Journal of Grid Computing (2023) 21:75
https://doi.org/10.1007/s10723-023-09711-9

RESEARCH

Efficient Prediction of Makespan Matrix Workflow Scheduling Algorithm for Heterogeneous Cloud Environments

Longxin Zhang · Minghui Ai · Runti Tan · Junfeng Man · Xiaojun Deng · Keqin Li

Received: 23 June 2023 / Accepted: 29 October 2023

Abstract Leveraging a cloud computing environment for executing workflow applications offers high flexibility and strong scalability, thereby significantly improving resource utilization. Current scholarly discussions heavily focus on effectively reducing the scheduling length (makespan) of parallel task sets and improving the efficiency of large workflow applications in cloud computing environments. Effectively managing task dependencies and execution sequences plays a crucial role in designing efficient workflow scheduling algorithms. This study forwards a high-efficiency workflow scheduling algorithm based on predict makespan matrix (PMMS) for heterogeneous cloud computing environments. First, PMMS calculates the priority of each task based on the predict makespan (PM) matrix and obtains the task scheduling list. Second, the optimistic scheduling length (OSL) value of each task is calculated based on the PM matrix and the earliest finish time. Third, the best virtual machine is selected for each task according to the minimum OSL value. A large number of substantial experiments show that the scheduling length of workflow for PMMS, compared with state-of-the-art HEFT, PEFT, and PPTS algorithms, is reduced by 6.84%–15.17%, 5.47%–11.39%, and 4.74%–17.27%, respectively. This hinges on the premise of ensuring priority constraints and not increasing the time complexity.

Keywords Cloud computing · Priority constraints · Predict makespan matrix · Scheduling length · Workflow scheduling

L. Zhang (✉) · M. Ai ·R. Tan ·X. Deng
School of Computer Science, Hunan University of Technology, Taishan, Zhuzhou 412007, Hunan, China
e-mail: longxinzhang@hut.edu.cn

M. Ai
e-mail: minghui_ai@163.com

R. Tan
e-mail: tanrunti@163.com

X. Deng
e-mail: little_army@hut.edu.cn

J. Man (✉)
School of Computer Science, Hunan First Normal University, Yuelu, Changsha 410205, Hunan, China
e-mail: mjfok@qq.com

K. Li
Department of Computer Science, State Universityof New York, 100 Hawk Drive, New Paltz 12561, NY, USA
e-mail: lik@newpaltz.edu

1 Introduction

Cloud computing can store, integrate related resources, and customize them on demand to provide users with personalized services. A cloud computing center is an excellent platform capable of running large-scale workflow applications and provides robust services to users through its powerful computing system, which is widely used in contexts such as smart cities, Internet of Things (IoT), weather forecasting, and other fields [1]. Many scientific studies are even cloud based computing

Published online: 28 November 2023

platforms, for instance, molecular dynamics [2], green IoT [3], etc. Heterogeneous cloud computing system (HCS) is a computing platform composed of multiple groups of resources and is interconnected through a high-speed network.

Task scheduling in HCS can be divided into either dynamic or static task scheduling [4,5]. Dynamic task scheduling is suitable for cases where the system environment and application parameters are unknown at the time of compilation, thus requiring decisions to be made during algorithm runs while bringing additional overhead. Static task scheduling means that information, such as communication time between tasks, has been predetermined before a task is started. This type of scheduling is usually divided into two phases [6]: task priority calculation and virtual machine (VM) selection. The task scheduling problem in cloud computing system [7] essentially designs appropriate scheduling strategies for the workflow application submitted to the cloud center. Afterward, it assigns each task in the application to the appropriate VM according to the kind of constraints (such as priority constraints, budget constraints, etc.), thereby completing the entire application scheduling in a short time (or at a low cost, etc.).

Existing algorithms, such as heterogeneous earliest finish time (HEFT) algorithm [8] and high performance task scheduling algorithm [9], overlook the characteristics of the successor tasks of the current task when selecting VM for a task. This ultimately affects (to some extent) the efficiency of these algorithms. It is worth mentioning that the predict earliest finish time (PEFT) algorithm [10] and the algorithm that calculates the priority of tasks and selects VMs on the basis of a predict cost matrix (PPTS) [11] can maintain the same time complexity as HEFT, however, PEFT does not care about the relative importance of each task, and PPTS may have priority order violations in workflows with minor computational costs. All of the above may result in reducing the efficiency of these algorithms. Hence, an efficient workflow scheduling algorithm based on the predict makespan matrix (PMMS) is proposed herein to minimize the makespan of workflow tasks in HCS. The reported predict makespan (PM) matrix generates an efficient task scheduling list in the task prioritizing phase and positively guides VM selection corresponding to tasks. PMMS considers both the impact of immediate successor tasks and the importance of the current task, thus making the task-VM allocation more reasonable. PMMS minimizes the scheduling length (makespan) [12] of workflow applications without sacrificing the algorithm's time complexity.

The contributions of the paper are as follows:

- It develops a priority calculation algorithm based on a PM matrix to minimize the makespan of workflow applications while satisfying priority constraints. The PM matrix considers the impact of the current task and predicts the impact of its successor on the whole workflow application, thus reducing the scheduling length.
- It also designs a novel task rank calculation rule in the priority calculation phase. Due to the "forward-looking" feature of the PM matrix, the priority calculation of tasks is more reasonable, and high priority tasks would be unscheduled after low priority tasks, thus improving the efficiency of the entire algorithm.
- It validates the superiority of the PMMS algorithm against the advanced HEFT, PEFT, and PPTS algorithms by comparing the parallel application of four kinds of randomly generated real world workflows of varying sizes.

The rest of the study is summarized as follows. Section 2 introduces the related work on heuristic task scheduling algorithms. Section 3 describes the workflow application model, the system model of cloud computing, and formalizes the problem. Section 4 elaborates on the concept of PM matrix and describes the PMMS algorithm in detail, including the specific strategies for the task priority and VM selection phases. We then show experimental results to evaluate the performance of the PMMS algorithm in Section 5. Finally, Section 6 concludes the current study.

2 Related Works

The efficiency of executing parallel applications in HCS relies heavily on the chosen scheduling method [13]. Many studies aim at minimizing the makespan of the entire workflow application. The task scheduling problem in HCS [14] is more complex compared to homogeneous computing systems. Following different optimization objectives, the existing heuristic algorithms can be classified into various objectives such as minimizing scheduling length, minimizing monetary cost, minimizing energy consumption, maximizing reliability algorithms, etc.

Topcuoglu et al. [8] presented the HEFT algorithm, which is a classic heuristic algorithm to minimize the scheduling length. HEFT assigns a VM with the minimum value of earliest finish time (EFT) to the candidate task without considering the computational overhead of any successor node of the current task. If there is a large disparity between the computational cost of the current task node and its direct successor node, there is not an optimal priority queue generated by the HEFT algorithm. Therefore, HEFT in HCS may lead to the algorithm's low scheduling efficiency. Arabnejad et al. [10] reported the PEFT algorithm according to an optimistic cost table, aiming to reduce the completion time of the entire workflow and decrease the time complexity of the algorithm. However, it focuses on reducing the earliest start time (EST) of succeeding tasks and does not pay attention to the relative importance of each task. The assignment results of tasks are not directly related to their own priority level, and tasks with higher priority may be scheduled after those with lower priority, which will lead to an increase in the scheduling length and an unreasonable assignment strategy for tasks. Djigal et al. [11] introduced an algorithm named PPTS to ultimately minimize the makespan of the workflow. However, it is not applicable to all workflow applications. For workflow applications with minor computational costs, PPTS may have problems in calculating the priority of tasks that violate the priority order specified in task scheduling. Zhang et al. [6] described an algorithm for relative distance. It maps a VM using the relative distance of tasks to improve the efficiency of VM utilization and reduce the makespan of applications while satisfying the deadline constraint. While the algorithm prioritizes enhancing the utilization efficiency of VMs, it may result in the issue of imbalanced resource allocation. Meanwhile, Kelefouras and Djemame [15] explored a strategy that can minimize the scheduling length of the task set by recognizing whether a task is executed either single-threaded or multithreaded. This approach can substantially reduce the scheduling lenghth and enhance the utilization of resources in task sets. Nonetheless, it may prove inadequate for accommodating intricate task dependencies and resource contention. Youness et al. [16] presented two fault-tolerant scheduling algorithms, namely the typical heuristic algorithm and the weighted mean completion time optimization algorithm, based on simulated annealing method. By introducing a weighted average maximum completion time and adding position based on task replication, makespan is decreased and system reliability is improved.

Arabnejad et al. [17] developed a deadline-budget-constrained-aware strategy, which introduces an adjustable cost-time trade-off to determine the optimal scheduling under the condition that both deadline and budget constraints are satisfied. The algorithm takes into account multiple factors, including cost, time, deadline, and budget constraints. Accordingly, it exhibits a high level of complexity, thereby demanding additional computing resources and time for its execution. In HCS, large workflow scheduling strategies usually take budget cost into account, and many classical algorithms are based on task scheduling lists computed by the HEFT algorithm. Arabnejad et al. [18] also proposed the heterogeneous budget constrained scheduling (HBCS) algorithm. HBCS minimizes the execution time of submitted applications within the user-budget cost specified by users. The budget allocation method of HBCS may be biased against low-priority tasks, resulting in extended workflow completion times. Chen et al. [19] improved HBCS and subsequently devised the budget level minimized scheduling length algorithm, which adopts the strategy of divide and conquer, changes the budget constraint of each subtask using the budget level, and achieves makespan minimization. This strategy can achieve a balance between the budgets of low and high priority tasks, while simultaneously reducing the scheduling length of workflow applications. Because other algorithms tend to consider energy consumption, Quan et al. [20] studied an algorithm via a weight-based mechanism to pre-allocate energy for unscheduled tasks to address the unfairness of traditional energy allocation methods for low-priority tasks. However, the time complexity of the algorithm is relatively high.

Recently, the goal of workflow scheduling in HCS has gradually focused on optimizing monetary cost, and either reducing energy consumption or improving scheduling reliability [21]. Chen et al. [22] focused on monetary cost optimization algorithms, such as the downward cost optimization algorithm, with "downward" referring to minimizing the cost of upward rank values from the entry to the exit tasks in descending order. Meanwhile, the downward upward cost optimization refers to minimizing the cost of upward ranking values from the exit to the entry tasks in ascending order. Liu et al. [23] proposed an online multi-workflow scheduling algorithm, NOSF, for schedul-

ing multiple workflows in a manner that not only predicts and dynamically adjusts task execution times to meet deadline constraints but also optimizes resource utilization and system performance. NOSF exhibits scalability and adaptability, accommodating various workflow scheduling problems with diverse types and scales. However, one limitation of NOSF is its failure to account for the variability in task execution times, thereby potentially leading to resource inefficiency and additional costs. Chen et al. [24] suggested an energy consumption minimization algorithm named MECABP based on the available budget preassignment policy and dynamic voltage-frequency scaling technique, thereby minimizing energy consumption while complying with cost constraints. The equitable distribution of budget costs among subtasks may prove inequitable for lower priority tasks, leading to increased energy consumption and extended completion times. Zhang et al. [25] meanwhile investigated a method that uses a mission-critical remapping (RMREC) strategy to decrease energy consumption under budget cost constraints. However, unlike the MECABP algorithm, RMREC uses critical task mapping to ensure that the energy consumption of tasks during scheduling does not exceed the given budget, avoids the waste of resources, and improves the efficiency of algorithm scheduling. Based on shared recovery technology, Zhang et al. [26] devised an algorithm to minimize energy consumption while maintaining the original reliability of the system without violating the deadline and priority constraints of the tasks. The algorithm's time complexity increases due to the consideration of multiple constraints. Saeedizade and Ashtiani [27] developed a dynamic deadline and budget-aware workflow scheduling algorithm (DDBWS), tailored for the workflow as a service environment. DDBWS uses dual factors to balance cost and resource utilization, ensuring a high success rate and effective planning of the algorithm. For the scheduling problem of parallel tasks in heterogeneous computer systems, Zhang et al. [28] examined a competition-aware reliability management algorithm with deadline and energy budget constraints, improving the reliability of task scheduling through redundant recycling. Redundancy recovery necessitates extra computing and communication resources, which leads to an increase in the system's overhead and energy consumption. Rodriguez and Buyya [29] introduced a multi-workflow elastic resource allocation and scheduling algorithm (EPSM) designed for WaaS platform. The algorithm is capable of adapting to environ-

Table 1 Summary of the reviewed algorithms

Literature	Behavior	Methods
Topcuoglu et al. [8]	Static	List scheduling
Arabnejad et al. [10]	Static	Optimistic cost table
Djigal et al. [11]	Static	Predict cost matrix
Zhang et al. [6]	Static	VMs mapping by the relative distance of tasks
Kelefouras and Djemame [15]	Static	Single/multithreaded identification
Youness et al. [16]	Static	Simulated annealing method
Arabnejad et al. [17]	Dynamic	Deadline-budget-constrained-aware strategy
Arabnejad et al. [18]	Static	Remaining cheapest budget
Chen et al. [19]	Static	Budget level factor
Quan et al. [20]	Static	Weight-based mechanism to pre-allocate energy
Chen et al. [22]	Static	Downward/upward cost optimization
Liu et al. [23]	Dynamic	Predicting and dynamically adjusting task execution time
Chen et al. [24]	Static	The average allocation of budget cost
Zhang et al. [25]	Static	Mission-critical remapping
Zhang et al. [26]	Static	Solving the multi-resource packing problem
Saeedizade and Ashtiani [27]	Dynamic	Dynamic deadline and budget-aware
Zhang et al. [28]	Static	Redundant recycling
Rodriguez and Buyya [29]	Dynamic	Containers, Scalable resource

mental and workload changes while still meeting the deadline constraint, reducing the cost of a workflow application. However, EPSM does not consider factors such as task priority, data transmission, and associated aspects. Table 1 summarizes the primary features and distinctions of the algorithms discussed in Section 2.

VMs are the fundamental resource units in cloud environments. Employing a finite number of VMs can ensure the efficient allocation and utilization of resources, preventing needless resource squandering and avoiding additional costs in accordance with the size of the workflow and the budgetary limitations specified upon the user. In practical scenarios, there are often constraints on the number of VMs, such as hardware resources and energy costs. Moreover, from the user's perspective, renting a limited number of VMs can satisfy their requirements while also reducing rental expenses. The current study only considers the task priority constraint and explores the workflow scheduling length minimization problem in HCS using a limited number of VMs.

3 Models

This section introduces the application and system models. Afterward, it formally describes the problem found herein.

3.1 Application Model

The directed acyclic graph (DAG) can be used to depict workflow applications running on VMs [30,31], expressed here as $G = \langle N, E, C, W \rangle$, where N is the set of tasks, with each task ξ_i ($\xi_i \in N$) varying in execution time on different VMs. E is the set of edges

Table 2 Description of notations

Notation	Definition
VM	Virtual machines
DAG	Directed Acyclic Graph
V	The set of VMs, and $\|V\|$ is the number of VMs
ξ_i	The i-th task
vm_j	The j-th VM
$e_{i,j}$	Communication edge connecting task ξ_i and task ξ_j
$c_{i,j}$	Communication time between tasks ξ_i and ξ_j
$w_{i,j}$	Execution cost of ξ_i on vm_j
$\overline{c_{i,j}}$	The average communication time between tasks ξ_i and ξ_j
$\overline{w_{i,j}}$	The average execution cost of ξ_i on $\|V\|$ VMs
$avail(vm_j)$	The earliest time when vm_j gets ready
$AFT(\xi_k)$	The actual finish time of task ξ_j's predecessors
CP	The set of critical path nodes in the DAG
$pre(\xi_i)$	The set of immediate predecessor nodes for task ξ_i
$succ(\xi_i)$	The set of immediate successors of task ξ_i
$EST(\xi_i, vm_j)$	Earliest start time of task ξ_i on vm_j
$EFT(\xi_i, vm_j)$	Earliest finish time of task ξ_i on vm_j
$PM(\xi_i, vm_j)$	Predict makespan value of task ξ_i on vm_j
$rank_{PM}(\xi_i)$	The rank value of ξ_i
θ	A constant that takes the value of 1.0
$OSL(\xi_i, vm_j)$	Optimistic schedule length value of task ξ_i on vm_j
Makespan	Actual finish time of the exit task (scheduling length)
SLR	Scheduling length ratio
CCR	Communication computing ratio

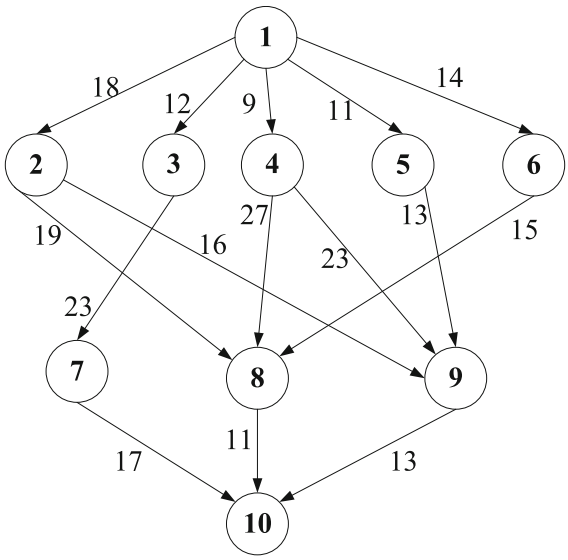

Fig. 1 Motivational example diagram of a DAG parallel application

directly connecting two tasks in the DAG. Each edge $e_{i,j}$ ($e_{i,j} \in E$) indicates the priority constraint between tasks ξ_i and ξ_j, where ξ_j cannot be executed before task ξ_i. C is the set of communication costs between two tasks that are directly connected, where $c_{i,j}$ ($c_{i,j} \in C$) represents the communication time of $e_{i,j}$. $c_{i,j} = 0$ if and only if ξ_i and ξ_j are executed on the same VM. W is the set of the computational cost of tasks on different VMs. $w_{i,j}$ ($w_{i,j} \in W$) is the computing cost of the task ξ_i assigned to execute on VM vm_j ($vm_j \in V$), where j denotes the j-th VM. $pre\,(\xi_i)$ refers to the set of direct predecessor nodes of task ξ_i and $succ\,(\xi_i)$ denotes the set of direct successor nodes of task ξ_i. A task without any predecessor can be represented as ξ_{entry}, while a task without a successor is represented as ξ_{exit}. It is agreed herein that the studied DAG workflow application has a unique entry/exit task. For simplicity, Table Table 2 summarizes the descriptions of notations used herein.

As an example of a typical DAG parallel application, Table 3 presents the computation cost of each task on three different sets of VMs seen in Fig. 1.

Task ξ_3 may enter the ready state only when its immediate predecessor task ξ_1 is finished. In Table 3, the computational cost of the second row in the second column indicates that the execution time of task ξ_2 on vm_1 is 68. When tasks ξ_1 and ξ_3 are not executed on the same VM, the communication cost between them is $c_{1,3} = 12$.

The computation time of ξ_i on vm_j is denoted by $w_{i,j}$. The average computation time $\overline{w_{i,j}}$ for ξ_i can be defined by

$$\overline{w_{i,j}} = \Big(\sum_{j=1}^{|V|} w_{i,j}\Big) \Big/ |V|, \tag{1}$$

where $|V|$ is the number of VMs.

$c_{i,j}$ is the communication cost required to transmit data $data_{i,j}$ from tasks ξ_i to ξ_j. The average communication time $\overline{c_{i,j}}$ is estimated as follows:

$$\overline{c_{i,j}} = \overline{L} + \frac{data_{i,j}}{\overline{B}}, \tag{2}$$

Table 3 Computational time of the task nodes in Fig. 1

$Task_i$	vm_1	vm_2	vm_3
ξ_1	51	21	36
ξ_2	68	71	9
ξ_3	84	31	13
ξ_4	27	16	35
ξ_5	22	43	34
ξ_6	16	54	13
ξ_7	33	22	31
ξ_8	51	20	35
ξ_9	48	60	31
ξ_{10}	47	78	63

where $\overline{L}$ is the average latency time of all VMs and $\overline{B}$ is the average transmission rate between VMs.

The EST of ξ_i on vm_j is the actual finish time of the direct predecessor tasks that ξ_i follows. Its expression can be calculated as

$$EST\left(\xi_i, vm_j\right) = \begin{cases} 0, & if\ \xi_i = \xi_{entry}; \\ \max\left\{avail\left(vm_j\right), \max\limits_{\xi_k \in pred(\xi_i)}\left\{AFT\left(\xi_k\right) + c_{k,i}\right\}\right\}, & otherwise, \end{cases} \tag{3}$$

where $avail\left(vm_j\right)$ is the earliest time when vm_j gets ready, and $AFT\left(\xi_k\right)$ is the actual finish time of ξ_i's predecessors.

Therefore, the earliest finish time $EFT\left(\xi_i, vm_j\right)$ of task ξ_i on vm_j is defined by

$$EFT\left(\xi_i, vm_j\right) = EST\left(\xi_i, vm_j\right) + w\left(\xi_i, vm_j\right). \tag{4}$$

After all tasks in the DAG are scheduled and executed, the scheduling length (also name makespan) is the actual completion time of the exit task. Therefore, makespan can be expressed as

$$\text{Makespan} = \max\left\{AFT\left(\xi_{exit}\right)\right\}. \tag{5}$$

3.2 System Model

The HCS model used herein is shown in Fig. 2. An HCS consists of three modules, namely users, resource deployment, and resource management. End users upload requirements and applications to the cloud center through user modules. The resource deployment module serves as a bridge between the users and the resource management module. The requests submitted by end users are then preprocessed by the resource deployment module, which includes the processing of applications and the design of scheduling strategies, etc. The task scheduler then transmits the user submitted tasks to the resource management module. In the resource deployment module, the resource manager sends the node status information obtained by the resource management module to the job scheduler and the task scheduler. Meanwhile, the job scheduler receives the request from the users module and the information sent by the resource manager, then transforms the application into the corresponding DAG and transmits it to the task scheduler. The task scheduler then assigns the user-submitted applications to the resource management module through the scheduling policy. The resource management module contains $|V|$ heterogeneous VMs and allocates appropriate VMs in accordance with different tasks to meet the personal-

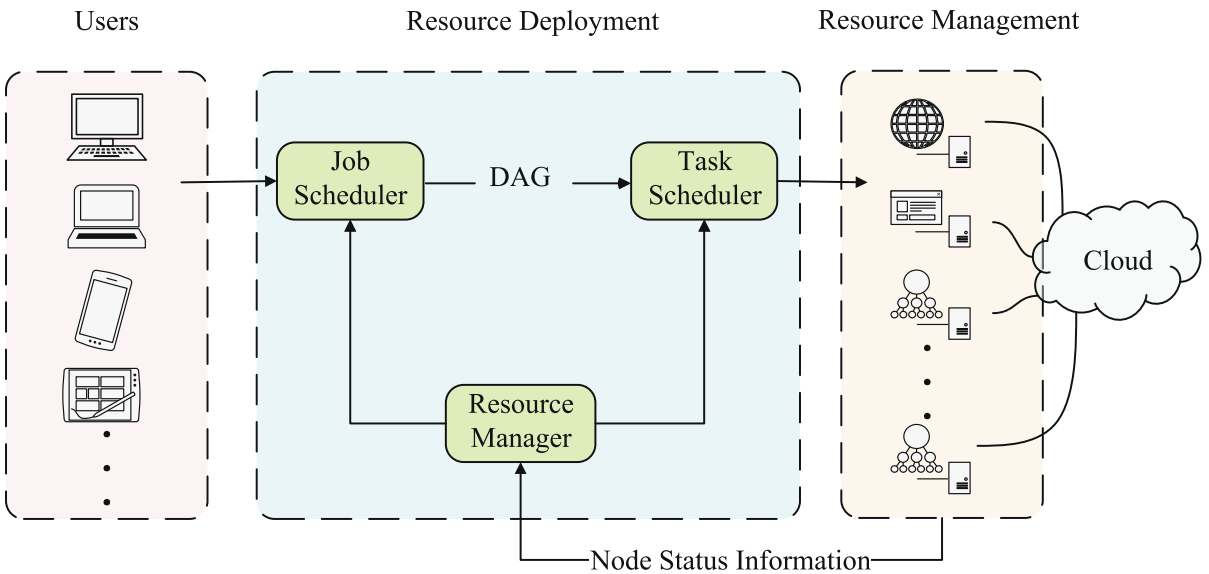

Fig. 2 HCS model

ized needs of users to save resources and reduce energy consumption.

3.3 Problem Definition

Workflow scheduling in HCS is the assignment of tasks to VMs for execution. This study posits that cloud computing center reasonably schedules each task in workflow G to VMs only when the priority constraint of workflow application submitted by users is satisfied to realize the shortest scheduling length of G. Its formalization is as follows:

$$\text{Minimize : Makespan,} \tag{6}$$

$$\text{Subject to : } \forall \xi_i \in N, \forall vm_j \in V. \tag{7}$$

4 The PMMS Algorithm

The core idea of PMMS is to use the $PM(\cdot)$ function to calculate the PM matrix of each task, thereby determining the priority order of tasks and affecting the corresponding VM allocation of each one. This section introduces the definition of the PM matrix, elaborates the task priority calculation phase and VM selection phase of PMMS, analyzes the PMMS's time complexity, and finally selects the DAG diagram shown in Fig. 1 to describe the flow of the PMMS algorithm.

4.1 Predict Makespan Matrix

The $PM(\cdot)$ function is used to construct a matrix named PM containing the values corresponding to each task-VM pair. The predicted completion time of the different task-VM pairs is expressed as a PM matrix, with the rows and columns representing tasks and VMs, respectively. The PM matrix has two important functions: (1) determine the rank value of each task to accordingly generate the task scheduling list in the task priority stage; and (2) select the most appropriate VM for each task in the scheduling list to minimize the makespan of the entire application.

Definition 1 $PM(\cdot)$ **function** $PM\left(\xi_i, vm_j\right)$ denotes the PM value when ξ_i is executed on vm_j. Calculated sequentially from down to up starting from the exit task, the PM value of ξ_i on vm_j is related to three properties: (1) the PM value of the successor task ξ_k on vm_γ; (2) the computational cost of the current task ξ_i on vm_γ; and (3) the average communication cost $\overline{c_{i,k}}$ between the current task ξ_i and its direct successor task ξ_k. $PM\left(\xi_i, vm_j\right)$ is calculated as

$$PM\left(\xi_i, vm_j\right) = \begin{cases} 0, & \text{if } \xi_i = \xi_{exit}; \\ \max\limits_{\xi_k \in succ(\xi_i)} \left\{ \min\limits_{vm_\gamma \in V} \left\{ PM\left(\xi_k, vm_\gamma\right) + w\left(\xi_i, vm_\gamma\right) + \overline{c_{i,k}} \right\} \right\}, & \text{otherwise,} \end{cases} \tag{8}$$

for the entry task, $PM\left(\xi_{exit}, vm_j\right) = w\left(\xi_{exit}, vm_j\right)$, when $vm_j = vm_\gamma$, $\overline{c_{i,k}}$=0.

4.2 Task Prioritizing Phase

Definition 2 Task rank value $rank_{PM}\left(\xi_i\right)$ The scheduling sequence of tasks in DAG is determined according to the value of task predict matrix priority $rank_{PM}$. The expression $rank_{PM}\left(\xi_i\right)$ for the $rank_{PM}$ value of ξ_i is defined as follows:

$$rank_{PM}(\xi_i) = \frac{\sum_{i=1}^{|V|} PM(\xi_i, vm_j)}{|V|}. \tag{9}$$

When the value is calculated only in accordance with (8) and (9), the $rank_{PM}$ value of the successor node for the current task may be greater than its own $rank_{PM}$ value. If the tasks are sorted by $rank_{PM}$ values only, the precedence constraint of task set may be violated. To solve this problem, a scaling-down policy inspired by the scaling-up method in the list-based heuristic makespan minimization scheduling algorithm called HMDS-Bl [32] is thus designed. (The scaling-up method of HMDS-Bl is that when the rank value of a task is less than or equal to the maximum of the rank values of all its successor tasks, the rank value of that task is scaled up in a specified proportion until it is greater than the maximum of the rank values of all its successor tasks.)

Definition 3 Scaling-down policy When the task node ξ_i and its direct successor node ξ_γ ($\xi_\gamma \in succ\left(\xi_i\right)$) meet the condition $rank_{PM}\left(\xi_i\right) \leq rank_{PM}\left(\xi_\gamma\right)$, the $PM\left(\xi_\gamma, vm_j\right)$ value of the successor node is scaled down in accordance with (10), thereby reducing the corresponding $rank_{PM}\left(\xi_\gamma\right)$ of the successor node

to satisfy the priority constraint. This is expressed as the following:

$$PM\left(\xi_\gamma, vm_j\right) = PM\left(\xi_\gamma, vm_j\right) \times \frac{rank_{PM}\left(\xi_i\right)}{rank_{PM}\left(\xi_\gamma\right) + \theta}, \tag{10}$$

where θ is a constant that takes the value of 1.0. This value was chosen after conducting a large number of tuning trials and carefully comparing the experimental results.

Algorithm 1 The TP Algorithm.

Input: $G = \{N, E, C, W\}, V$.
Output: $PM(\xi_i, vm_j)$, $rank_{PM}(\xi_i)$, and the priority list taskList.

1: **for** each task ξ_i in N **do**
2: **for** each vm_j in V **do**
3: Calculate $PM(\xi_i, vm_j)$ by (8);
4: **end for**
5: Calculate $rank_{PM}(\xi_i)$ by (9);
6: **end for**
7: **for** each task ξ_i in N **do**
8: **if** $rank_{PM}(\xi_i) \leq rank_{PM}(\xi_\gamma)$ && $\xi_\gamma \in succ(\xi_i)$ **then**
9: **for** each vm_j in V **do**
10: Recalculate $PM(\xi_\gamma, vm_j)$ by (10);
11: **end for**
12: Recalculate $rank_{PM}(\xi_i)$ by (9);
13: **end if**
14: **end for**
15: Sort the task in a taskList according to the descending order of $rank_{PM}(\xi_i)$ values;
16: return $PM(\xi_i, vm_j)$, $rank_{PM}(\xi_i)$, taskList.

The task priority (TP) algorithm is shown in Algorithm 1. Lines 1–14 above recursively calculate the $rank_{PM}$ value of each task. Lines 2–4 calculate the $PM(\xi_i, vm_j)$ value of each task on different VMs. Line 5 obtains the initial $rank_{PM}$ value of each task. When the $rank_{PM}(\xi_i)$ value of the ξ_i is less than or equal to the $rank_{PM}(\xi_\gamma)$ value of its direct successor node ξ_γ, (i.e., $rank_{PM}(\xi_i) \leq rank_{PM}(\xi_\gamma)$ and $\xi_\gamma \in succ(\xi_i)$), then $PM(\xi_\gamma, vm_j)$ and the $rank_{PM}(\xi_\gamma)$ of ξ_γ is updated in accordance with (10) in lines 7-14. Line 15 obtains the task priority queue taskList based on the updated $rank_{PM}(\xi_\gamma)$ values in descending order. Line 16 returns the updated $PM(\xi_i, vm_j)$, $rank_{PM}(\xi_\gamma)$ and taskList.

Table 4 shows the $PM(\xi_i, vm_j)$ matrix of different task nodes calculated in accordance with (8) under Algorithm 1 and the priority $rank_{PM}(\xi_i)$ of each task as calculated by (9). The priority list taskList in descending order of $rank_{PM}$ is $\{\xi_1, \xi_4, \xi_5, \xi_3, \xi_2, \xi_6, \xi_9, \xi_8, \xi_7, \xi_{10}\}$.

4.3 VM Selection Phase

Definition 4 Optimistic schedule length (*OSL*) *OSL* is the sum of $EFT(\xi_i, vm_j)$ and $PM(\xi_i, vm_j)$. The $OSL(\xi_i, vm_j)$ value of ξ_i on vm_j can be calculated by

$$OSL\left(\xi_i, vm_j\right) = EFT\left(\xi_i, vm_j\right) + PM\left(\xi_i, vm_j\right). \tag{11}$$

Each task-VM assignment is associated with three attributes of the task on the corresponding VM, including *EST*, *EFT*, and *OSL*. In this section, the *OSL* value of each task ξ_i on vm_j is first computed. The VM with the smallest *OSL* value is selected to execute the corresponding task.

4.4 PMMS

The pseudo code of PMMS is shown in Algorithm 2. Line 1 calls Algorithm 1. Lines 2–13 are the core part of PMMS. In lines 4–10, every VM is traversed for each candidate task. Line 5 calculates the *EFT* of the candidate task on the current VM. Line 6 calculates the *OSL* value of the task on the current VM. Lines 7–9 then select the VM with the smallest *OSL* value and obtain the *EFT* of the task on the VM that meets task priority constraints. Line 12 updates the taskList. When all task nodes are finished, the actual makespan of G is calculated in line 14.

4.5 Time Complexity Analysis

When a user submits a G with $|N|$ task to a cloud center with a resource pool, $|V|$ VMs are thus obtained. In Algorithm 1, the time complexity for computing the *PM* matrix in lines 2–4 is $O\left(|N| \times |V|\right)$. Thus, the time complexity of lines 1-6 should be $O\left(|N|^2 \times |V|\right)$. In lines 7–14, the algorithm compares the $rank_{PM}$ values between tasks and updates the $PM\left(\xi_\gamma, vm_j\right)$ and $rank_{PM}\left(\xi_i\right)$ values, whose running time is $O\left(|N|^2 \times |V|\right)$. Therefore, Algorithm 1 has a time complexity of order $O\left(|N|^2 \times |V|\right)$. In

Table 4 *PM* matrix of each task in Fig. 1 under PMMS

$Task_i$	$PM\ (\xi_i, vm_1)$	$PM\ (\xi_i, vm_2)$	$PM\ (\xi_i, vm_3)$	$rank_{PM}(\xi_i)$
ξ_1	161	147	147	151.67
ξ_2	126	126	107	119.67
ξ_3	130	128	107	121.67
ξ_4	125	123	133	127.00
ξ_5	117	130	128	125.00
ξ_6	114	126	111	117.00
ξ_7	80	97	94	90.33
ξ_8	98	107	94	98.00
ξ_9	95	107	94	98.67
ξ_{10}	47	78	63	62.67

Algorithm 2 PMMS.

Input: $PM(\xi_i, vm_j)$, $rank_{PM}(\xi_i)$, $G = \{N, E, C, W\}$, V.
Output: Makespan
1: Call the Algorithm 1;
2: **while** taskList is not empty **do**
3: $vm' \leftarrow vm_0$
4: **for** each vm_j in V **do**
5: Calculate $EFT(\xi_i, vm_j)$ by (4);
6: Calculate $OSL(\xi_i, vm_j)$ by (11);
7: **if** $OSL(\xi_i, vm_j) \leq OSL(\xi_i, vm')$ **then**
8: $OSL(\xi_i, vm') \leftarrow OSL(\xi_i, vm_j)$;
9: **end if**
10: **end for**
11: Assign ξ_i to vm';
12: Update taskList;
13: **end while**
14: Makespan $\leftarrow EFT(\xi_{exit}, vm_j)$;
15: return Makespan.

Algorithm 2, lines 4–10 take the most time to calculate the *EFT* value and *OSL* value with a time complexity of $O\left(|N| \times |V|\right)$. The total time complexity for Algorithm 2 is also $O\left(|N|^2 \times |V|\right)$. Hence, PMMS' time complexity is $O\left(|N|^2 \times |V|\right)$.

4.6 Motivational Example

For the DAG in Fig. 1, PMMS works as follows. Step 1 calculates the *PM* value of each task node on different VMs using (8). Step 2 then calculates the $rank_{PM}\ (\xi_i)$ value of each task using (9). Step 3 compares the initial $rank_{PM}\ (\xi_i)$ of the task node and updates the corresponding *PM* and $rank_{PM}$ values according to (10) and (9), respectively. For Fig. 1, the $rank_{PM}$ value of the immediate successor node of each task is smaller than its predecessor node, hence no update is needed. Step 4 sorts the tasks by the $rank_{PM}\ (\xi_i)$ value obtained in Step 3 in decreasing order and obtains the task priority list $\{\xi_1, \xi_4, \xi_5, \xi_3, \xi_2, \xi_6, \xi_9, \xi_8, \xi_7, \xi_{10}\}$. Step 5 computes the *EFT* of each task on the available VM node according to (3) and (4). Step 6 calculates the value of *OSL* for each task on the VM. Step 7 assigns each task to the VM with the smallest *OSL* value. The results of Fig. 1 under PMMS are shown in Table 5.

Task ξ_3 has the smallest *OSL* value on vm_3 with $OSL\ (\xi_3, vm_3) = 153$. Therefore, ξ_3 is assigned to vm_3, at which point ξ_3 runs on vm_3 with $EFT\ (\xi_3, vm_3) = 46$. Task ξ_8 has the smallest *OSL* value on vm_2 with $OSL\ (\xi_8, vm_2) = 201$, so ξ_8 runs on vm_2 as well with $EFT\ (\xi_8, vm_2) = 103$. The scheduling results of the DAG shown in Fig. 1 under HEFT, PEFT, PPTS, and PMMS are shown in Fig. 3, with corresponding makespans of 183, 168, 186, and 161, respectively. Compared with HEFT, PEFT and PPTS, the PMMS algorithm reduces the makespan by 12.02%, 4.17%, and 13.44%, respectively.

5 Experiments

This section presents in detail the three algorithms to be compared, the experimental environment, experimental setting, and evaluation metrics of workflow algorithms. Four real scientific workflows are selected to test the performance of PMMS with HEFT, PEFT, and PPTS algorithms.

Table 5 Intermediate results of each task in Fig. 1 under PMMS

$Task_i$	*EFT*			*OSL*			*vm* assigned
	vm_1	vm_2	vm_3	vm_1	vm_2	vm_3	
ξ_1	51	**21**	36	212	**168**	183	vm_2
ξ_4	57	**37**	65	182	**160**	198	vm_2
ξ_5	**54**	80	66	**171**	210	194	vm_1
ξ_3	138	68	**46**	268	196	**153**	vm_3
ξ_2	122	108	**55**	248	234	**162**	vm_3
ξ_6	70	91	**68**	184	217	**179**	vm_3
ξ_9	119	131	**99**	214	238	**193**	vm_3
ξ_8	134	**103**	134	232	**201**	232	vm_2
ξ_7	**102**	125	130	**182**	222	224	vm_1
ξ_{10}	**161**	197	182	**208**	275	245	vm_1

Bold values indicate the best results obtained by PMMS in terms of EFT/OSL

5.1 Comparative Algorithms

Because the PMMS, HEFT, PEFT, and PPTS algorithms have the same application and system models (with the goal being to minimize the makespan), all algorithms are compared herein. Their working principle can be briefly summarized as follows.

HEFT This is a well-known and the most classical heuristic list scheduling strategy. First, HEFT starts from the exit task, traverses upward to the entry task, and computes the upward rank values of task nodes in turn. Second, the task priority queue is generated based on the result of the descending order of the upward rank values. Third, the EFT of the task on each VM is computed according to the order of the task priority queue. Fourth, the VM with the smallest *EFT* value is selected to schedule the corresponding task.

PEFT The highlight of PEFT is an Optimistic Cost Table (OCT) based policy. In the task prioritization phase, the $OCT(\xi_i, vm_j)$ value of task ξ_i on vm_j is first computed as in (12). Then the rank value $rank_{oct}(\xi_i)$ of each task is calculated using (13) and ranked in descending order of $rank_{oct}(\xi_i)$. In the VM selection phase, the optimistic EFT (O_{EFT}) of each task is cal-

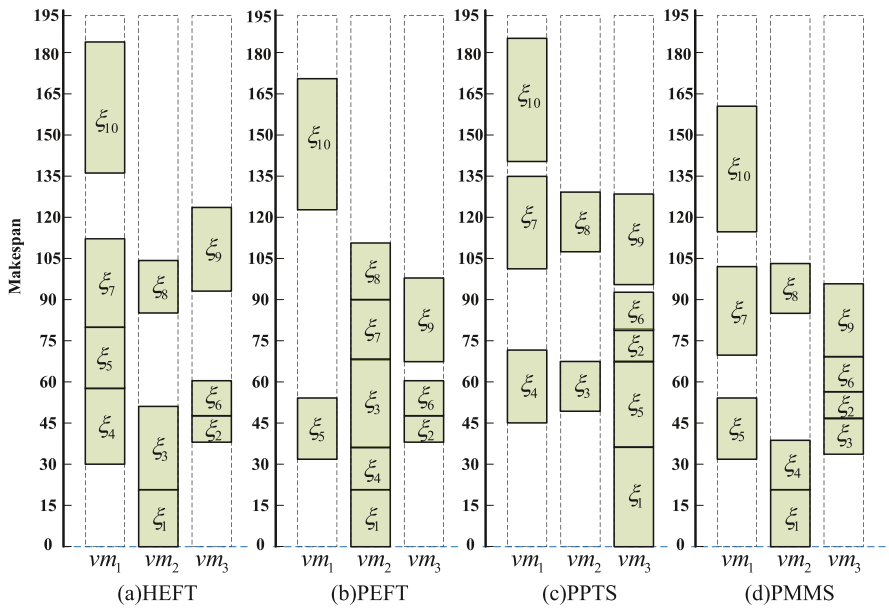

Fig. 3 Scheduling result of the workflow Fig. 1

culated using (14), and the task is assigned to the VM with the smallest O_{EFT} value.

$$OCT\left(\xi_i, vm_j\right) = \max_{\xi_k \in succ(\xi_i)} \left\{ \min_{vm_\gamma \in P} \left\{ OCT\left(\xi_k, vm_\gamma\right) + w\left(\xi_k, vm_\gamma\right) + \overline{c_{i,k}} \right\} \right\}. \tag{12}$$

$$rank_{oct}(\xi_i) = \frac{\sum_{j=1}^{|V|} OCT(\xi_i, vm_j)}{|V|}. \tag{13}$$

$$O_{EFT}\left(\xi_i, vm_j\right) = EFT\left(\xi_i, vm_j\right) + OCT(\xi_i, vm_j). \tag{14}$$

PPTS PPTS improves both the task priority phase and the VM selection phase. In the task priority phase, (15) and (16) are used to calculate the rank value $rank_{PCM}(\xi_i)$ of task ξ_i. For the exit task in (15), $PCM(\xi_{exit}, vm_j) = w(\xi_{exit}, vm_j)$. Then, the tasks are ranked in descending order according to the $rank_{PCM}(\xi_i)$ value. In the VM selection phase, the $LA_{EFT}(\xi_i, vm_j)$ is calculated in (17) and the VM with the smallest $LA_{EFT}(\xi_i, vm_j)$ value is selected to schedule corresponding tasks.

$$PCM\left(\xi_i, vm_j\right) = \max_{\xi_k \in succ(\xi_i)} \left\{ \min_{vm_\gamma \in P} \left\{ PCM\left(\xi_k, vm_\gamma\right) + w\left(\xi_k, vm_\gamma\right) + w\left(\xi_i, vm_\gamma\right) + \overline{c_{i,k}} \right\} \right\}. \tag{15}$$

$$rank_{PCM}(\xi_i) = \frac{\sum_{j=1}^{|V|} PCM(\xi_i, vm_j)}{|V|}. \tag{16}$$

$$LA_{EFT}(\xi_i, vm_j) = EFT(\xi_i, vm_j) + PCM(\xi_i, vm_j). \tag{17}$$

5.2 Experimental Environment

The experimental environment is similar to that in extant literature [33–35] to simulate real application scenarios. Said experiments are conducted based on the Amazon EC2 instance, thus providing users with more than 35 different instance types and 8 zones. We applied for 21 VMs in the Amazon EC2 cloud to test the algorithm's performance. Each VM can be configured with a different type and frequency of CPU, 2 to 16 GB memory, a CentOS 7.3 operating system, and Python as the experimental development language. The experimental environmental parameters are as follows: 0.01 h $\leq w_{i,k} \leq$ 192 h, 0.01 h $\leq c_{i,j} \leq$ 30 h, $\overline{B}$ = 20 Mb/s, and the average latency of VMs to obtain services is 97 s [15].

In order to evaluate the performance of PMMS and the compared algorithms, four typical real scientific workflows are used whose structures are illustrated in Fig. 4. Figure 4(a) presents an example of the application of simple Epigenomics (EP) [36] in biogenetics. EP is typically a highly-parallel application with multiple communication channels operating different blocks of data that are not directly connected to each other. Figure 4(b) shows a Gaussian Elimination (GE) [8] parallel workflow application with $\mu = 5$, where μ is the dimension of the GE matrix. The number of task nodes (S_μ) and μ in GE satisfy the following equation: $S_\mu = (\mu^2 + \mu - 2)/2$. Figure 4(c) illustrates the LIGO inspiral Analysis Workflow, while Fig. 4(d) shows an example of the Montage workflow, usually applied in astronomy [36]. Without loss of generality, these four different types of real scientific workflows are randomly generated using a number of tasks ranging from 18 to 1,084.

5.3 Experimental Evaluation and Settings

5.3.1 Evaluation Metrics

The scheduling length ratio (SLR) [37,38], makespan [7,21], and execution time [39] are used in the experiments as performance metrics to evaluate PMMS and compared algorithms. The SLR is defined as:

$$\text{SLR} = \frac{\text{Makespan}}{\sum_{\xi_i \in CP} \min_{vm_j \in V}\{w_{i,j}\}}, \tag{18}$$

where the denominator of (18) is the sum of the tasks with the minimum computational costs in G and $\xi_i \in CP$ (CP refers to the critical path of G). As the denominator stays the same for different scheduling algorithms

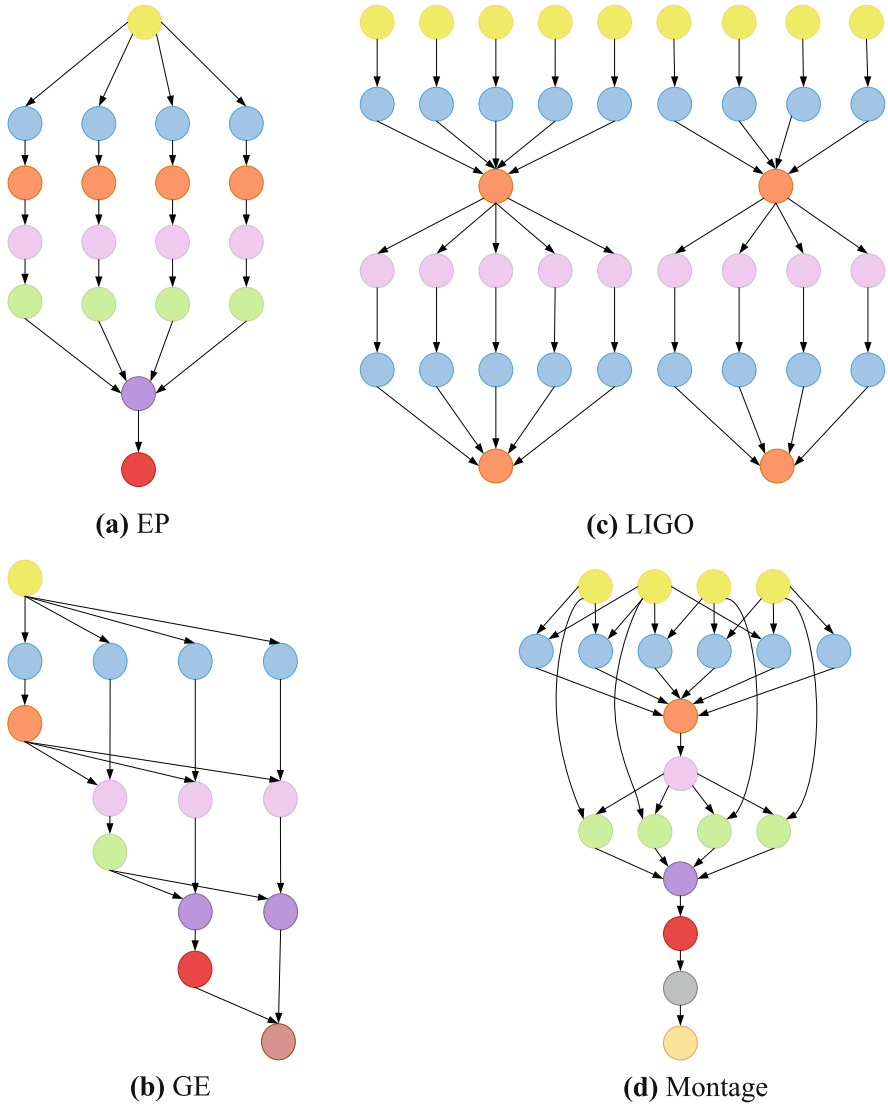

Fig. 4 Architecture diagram of four scientific workflow applications

when considering the same application and the set of VMs, the smaller the makespan is, the smaller SLR is. Consequently, a smaller value of SLR indicates a better performance of the algorithm.

The execution time of workflow is calculated as the total computational time taken by all tasks running on the respective VM.

To avoid occasionality, the communication computing ratio (CCR) [8], which is the ratio of the average cost of communication to computation, is also noted when selecting the task set. In a DAG, a smaller CCR value means that it is a computation intensive application, while a larger CCR value means that it is a communication intensive application. In the experiment, different results are obtained by adjusting the value of CCR and averaging all experimental results.

5.3.2 Experimental Settings

The following three sets of experiments with different variables are set in EP, GE, LIGO, and Montage parallel application workflows:

(1) The scale of tasks is variable. The workflow varies in size while the number of VMs is kept at 3 and other parameters remain consistent to test the average SLR of the four algorithms. The scale of tasks for each real-world workflow used in this section is as follows:

(i) EP: 19, 51, 103, 523, 1,003.
(ii) GE: 27, 53, 298, 494, 1,033.
(iii) LIGO: 40, 94, 274, 544, 1,084.
(iv) Montage: 18, 66, 126, 606, 1,056.

(2) The scale of VMs is variable. The number of tasks of EP, GE, LIGO, and Montage parallel workflows is 1,003, 1,033, 1,084, and 1,056, respectively. The experiments utilize the following numbers of VMs: 3, 6, 9, 12, 15, 18, and 21. We assess the average SLR and execution time of the four algorithms.
(3) The scale of tasks and the value of CCR are variable. The workflow's size varies while maintaining the number of VMs at 21, and all other parameters remain constant to assess the SLR of the four algorithms. The scale of tasks for each real-world workflow employed in this scenario is identical to that mentioned in (1). The experiments utilize the following values of CCR: 0.5, 1.0, 2.0, 5.0.

5.4 Experimental Results

5.4.1 Varying Number of Tasks

We compare the average SLR of PMMS, PEFT, HEFT, and PPTS algorithms in four kinds of workflow applications with a diverse number of tasks. By adjusting the CCR values, the number of tasks of the five different sizes is tested 10 times, subsequently averaging the results thereafter. All experimental data in this section are obtained by testing various sizes of the application while keeping the number of VMs at 3. Figure 5 shows the compared results.

Taking the number of tasks in EP as a variable, Fig. 5(a) depicts the average SLR values of the four algorithms in parallel application of EP with five different task numbers. It shows that the average SLR of PEFT, HEFT, and PPTS algorithms is greater than that of PMMS. For $|N| = 19$, then the average SLR of PMMS, compared with PEFT, HEFT, and PPTS, is reduced by 14.18%, 4.72%, and 4.72%, respectively. Meanwhile, for $|N| = 51$, the SLR values of all four algorithms are greater than 8.0. In the remaining experiments with different size of tasks, the SLR values of the four algorithms are all less than 2.5. It is due to the fact that the sum of the minimum computational costs of the tasks on the critical path in G is tiny for the four algorithms when processing a workflow application with $|N| = 51$. PMMS algorithm has a good performance in reducing completion time – this is because when PMMS algorithm calculates the PM matrix, the PM value of each task considers communication costs for the task and its successor nodes, thereby making the task scheduling sequence more reasonable. This fully considers the relative importance of each task, thus effectively improving the scheduling efficiency. However, the HEFT algorithm overlooks the characteristics of each task's successor node, while PEFT ignores the relative importance of each task.

Five GE parallel applications of different scales are tested in this experiment. Experimental results are shown in Fig. 5(b). It shows that when the scale of the task node is less than or equal to 53, there is less than 1% difference between the average SLR of PMMS, PEFT, HEFT, and PPTS algorithms, and that their average SLR value is less than 1.5 due to the characteristic of GE, which is composed of GE matrices. Especially in small-scale GE applications, the difference is indistinct. However, with the continuous expansion of task scale, the average SLR of the four algorithms also keeps increasing and presents significant differences. For example, with a size of 494 nodes, the average SLR of PMMS, compared to the PEFT, HEFT, and PPTS algorithms, is 7.83%, 7.17%, and 8.48% lower, respectively. The PMMS algorithm therefore evidently improves makespan reduction.

Figure 5(c) depicts the experimental results of LIGO parallel applications with five different sizes of under the four algorithms. When $|N| = 94$, the SLR value of all three algorithms is less than 2.5. The reason for this result may be that the sum of the minimum computational costs of the tasks on the critical path in G is relatively large. In the remaining experiments with different sizes of tasks, the SLR values of the four algorithms are all greater than 6.5. When the number of task nodes is 544, of PMMS is superior than PEFT, HEFT, and PPTS by 6.04%, 4.20%, and 6.12%, respectively in terms of the average SLR. The PMMS algorithm therefore effectively decreases the makespan of LIGO applications.

Figure 5(d) presents the experimental results of five different tasks with the Montage applications under the four compared algorithms. When the number of tasks in the Montage application equals to 1,056, the average SLR of the four algorithms reaches maximum and is all greater than 85.0. For the Montage application with 606 tasks however, the proposed PMMS outperforms

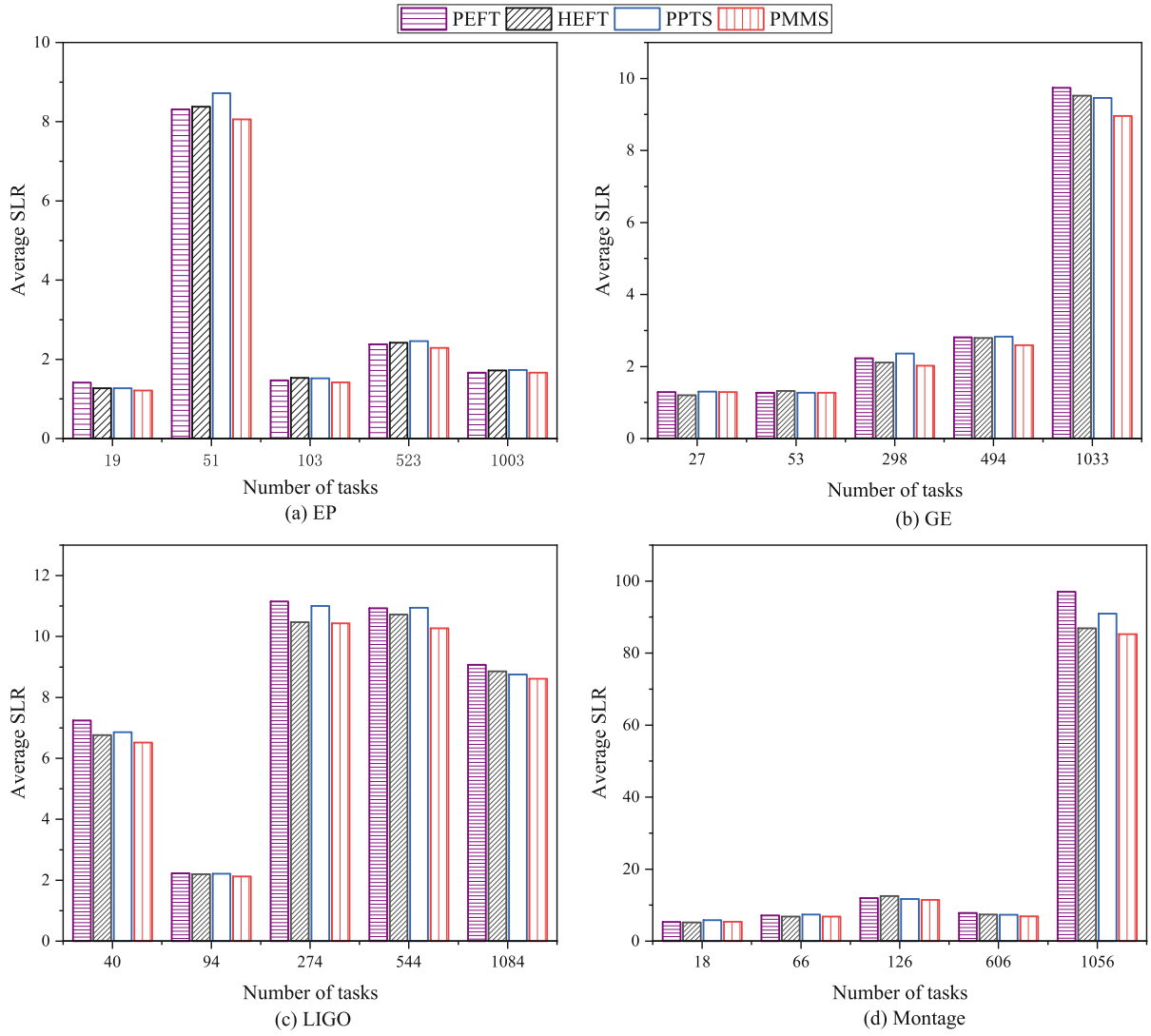

Fig. 5 Average SLR comparison of four workflows with different numbers of tasks

PEFT, HEFT, and PPTS in terms of the average SLR by 11.52%, 6.62%, and 5.86%, respectively. Clearly, PMMS is superior to PEFT, HEFT, and PPTS algorithms in decreasing the makespan.

In all workflow parallel applications, the makespan of workflows increases following the scale of the workflow.

5.4.2 Different Number of VM

To reflect the rigor and rationality of the experiment setting used, the number of VMs is used as the variable while the execution cost of each task on different VMs remained unchanged.

(1) Average SLR

The experimental results depicted in Fig. 6 illustrates the average SLR of the four algorithms across four workflow applications, with the number of VMs as a variable. The average SLR is influenced by both the makespan and the minimum computational overhead sum at the critical path task of the workflow. As the number of VMs increases, there is a gradual decrease in the makespan of the workflow application. However, the critical path of the workflow application experiences irregular variations when the number

of VMs is altered. Consequently, irrespective of the specific workflow application, the average SLR value exhibits an inconsistent trend. Figure 6(a) shows the experimental results of the average SLR for four algorithms, while altering the number of VMs with a task quantity of 1,003 for EP. Notably, PMMS consistently outperforms the other three algorithms in terms of average SLR, regardless of the VM count employed. When $|V| = 6$, the PMMS algorithm achieves a noteworthy reduction in average SLR by 5.70%, 2.52%, and 3.12% when compared to HEFT, PEFT, and PPTS, respectively. The average SLR experimental results of the four algorithms on the GE workflow are depicted in Fig. 6(b). For $|V| = 9$, achieves significant reductions in average SLR, outperforming HEFT, PEFT, and PPTS by 9.25%, 11.34%, and 5.80%, respectively. Hence, PMMS excels in effectively reducing the average SLR on GE. The average SLR results for the four algorithms applied to LIGO with varying VM quantities are shown in Fig. 6(c). When $|V|$ =3, 6, and 15, PMMS demonstrates substantial superiority over the other three algorithms in reducing average SLR. For $|V|$ values of 9, 12, 18, and 21, PMMS exhibits a slight advantage over PEFT and PPTS in reducing average SLR, while clearly surpassing HEFT. In Fig. 6(d), the average SLR of the four algorithms on Montage is depicted for vary-

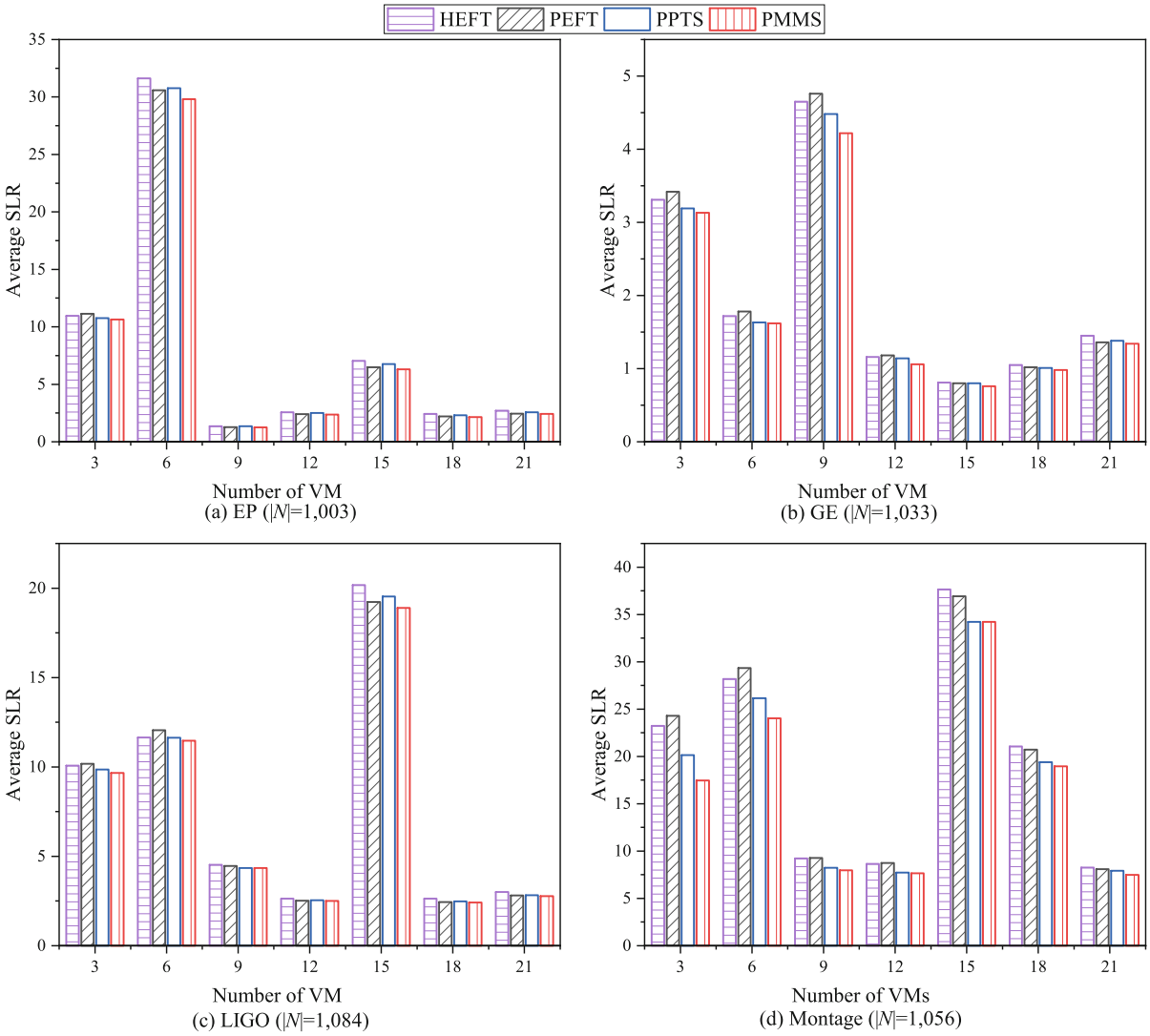

Fig. 6 Average SLR comparison of four workflows with different numbers of VMs

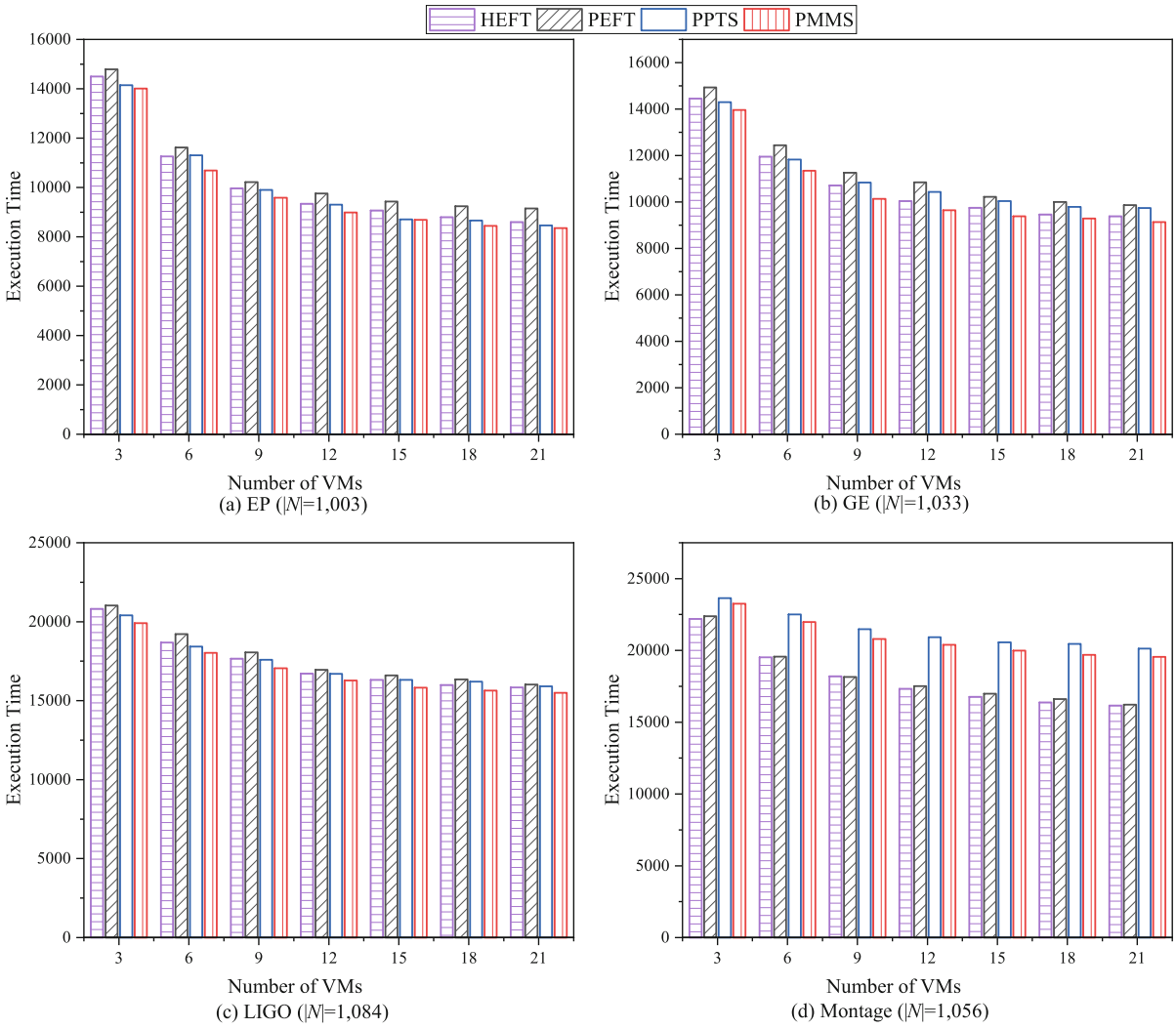

Fig. 7 Execution Time comparison of four workflows with different numbers of VMs

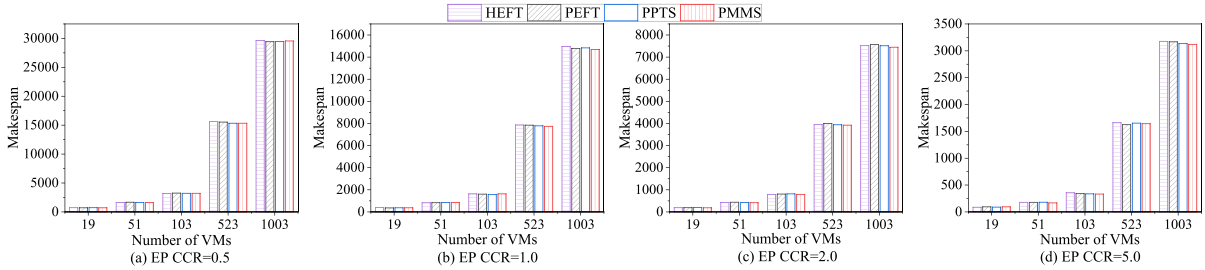

Fig. 8 Makespan comparison of four algorithms with different values of CCR in EP

ing quantities of VMs. When the number of VMs is 3, PMMS outperforms the PEFT, HEFT, and PPTS algorithms by 24.91%, 28.16%, and 13.31%, respectively, in terms of SLR. Furthermore, when the VM scale reaches 6, the average SLR of PMMS decreases by 14.76%, 18.13%, and 8.11% when compared to the PEFT, HEFT, and PPTS algorithms, respectively.

It can be concluded that PMMS surpasses the other comparative algorithms in reducing the average SLR. For workflows of identical size and with the same number of VMs, the minimum computational overhead of the critical path nodes remains constant. Consequently, the PMMS algorithm excels in reducing the makespan of the workflow application, outperforming the HEFT, PEFT, and PPTS algorithms. Additionally, PMMS stands out with exceptional performance, particularly in the Montage workflow application.

(2) Execution Time

As shown in Fig. 7(a), we conduct experiments with a fixed number of 1,003 EP tasks while varying the number of VMs. As the number of VMs increases, the execution time of all four algorithms consistently decreases. Notably, PMMS exhibits the shortest execution times. For $|V| = 6$, PMMS achieves a reduction in execution time by 5.10%, 8.01%, and 5.52% compared to HEFT, PEFT, and PPTS, respectively. This demonstrates that PMMS significantly contributes to saving execution time and enhancing scheduling efficiency in EP. Figure 7(b) illustrates the execution times of the four algorithms on GE for various numbers of VMs. It is evident that as the number of VMs increases, there is a consistent decrease in the execution time. Notably, at $|V| = 12$, PMMS achieves a reduction in execution time by 3.95%, 11.04%, and 7.52% compared to HEFT, PEFT, and PPTS, respectively. The execution time of the four algorithms on LIGO for various numbers of VMs is depicted in Fig. 7(c), where it is evident that the execution time of the workflow consistently decreases as the number of VMs increases. Notably, PMMS stands out as significantly superior to the other three comparison algorithms in reducing execution time. When $|V| = 9$, the PMMS algorithm achieves execution times that are 3.44%, 5.53%, and 3.04% lower than those of HEFT, PEFT, and PPTS, respectively. The execution time of the four algorithms in Montage for various numbers of VMs is illustrated in Fig. 7(d), demonstrating a decrease in execution time as the number of VMs increases. Although the execution time of the PMMS algorithm is lower than that of PPTS, it surpasses HEFT and PEFT. This outcome signifies the practical applicability of the PMMS algorithm. PMMS demonstrates superior performance across EP, GE, and LIGO workflows.

In summary, regardless of the type of workflow application, the rate of decline in execution time slows down after reaching $|V| = 15$. This indicates that as the number of VMs increases, its impact on execution time diminishes. The PMMS algorithm consistently outperforms HEFT, PEFT, and PPTS in reducing execution time and enhancing scheduling efficiency.

5.4.3 Different Value of CCR

Figure 8 illustrates the makespan of the four algorithms as the CCR value of the EP workflow is altered, with $|V| = 21$. Based on a comprehensive observation, it can be deduced that there is a decreasing trend in the peak (maximum) makespan of the four algorithms as the CCR value increases from 0.5 to 5.0. In the case of $|N| = 1,003$, the makespan of PMMS in Fig. 8(a) is 29,568, while the makespan of PMMS in Fig. 8(b), (c), and (d) is 14,695, 9,735, and 7,447, respectively. Figure 9 shows the makespan of GE across four algorithms. Notably, as the CCR

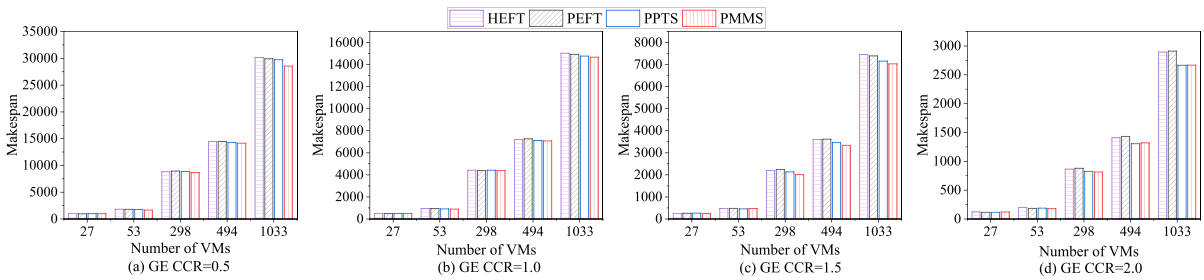

Fig. 9 Makespan comparison of four algorithms with different values of CCR in GE

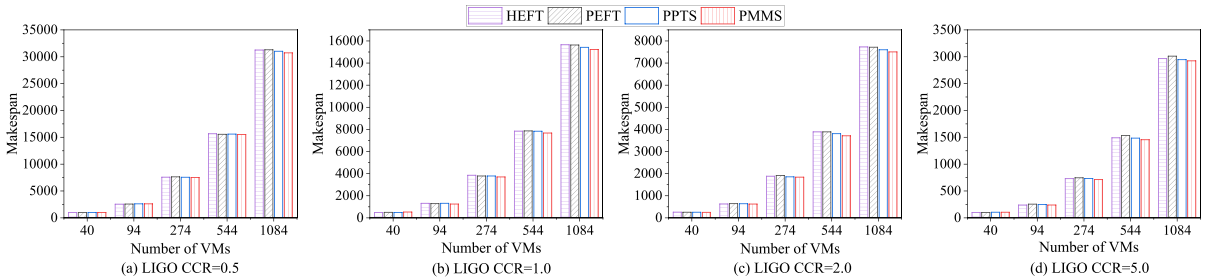

Fig. 10 Makespan comparison of four algorithms with different values of CCR in LIGO

value ascends from 0.5 to 5.0, the peak makespan of all four algorithms undergoes a gradual reduction. For $|N| = 1,033$, the makespan of PMMS in Fig. 9(a) reaches 28,556, while the makespan of PMMS in Fig. 9(b), (c), and (d) achieve values of 14,677, 9,544, and 7,035, respectively. Figure 10 illustrates the makespan of the four algorithms as the CCR value of the LIGO workflow varies, with $|V| = 21$. When $|N| = 544$, the makespan of PMMS in Fig. 10(a) is 15,514, whereas the makespan of PMMS in Fig. 10(b), (c), and (d) amounts to 7,675, 4,957, and 3,713 correspondingly. Figure 11 depicts the makespan of the four algorithms in Montage. When considering $|N| = 606$, the makespan of PMMS in Fig. 11(a) is 9,917. In contrast, the makespan of PMMS in Fig. 11(b), (c), and (d) stands at 5,399, 3,441, and 2,650, respectively. A lower CCR value signifies a computation intensive application, whereas a higher CCR value indicates a communication intensive application. While keeping all other variables constant, it is generally observed that computation intensive applications tend to demonstrate higher makespans compared to their communication intensive counterparts. Furthermore, it is evident from Figs. 9–11 that regardless of variations in CCR and the number of tasks, the makespan of PMMS remains lower than that of the other three comparison algorithms. The PMMS algorithm performs well in reducing the scheduling length of workflow applications. Additionally, when comparing the makespan values of EP, GE, LIGO, and Montage in the four algorithms, it becomes apparent that the makespan of EP and LIGO, which are characterized by high parallelism and intricate task dependencies, tends to be higher than that of GE and Montage, which exhibit lows parallelism and simple task dependencies.

6 Conclusions

We design an efficient workflow scheduling algorithm (called PMMS) to promote efficiency in HCS. The PMMS algorithm includes a task priority calculation phase and a VM selection phase and calculates the $rank_{PM}$ value of each task by using PM matrix in the task priority calculation phase and thereafter obtains the task scheduling list in descending order according to $rank_{PM}$. In the VM selection phase, OSL is computed for each task according to the PM matrix. Afterwards, the VM with the minimum OSL is selected for it, thereby minimizing the makespan of workflow applications. Through extensive comparison experiments of EP, GE, LIGO, and Montage with

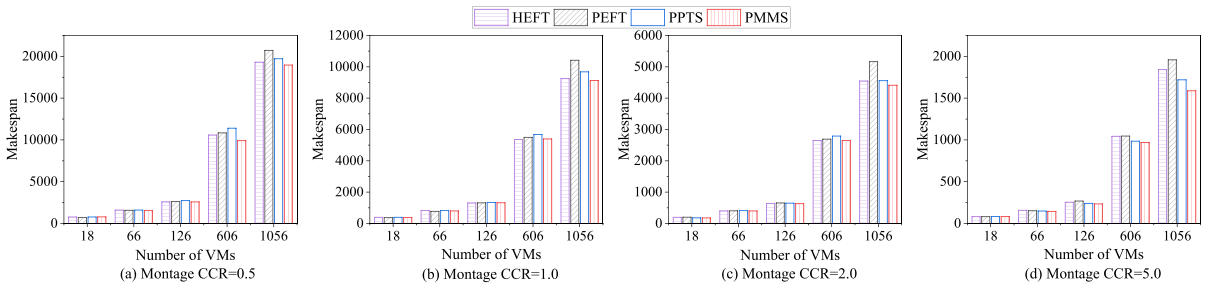

Fig. 11 Makespan comparison of four algorithms with different values of CCR in Montage

different scales, PMMS greatly reduces SLR compared with the advanced HEFT, PEFT, and PPTS algorithms. Compared with the advanced HEFT, PEFT, and PPTS algorithms, the makespan of PMMS algorithm in the Montage workflow application is reduced by 11.60%, 7.47%, and 8.09%, respectively, thereby significantly improving the scheduling efficiency of workflow parallel application. Future succeeding studies are encouraged to explore the impact of different θ values on the performance of our proposed method and focus on combining the idea of designing a heuristic scheduling algorithm by using predict matrix with task offloading and energy consumption reduction in edge computing.

Author contributions Longxin Zhang: Methodology, Data curation, Formal analysis, Software, Visualization, Writing - original draft, Writing - review & editing, Supervision. Minghui Ai: Conceptualization, Project administration, Writing - original draft. Runti Tan: Resources, Validation. Junfeng Man: Data curation, Formal analysis, Software. Xiaojun Deng: Data annotation, Investigation, Visualization. Keqin Li: Writing - review & editing, Supervision.

Funding This work was partially funded by the National Key R&D Program of China (Grant No. 2018YFB1003 401), the Natural Science Foundation of Hunan Province (Grant Nos. 2023JJ50204, 2022JJ50002, 2021JJ50049), the Scientific Research Foundation of Hunan Provincial Education Department (Grant Nos. 23B0562, 22B0598) and the National Natural Science Foundation of China (Grant Nos. 61702178, 62072172).

Data Availability The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

This work is original and not have been published elsewhere in any form or language.

Competing interests The authors declare that they have no known conflict financial interests or personal relationships that could have appeared to influence the work reported in this study.

References

1. Li, K.: Profit maximization in a federated cloud by optimal workload management and server speed setting. IEEE Transactions on Sustainable Computing. **7**(3), 668–680 (2021)
2. Sharma, P., Jadhao, V.: Molecular dynamics simulations on cloud computing and machine learning platforms. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), pp. 751–753 (2021). IEEE
3. Muniswamaiah, M., Agerwala, T., Tappert, C.C.: Green computing for internet of things. In: 2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 182–185 (2020). IEEE
4. Kavanagh, R., Djemame, K., Ejarque, J., Badia, R.M., Garcia-Perez, D.: Energy-aware self-adaptation for application execution on heterogeneous parallel architectures. IEEE Transactions on Sustainable Computing. **5**(1), 81–94 (2019)
5. Jiang, J., Lin, Y., Xie, G., Fu, L., Yang, J.: Time and energy optimization algorithms for the static scheduling of multiple workflows in heterogeneous computing system. Journal of Grid Computing. **15**, 435–456 (2017)
6. Zhang, L., Zhou, L., Salah, A.: Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments. Inf. Sci. **531**, 31–46 (2020)
7. Huang, J., Li, R., Jiao, X., Jiang, Y., Chang, W.: Dynamic dag scheduling on multiprocessor systems: reliability, energy, and makespan. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 3336–3347 (2020)
8. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
9. Ilavarasan, E., Thambidurai, P., Mahilmannan, R.: High performance task scheduling algorithm for heterogeneous computing system. In: ICA3PP, vol. 2005, pp. 193–203 (2005). Springer
10. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**(3), 682–694 (2013)
11. Djigal, H., Feng, J., Lu, J.: Task scheduling for heterogeneous computing using a predict cost matrix. In: Workshop Proceedings of the 48th International Conference on Parallel Processing, pp. 1–10 (2019)
12. Li, K.: Design and analysis of heuristic algorithms for energy-constrained task scheduling with device-edge-cloud fusion. IEEE Transactions on Sustainable Computing **01**, 1–13 (2022)
13. Djigal, H., Feng, J., Lu, J., Ge, J.: Ippts: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **32**(5), 1057–1071 (2020)
14. Rizvi, N., Ramesh, D.: Hbdcws: heuristic-based budget and deadline constrained workflow scheduling approach for heterogeneous clouds. Soft. Comput. **24**(24), 18971–18990 (2020)
15. Kelefouras, V., Djemame, K.: Workflow simulation and multi-threading aware task scheduling for heterogeneous computing. Journal of Parallel and Distributed Computing **168**, 17–32 (2022)
16. Youness, H., Omar, A., Moness, M.: An optimized weighted average makespan in fault-tolerant heterogeneous mpsocs. IEEE Trans. Parallel Distrib. Syst. **32**(8), 1933–1946 (2021)
17. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans. Parallel Distrib. Syst. **30**(1), 29–44 (2018)
18. Arabnejad, H., Barbosa, J.G.: A budget constrained scheduling algorithm for workflow applications. Journal of Grid Computing **12**, 665–679 (2014)
19. Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., Li, K.: Efficient task scheduling for budget constrained parallel applications

on heterogeneous cloud computing systems. Futur. Gener. Comput. Syst. **74**, 1–11 (2017)
20. Quan, Z., Wang, Z.-J., Ye, T., Guo, S.: Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **31**(5), 1165–1182 (2019)
21. Peng, J., Li, K., Chen, J., Li, K.: Reliability/performance-aware scheduling for parallel applications with energy constraints on heterogeneous computing systems. IEEE Transactions on Sustainable Computing **7**(3), 681–695 (2022)
22. Chen, W., Xie, G., Li, R., Li, K.: Execution cost minimization scheduling algorithms for deadline-constrained parallel applications on heterogeneous clouds. Clust. Comput. **24**, 701–715 (2021)
23. Liu, J., Ren, J., Dai, W., Zhang, D., Zhou, P., Zhang, Y., Min, G., Najjari, N.: Online multi-workflow scheduling under uncertain task execution time in iaas clouds. IEEE Transactions on Cloud Computing **9**(3), 1180–1194 (2019)
24. Chen, Y., Xie, G., Li, R.: Reducing energy consumption with cost budget using available budget preassignment in heterogeneous cloud computing systems. IEEE Access. **6**, 20572–20583 (2018)
25. Zhang, L., Wang, L., Wen, Z., Xiao, M., Man, J.: Minimizing energy consumption scheduling algorithm of workflows with cost budget constraint on heterogeneous cloud computing systems. IEEE Access **8**, 205099–205110 (2020)
26. Zhang, L., Li, K., Li, K., Xu, Y.: Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems. International Journal of Electrical Power & Energy Systems. **78**, 499–512 (2016)
27. Saeedizade, E., Ashtiani, M.: Ddbws: a dynamic deadline and budget aware workflow scheduling algorithm in workflow-as-a-service environments. The Journal of Supercomputing **77**(12), 14525–14564 (2021)
28. Zhang, L., Li, K., Zheng, W., Li, K.: Contention-aware reliability efficient scheduling on heterogeneous computing systems. IEEE Transactions on Sustainable Computing **3**(3), 182–194 (2017)
29. Rodriguez, M.A., Buyya, R.: Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Futur. Gener. Comput. Syst. **79**, 739–750 (2018)
30. Xiao, X., Xie, G., Xu, C., Fan, C., Li, R., Li, K.: Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems. Journal of Computational Science **26**, 344–353 (2018)
31. Zhang, L., Ai, M., Liu, K., Chen, J., Li, K.: Reliability enhancement strategies for workflow scheduling under energy consumption constraints in clouds. IEEE Transactions on Sustainable Computing, 1–14 (2023). https://doi.org/10.1109/TSUSC.2023.3314759
32. Senapati, D., Sarkar, A., Karfa, C.: Hmds: a makespan minimizing dag scheduler for heterogeneous distributed systems. ACM Transactions on Embedded Computing Systems (TECS) **20**(5s), 1–26 (2021)
33. Faragardi, H.R., Sedghpour, M.R.S., Fazliahmadi, S., Fahringer, T., Rasouli, N.: Grp-heft: a budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds. IEEE Trans. Parallel Distrib. Syst. **31**(6), 1239–1254 (2019)
34. Han, P., Du, C., Chen, J., Du, X.: Minimizing monetary costs for deadline constrained workflows in cloud environments. IEEE Access **8**, 25060–25074 (2020)
35. Zhang, L., Wang, L., Xiao, M., Wen, Z., Peng, C.: Em woa: A budget constrained energy consumption optimization approach for workflow scheduling in clouds. Peer-to-Peer Networking and Applications, 1–15 (2022)
36. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K.: Characterization of scientific workflows. In: 2008 Third Workshop on Workflows in Support of Large-Scale Science, pp. 1–10 (2008). IEEE
37. Singh, R.M., Awasthi, L.K., Sikka, G.: Towards metaheuristic scheduling techniques in cloud and fog: an extensive taxonomic review. ACM Computing Surveys (CSUR) **55**(3), 1–43 (2022)
38. Zhang, L., Li, K., Li, C., Li, K.: Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Inf. Sci. **379**, 241–256 (2017)
39. Gupta, A., Faraboschi, P., Gioachin, F., Kale, L.V., Kaufmann, R., Lee, B.-S., March, V., Milojicic, D., Suen, C.H.: Evaluating and improving the performance and scheduling of hpc applications in cloud. IEEE Transactions on Cloud Computing **4**(3), 307–321 (2016)