering from Southeast University, Nanjing, China, in 2009 and 2018, respectively. He is currently a lecturer in the Department of Computer Science and Engineering at Southeast University, Nanjing, China. He has published more than 20 papers in international journals and conferences, such as *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, and *IEEE Transactions on Automation Science and Engineering*. His main research interests include task scheduling in cloud computing, service-oriented computing, evolutionary computation, and data normalization in smart buildings.



Zhicheng Cai (Member, IEEE) received the Ph.D. degree in Applied Computer Science from Southeast University, Nanjing, China, in 2015. He is an associate professor with the Nanjing University of Science and Technology, China. His research interests focus on resource scheduling in cloud, fog, and edge computing. He has authored more than 20 publications in journals such as *IEEE Transactions on Computers*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Automation Science and Engineering*, and *Future Generation Computer Systems*, and at conferences such as ICSOC, ICPAD-S, ISPA, ICA3PP, CLOUD, HPCC, SMC, CBD, and CASE.



Rubén Ruiz is a full professor of Statistics and Operations Research at the Universitat Politècnica de València, Spain. He is a co-author of more than 60 papers in international journals and has participated in presentations of more than a hundred papers in national and international conferences. He is editor of Elsevier's journal *Operations Research Perspectives (ORP)* and co-editor of the JCR-listed journal *European Journal of Industrial Engineering (EJIE)*. He is also an associate editor of other important journals such as *TOP* and *Applied Mathematics and Computation*, as well as a member of the editorial boards of several journals, most notably *European Journal of Operational Research* and *Computers and Operations Research*. His research interests include scheduling and routing in real-life scenarios.



Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored more than 625 publications and seven textbooks. He is one of the most highly cited authors in computer science and software engineering worldwide (h-index=155, g-index=340, more than 125,500 citations). Microsoft Academic Search Index ranked him as the #1 author in the world (2005–2016) for both field rating and citation evaluations in the area of distributed and parallel computing. He was recognized as a "Web of Science Highly Cited Researcher" during 2016–2021 by Thomson Reuters.

APPENDIX A

PROOF OF THEOREMS

Theorem A1. If $D \in (0, \min_{t_i \in T} \{MET(t_i)^*/MET(t_i) \cdot c_{vm}^{ref}/c_{sc}^{ref} \cdot BTU\}]$, then $C_{SC} \leq C_H$ under the reference execution model.

Proof. Suppose T_{vm} is the set of tasks assigned to the VM pool in the schedule corresponding to C_H . $T_{vm} \neq \emptyset$, otherwise $C_{SC} = C_H$. Let $T_w = \sum_{t_i \in T_{vm}} MET(t_i)$ denote the total reference workload of T_{vm} , and $T_w^* = \sum_{t_i \in T_{vm}} MET(t_i)^*$. Obviously, $T_w^* \leq T_w$. Since $D \in (0, \min_{t_i \in T} \{MET(t_i)^*/MET(t_i) \cdot c_{vm}^{ref}/c_{sc}^{ref} \cdot BTU\}]$, we have

$$D \leq \frac{T_w^*}{T_w} \cdot \frac{c_{vm}^{ref}}{c_{sc}^{ref}} \cdot BTU.$$

Therefore,

$$\begin{aligned} C_H - C_{SC} &\geq \left\lceil \frac{T_w^*}{D} \right\rceil \cdot BTU \cdot c_{vm}^{ref} - T_w \cdot c_{sc}^{ref} \\ &\geq \left\lceil \frac{T_w \cdot c_{sc}^{ref}}{c_{vm}^{ref} \cdot BTU} \right\rceil \cdot BTU \cdot c_{vm}^{ref} - T_w \cdot c_{sc}^{ref} \\ &\geq \frac{T_w \cdot c_{sc}^{ref}}{c_{vm}^{ref} \cdot BTU} \cdot BTU \cdot c_{vm}^{ref} - T_w \cdot c_{sc}^{ref} = 0. \end{aligned}$$

□

Theorem A2. Assume $c_{sc}^{ref} > c_{vm}^{ref}$. Let $ub = \lceil c_{vm}^{ref}/(c_{sc}^{ref} - c_{vm}^{ref}) \rceil$. $\Delta C_i^{SC} \geq \Delta C_i^{VM}$ if $MET(t_i) \in \bigcup_{k=1}^{ub} [k \cdot BTU \cdot c_{vm}^{ref}/c_{sc}^{ref}, k \cdot BTU] \cup (ub \cdot BTU, +\infty)$ under the reference VM-based cost comparison.

Proof. Let $ET(t_i, r)$, $ST(t_i, r)$, and $FT(t_i, r)$ be the execution time, start time, and finish time of t_i on resource instance r , respectively, while r_1 and r_2 are the SC and VM instances to which t_i is allocated. Under the reference execution model used in RRD, the SC execution duration of t_i is estimated by $MET(t_i)$. Thus,

$$\Delta C_i^{SC} = c_{sc}^{ref} \cdot MET(t_i).$$

r_2 is either a new VM instance or an already rented VM instance. If r_2 has been rented before scheduling t_i , part of the last time interval of r_2 must be occupied by t_i . Otherwise, a new VM instance is selected. Suppose that $rel(r_2)$ is the release time of the last time interval of r_2 before t_i is allocated to it. Under the reference execution model, $MET(t_i)$ provides an upper-bound estimate of the execution duration used in RRD. Then

$$FT(t_i, r_2) - rel(r_2) \leq ET(t_i, r_2) \leq MET(t_i),$$

and

$$\Delta C_i^{VM} \leq c_{vm}^{ref} \cdot BTU \cdot \left\lceil \frac{MET(t_i)}{BTU} \right\rceil.$$

If r_2 is a new instance,

$$\Delta C_i^{VM} = c_{vm}^{ref} \cdot BTU \cdot \left\lceil \frac{MET(t_i)}{BTU} \right\rceil.$$

Let $\lceil MET(t_i)/BTU \rceil = k$, where $MET(t_i) \in ((k-1) \cdot BTU, k \cdot BTU]$. k is a positive integer if t_i is not a dummy task. To guarantee $\Delta C_i^{SC} \geq \Delta C_i^{VM}$, it is sufficient to require

$$c_{sc}^{ref} \cdot MET(t_i) \geq c_{vm}^{ref} \cdot BTU \cdot k,$$

because $\Delta C_i^{VM} \leq c_{vm}^{ref} \cdot BTU \cdot k$. This sufficient condition leads to

$$MET(t_i) \geq k \cdot BTU \cdot \frac{c_{vm}^{ref}}{c_{sc}^{ref}}.$$

Therefore, $MET(t_i) \in \eta_2$ is a sufficient condition for $\Delta C_i^{SC} \geq \Delta C_i^{VM}$, where η_2 is defined as

$$\eta_2 = \bigcup_{k=1}^{+\infty} \left[k \cdot BTU \cdot \frac{c_{vm}^{ref}}{c_{sc}^{ref}}, k \cdot BTU \right] \cap ((k-1) \cdot BTU, k \cdot BTU].$$

$MET(t_i) \in [k \cdot BTU \cdot c_{vm}^{ref}/c_{sc}^{ref}, k \cdot BTU]$ when

$$k \cdot BTU \cdot c_{vm}^{ref}/c_{sc}^{ref} > (k-1) \cdot BTU,$$

i.e.,

$$k < 1 + \frac{c_{vm}^{ref}}{c_{sc}^{ref} - c_{vm}^{ref}}.$$

As k is an integer,

$$k \leq \left\lceil \frac{c_{vm}^{ref}}{c_{sc}^{ref} - c_{vm}^{ref}} \right\rceil = ub.$$

Therefore, η_2 can be rewritten as

$$\eta_2 = \bigcup_{k=1}^{ub} \left[k \cdot BTU \cdot \frac{c_{vm}^{ref}}{c_{sc}^{ref}}, k \cdot BTU \right] \cup (ub \cdot BTU, +\infty).$$

□

APPENDIX B

ENERGY CONSUMPTION ANALYSIS

Although WSHC minimizes rental cost rather than energy consumption, we further evaluate the estimated energy usage of the evaluated schedules to examine the relationship between the two objectives. Following the linear power model in [25], the power of a resource with utilization u is estimated as

$$P_r(u) = \frac{v_r}{4} (kP_{\max} + (1-k)P_{\max}u), \quad (30)$$

where v_r is the number of vCPUs, $P_{\max} = 250$ W is the peak power of the 4-vCPU reference resource, and $k = 0.7$ is the idle-to-peak power ratio. Startup and task execution intervals are treated as busy intervals with $u = 1$, while gaps between tasks on a VM are treated as idle intervals with $u = 0$. The lifetime of a resource begins when its startup starts and ends when its last assigned task finishes. The estimated energy of a schedule is calculated by integrating the power of all rented resources over their lifetimes and converting the result to kWh.

Table B1 reports the mean estimated energy and energy win ratio under both resource settings. The energy win ratio is the percentage of workflow instances for which an algorithm obtains the lowest estimated energy among the evaluated schedules, with ties counted for all tied algorithms. WSHC consumes 7.176 kWh on average under homogeneous resources, which is 7.8% higher than the 6.658 kWh obtained by CEDA. However, WSHC reduces the mean rental cost by 55.2% compared with CEDA, as shown in Table 4. Under heterogeneous resources, WSHC consumes 6.586 kWh on average, which is 13.9% higher than the 5.783 kWh obtained by SMOHEFT, while reducing

the mean rental cost by 19.9%. These results indicate that rental cost and energy consumption are distinct objectives: WSHC provides the strongest cost performance, but it does not optimize energy consumption explicitly.

TABLE B1
Estimated energy comparison under homogeneous and heterogeneous resources

Algorithm	Homogeneous		Heterogeneous	
	Mean Energy (kWh)	Win Ratio(%)	Mean Energy (kWh)	Win Ratio(%)
WSHC	7.176	0.4	6.586	3.2
BCWS	6.934	0.0	5.867	16.0
GRP-HEFT	8.708	0.1	5.802	14.8
SMOHEFT	6.914	0.0	5.783	92.0
CEDA	6.658	99.5	6.465	0.3
IC-PCP	7.375	0.0	6.553	0.2
PSO	8.609	0.0	8.721	0.0