frontiers | Frontiers in Environmental Science

TYPE Original Research
PUBLISHED 04 January 2023
DOI 10.3389/fenvs.2022.996483

OPEN ACCESS

EDITED BY
Mengchu Zhou,
New Jersey Institute of Technology,
United States

REVIEWED BY
Neeraj Arora,
Vardhaman Mahaveer Open University,
India
Lun Hu,
Xinjiang Technical Institute of Physics
and Chemistry (CAS), China
Quanwang Wu,
Chongqing University, China

*CORRESPONDENCE
Hao Sheng,
shenghao@buaa.edu.cn

SPECIALTY SECTION
This article was submitted to
Environmental Informatics and Remote
Sensing,
a section of the journal
Frontiers in Environmental Science

RECEIVED 17 July 2022
ACCEPTED 08 December 2022
PUBLISHED 04 January 2023

CITATION
Pu J, Meng Q, Chen Y and Sheng H
(2023), MPEFT: A novel task scheduling
method for workflows.
Front. Environ. Sci. 10:996483.
doi: 10.3389/fenvs.2022.996483

MPEFT: A novel task scheduling method for workflows

Juhua Pu[1,2], Qiaolan Meng[1,2], Yexuan Chen[1] and Hao Sheng[1,2]*

[1]State Key Laboratory of Software Development Environment, Beihang University, Beijing, China,
[2]Beihang Hangzhou Innovation Institute Yuhang, Hangzhou, China

Optimizing the scheduling algorithm is a key problem to improving the service efficiency of urban heterogeneous computing platforms. In this paper, we propose a novel list-based scheduling algorithm called Modified Predict Earliest Finish Time (MPEFT) for heterogeneous computing systems with the aim to minimize the total execution time. The algorithm consists of two stages: task prioritization and processor selection. In the task prioritization phase, the priority of tasks is calculated by time cost of all paths from a task to the exit task. Compared with the prior works, more accurate task priorities are obtained by considering not only the critical path but also the non-critical ones. In the processor selection phase, the processor is allocated for a task according to whether the computing resources are sufficient to its successive tasks. The experiments on randomly generated workflows and the workflows from practical applications show that the MPEFT outperforms other existing list scheduling algorithms.

KEYWORDS

workflow scheduling, DAG scheduling, list-based scheduling, heterogeneous platform, random graphs generator

1 Introduction

In recent years, with the acceleration of smart city construction, the smart city's computing environment has become more mature, and various urban digital services have emerged quietly. At the same time, the maturity of artificial intelligence algorithms also provides a powerful tool for data analysis. The rapid development of big data, cloud computing and artificial intelligence has promoted the development of smart cities. However, this explosive growth of smart city services presents a higher requirement for the load capacity and performance of urban computing centers. In addition to expanding computing resources, how to improve the efficiency and performance of task scheduling is also a matter of concern (Duan et al., 2020; Li, 2020).

Complex computing in smart cities often consists of multiple tasks. There may be dependencies between these tasks, and data transmission is usually required. These tasks with dependencies form a workflow, a directed acyclic graph (DAG) in which nodes represent the tasks and edges denotes the dependencies. Computing resources are often composed of machines with a certain degree of heterogeneity, e.g., CPUs and GPUs. An example workflow of resources vitalization for urban epidemic analysis and prediction is shown in Figure 1. It contains nine tasks for studying the epidemic situation from different perspectives. The ultimate objective task relies on three modeling sub-tasks, each

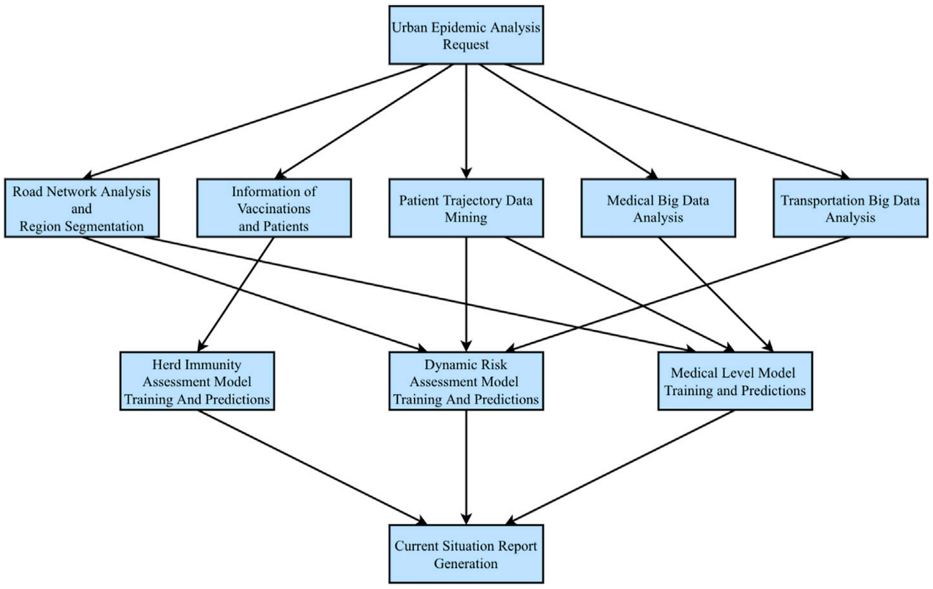

Urban Epidemic Analysis Request
Road Network Analysis and Region Segmentation
Information of Vaccinations and Patients
Patient Trajectory Data Mining
Medical Big Data Analysis
Transportation Big Data Analysis
Herd Immunity Assessment Model Training And Predictions
Dynamic Risk Assessment Model Training And Predictions
Medical Level Model Training and Predictions
Current Situation Report Generation

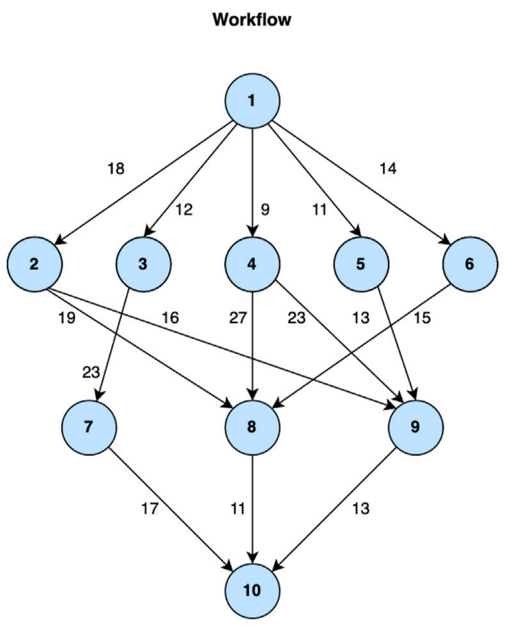
FIGURE 1
An workflow example of digital services.

of which is performed based on five basic services varying from partitioning the maps to assessing the track of infectors. For different types of tasks, specific computation resources are preferred, e.g., high-performance units such as GPUs and FPGAs are more suitable for the modeling and prediction tasks. Scheduling such workflows onto a heterogeneous computing platform so that their total execution time is minimum, is crucial for the efficiency and efficacy of the smart city management (Zheng et al., 2021).

The procedure can be formalized as a celebrated static scheduling problem, in which all information about the tasks and computation resources are known in advance, and its goal is to minimize the total execution time, i.e., the *makespan*. Due to its key importance in heterogeneous computing systems, the task scheduling problem has been widely studied and many heuristic algorithms were proposed in the literatures. The heuristic algorithms are mainly divided into three groups: list-based scheduling (Topcuoglu et al., 2002; Arabnejad and Barbosa, 2013; Djigal et al., 2019; Madhura et al., 2021),clustering-based scheduling (Wang and Sinnen, 2018), and duplication-based scheduling (Sulaiman et al., 2021). List-based scheduling algorithms have the advantages of low complexity and high efficiency and are widely studied and applied in many scenarios. The list-based scheduling algorithm completes the scheduling through two stages: calculating task priority and assigning processors in sequence Topcuoglu et al. (2002). The existing list-based scheduling algorithms determine the priority of tasks through the *critical path*, the longest path in the workflow from a task to the last task in DAG Zhou et al. (2017). However, this ignores the impact of other non-critical paths in the DAG, making tasks on other non-critical paths be inappropriately scheduled. For example, in Figure 1, the task Patient Trajectory Data Mining and the task Medical Big Data Analysis may have the same critical path, but the former should have a higher priority because there are more tasks to be calculated on the path from him to the exit task.

In this paper, we propose a new list-based scheduling algorithm, dubbed Modified Predict Earliest Finish Time (MPEFT), for scheduling workflows to fully heterogeneous computing resources with an aim to minimize the makespan. Compared with the existing list scheduling algorithms (Topcuoglu et al., 2002; Arabnejad and Barbosa, 2013; Djigal et al., 2019, 2020; Madhura et al., 2021) that mainly depend on the critical path, the MPEFT adopts more appropriate features to schedule the tasks. The novelty of the MPEFT is the calculation of task priority and the evaluation of critical paths. MPEFT treats critical paths and non-critical paths equally when calculating task priority, and determines priority based on task overhead on all paths. When allocating computing resources for tasks, MPEFT measures the impact of critical paths on the makespan and adjusts the allocation strategy. The impact of critical paths is evaluated by computing the proportion of their time cost in the whole execution cost.

Extensive experiments show that the performance of MPEFT is significantly better than other popular list-based scheduling algorithms in most scenarios.



Computation Cost

Task	p_1	p_2	p_3
t_1	14	16	9
t_2	13	19	18
t_3	11	13	19
t_4	13	8	17
t_5	12	13	10
t_6	13	16	9
t_7	7	15	11
t_8	5	11	14
t_9	18	12	20
t_{10}	21	7	16

FIGURE 2
A workflow with ten tasks and computation cost matrix on three processors.

2 Background

The scheduling problem studied in this paper is to assign tasks in a workflow onto computing resources. The DAG structure and computation cost matrix in Figure 2 illustrate the details of the example in Figure 1. There are ten tasks and three heterogeneous processors. The tasks will be scheduled onto three processors, such that the runtime of the whole workflow, i.e., the *makespan*, is minimum.

The computing resource in this paper is the processor set, which we denote by $P = \{p_1, \ldots, p_m\}$. A processor can execute any task, though the number of its executing tasks at any time can be only one. A workflow contains multiple tasks with some dependencies between them.

A workflow can be represented by a directed graph $G = (V, E)$, where $V = \{t_1, \ldots, t_n\}$ represents the set of tasks contained in the workflow and E represents the set of dependencies between tasks. Each edge $e_{i,j}$ in E indicates a dependency between a tuple (t_i, t_j) of tasks, which means that the execution of the task t_j depends on the output or the end state of the task t_i. For each edge $e_{i,j}$, the task t_j cannot start execution unless the task t_i has been performed. We call the task t_i the immediate predecessor of the task t_j, and the task t_j the immediate successor of the task t_i. The edge $e_{i,j}$ also has a specific weight $c_{i,j}$, indicating the communication time that it takes to transfer the result data from the task t_i to the task t_j. The $c_{i,j}$ are usually defined as:

$$c_{i,j} = \bar{L} + \frac{data_{i,j}}{\bar{B}} \tag{1}$$

where $\bar{L}$ represents the average startup time of the processor when the data transfer occurs, $\bar{B}$ represents the average transfer bandwidth over all processors in the computing platform, and $data_{i,j}$ is the volume of the data required to transfer between the two tasks. We note that when the two tasks are assigned to the same processor, the intermediate data does not need to be transferred and the network communication cost $c_{i,j}$ is considered to be 0.

In this paper, we denote the computation cost of all tasks on different processors by the matrix w, in which $w\ (t_i, p_j)$ is the processing time of the task t_i on the processor p_j. Due to the heterogeneity, a task may cost unequal computation times on different processors. For example, in Figure 2, the task t_1 takes 14, 16, and nine time units to be executed on p_1, p_2, and p_3, respectively. In addition, we denote by $w^*(t_i)$ the minimum computation cost of the task t_i over all processors. For example, $w^*(t_i)$ in Figure 2 is 9. To explain the algorithm formally, we first introduce some standard notations.

Definition 1 (**Entry and exit tasks**). *In a workflow, the entry task t_{entry} is defined as the task without any predecessor, and similarly, the exit task t_{exit} is defined as the task without any successor.*

We assume that there is only one entry task in the DAG graph. If a workflow contains multiple entry tasks, a predecessor with zero computational overhead is added ahead of these tasks.

TABLE 1 Summary of parameters.

Parameter	Definition
G	the DAG of Workflow
V	the set of all tasks of a workflow
E	the set of all edges of a workflow
P	the set of all processors
p_j	the jth processor in P
t_i	the ith task in a workflow
$e_{i,j}$	the edge between t_i and t_j
$pred\ (t_i)$	the set of immediate predecessor tasks of t_i
$succ\ (t_i)$	the set of immediate successor tasks of t_i
$w\ (t_i, p_j)$	the computation cost of t_i on p_j
$w^*(t_i)$	the min computation cost of t_i
$data_{i,j}$	the data transfer amount between t_i and t_j
$c_{i,j}$	the communication cost of ti and t_j
$avail\ (t_i, p_j)$	the earliest time of executing t_i on p_j
$AFT\ (t_i)$	the actual finished time of t_i
$EST\ (t_i, p_j)$	the earliest starting time of t_i on p_j
$EFT\ (t_i, p_j)$	the earliest finished time of t_i on p_j

The same assumption is made for the exit tasks. For example, in Figure 2, the entry task is t_1 and the exit task is t_{10}.

Definition 2 (**Successors and predecessors**). *Given a task t_i, we denote by $succ(t_i)$ the set of all immediate successors of t_i, and $pred(t_i)$ the set of all immediate predecessors of t_i. For example, in Figure 2, $succ(t_4)$ is $\{t_8, t_9\}$ and $pred(t_4)$ is $\{t_1\}$.*

The goal of workflow scheduling is to compute a scheme that schedules each task to an appropriate processor such that the makespan of the entire workflow is minimum.

Definition 3 (**Makespan**). *The makespan of a workflow is defined as the time when the last task in the workflow has been executed, that is:*

$$makespan = AFT\,(t_{exit}), \tag{2}$$

where $AFT\ (t_i)$ is the actual finished time of the task t_i.

The makespan is unknown unless all tasks have been scheduled on the processors. Thus, during the scheduling, some terms are introduced to estimate the actual finished time of each task. The most general one used in many literature (Topcuoglu et al., 2002) is the *Earliest Finished Time (EFT)*. The EFT is defined on the *Earliest Start Time (EST)*, which is also an estimator but for the actual start time of the tasks. Finally, more notations used in this paper are shown in Table 1.

Definition 4 (**Earliest Start Time (EST)**). *The Earliest Start Time $EST(t_i, p_j)$ represents the earliest time when the task t_i can be executed if t_i is assigned to processor p_j:*

$$EST\left(t_i, p_j\right) = \max\left\{avail\left(t_i, p_j\right), \max_{t_k \in pred\,(t_i)} \{AFT\,(t_k) + c_{k,i}\}\right\}. \tag{3}$$

where avail (t_i, p_j) represents the earliest time that p_j can start the execution of t_i and $\max_{t_k \in pred\,(t_i)}\{AFT\,(t_k) + c_{k,i}\}$ is the time when all tasks in pred (t_i) are completed and the intermediate data is transferred to p_j.

Definition 5 (**Earliest Finished Time (EFT)**). *The Earliest Finished Time $EFT(t_i, p_j)$ represents the earliest finished time of t_i on p_j:*

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

3 Related work

This paper focuses on the static single-objective workflow scheduling problem with DAG properties under heterogeneous computing resource platforms. The common objective of the problem is the makespan of the workflow. Theoretically, this problem is NP-complete and has not any polynomial algorithm unless NP = $\mathcal{P}$(Houssein et al., 2021). Thus, most of the existing methods are heuristic and high-efficient.

We can further classify the heuristic algorithms into the cluster-based (Boeres et al., 2004), the replication-based (Sulaiman et al., 2021) and the list-based (Topcuoglu et al., 2002; Arabnejad and Barbosa, 2013; Madhura et al., 2021; NoorianTalouki et al., 2022), among which the list-based scheduling algorithms have been widely studied due to their lower complexity and higher performance. This kind of algorithm generally consists of two phases: the task priority sorting and the processor selection. At the task prioritizing phase, the scheduling algorithm calculates a priority value for each task and obtains a list based on the priority. It usually takes into account several characteristics of the workflow, such as the execution time of tasks on different processors, dependencies between the tasks, and communication costs, to compute the task priority. At the processor selection phase, the scheduling algorithm progressively selects the most suitable processor for each task by the natural order of the task list. In many algorithms, the two phases are often tangled, i.e., the processor selection would depend on the priority of tasks.

We briefly review some list-based algorithms in Table 2. The critical path plays a certainly important role in the existing list-based algorithms Topcuoglu et al. (2002); Arabnejad and Barbosa (2013); Zhou et al. (2017); Djigal et al. (2019). Many metrics, e.g., $rank_u$ in HEFT, $rank_d$ in CPOP, and the OCT in PEFT, are calculated along some critical path in the workflow. However, we argue that the impact of the critical path on the final makespan might be overestimated in these algorithms, when there are some other paths that have a similar length as the critical one, i.e., such paths are also vital to the scheduling plan. Especially when the

TABLE 2 Some promising methods for workflow scheduling.

Method	Advance	Type of algorithm	Type of tasks	Objectives	Complexity
HEFT Topcuoglu et al. (2002)	Sort the task list based on $rank_u$ and $rank_d$	list-based	single workflow	single-objective (makespan)	$O\ (v^2 p)$
CPOP Topcuoglu et al. (2002)	Reduce makespan by optimizing critical path	list-based	single workflow	single-objective (makespan)	$O\ (v^2 p)$
PEFT Arabnejad and Barbosa (2013)	Predict makespan of critical path for each task by *OCT*	list-based	single workflow	single-objective (makespan)	$O\ (v^2 p)$
E-HEFT Hu et al. (2019)	Consider makespan and energy consumption in real-time scenarios	list-based	single workflow	multi-objective (makespan and energy consumption)	$O\ (v^2 p)$
ELSH Wu et al. (2021)	Consider the impact of node communication contention	list-based	single workflow	single-objective (makespan)	$O\ (v^4)$
ROSA Chen et al. (2018)	Combine computing resource adjustment and scheduling to optimize QoS	list-based	multiple workflows	multi-objective (makespan and monetary cost)	$O\ (v^3)$
CHP Boeres et al. (2004)	Apply clustering heuristic for homogeneous processors to heterogeneous environment	cluster-based	single workflow	single-objective (makespan)	$O\ (v^2 p)$
HLTSD Sulaiman et al. (2021)	Combine duplication-based and list-based method	duplication-based	single workflow	single-objective (makespan)	$O\ (v^2 p)$
SDHN Li et al. (2021a)	Apply NSGA-II to minimize both makespan and cost	genetic algorithm	multiple workflows	multi-objective (makespan and monetary cost)	$O\ (\tau n\ (v^2+n))$
QL-HEFT Tong et al. (2020b)	Calculate task priority using Q-learning method	RL	single workflow	single-objective (makespan)	-
MCDS Tuli et al. (2021)	Optimize scheduling strategies using Monte Carlo method	RL	multiple workflows	multi-objective (makespan, energy consumption *etc.*)	-

computation resources are limited, exaggerating the effect of the critical path on the final makespan would let the algorithm pay more attention to the critical tasks and the critical processors, whereas overlooking the substantial cost of the non-critical tasks.

For instance, in Example 2, some paths from the task t_1 through its different successors have similar length. Nevertheless, when allocating the processors for t_1, only the critical path, the path *via* t_2, is considered, which will mislead the algorithm to selecting the processor p_2 for t_1, since its successive critical task t_2 has the minimum runtime on p_2. A better scheduling for the task t_1 is however p_3, as t_1 has the minimum EFT on p_3, and the other non-critical but also important tasks can benefit from the fast computation of t_1. Please refer to Sec 4.4 for more discussions.

Some researches on the scheduling problem utilize more sophisticated meta-heuristic algorithms to iteratively optimize the scheduling solution (Li et al., 2021b; Wang and Zuo, 2021). Though these algorithms might obtain better performance than the heuristic ones, they usually suffer from poor efficiency, as the optimization procedure could be very lengthy. To accelerate the convergence of the optimal solution, many meta-heuristic algorithms often take the optimal solution from some heuristic algorithm as a good initialization (Wu et al., 2019; Pham and Fahringer, 2020; Li et al., 2021a). The proposed MPEFT could also serve as an initializer plugged into these meta-heuristic algorithms. Very Recently, the (deep) reinforcement learning is brought in for the scheduling problem (Tong et al., 2020b,a; Tuli et al., 2021). These approaches aim to learn a heuristic scheduling strategy instead of devising it by hand. The learning procedure involves in a large amount of evaluation and update of the current strategy, and thus are inevitably hindered by the same efficient problem as the meta-heuristic methods.

We briefly summarize some work related to task scheduling, as shown in Table 2. Methods based on list scheduling have been extensively studied. And some researchers try to use meta-heuristic methods and reinforcement learning methods to solve scheduling problems in more complex scenarios, at the cost of higher complexity. For example, in addition to the number of tasks, the complexity of SDHN Li et al. (2021a) based on genetic algorithms also needs to consider a large number of populations n and iterations τ.

4 The proposed algorithm MPEFT

Motivated by the deficiency of existing approaches analyzed in the section above, we propose a novel list-based scheduling algorithm, dubbed the Modified Predict Earliest Finish Time (MPEFT) algorithm, for heterogeneous computing systems. Before introducing the MPEFT algorithm, we define some attributes *OffspringSet* and *DCT* used for determining the task priorities, and for selecting processors for tasks.

Definition 6 (**Offspring set**). *The set OffspringSet*(t_i) *contains all the direct and indirect successor tasks of* t_i*:*

$$OffspringSet(t_i) = \begin{cases} \bigcup_{t_j \in succ(t_i)} OffspringSet(t_j), & if\ t_i is\ not\ an\ exit\ task \\ \varnothing, & otherwise \end{cases} \tag{5}$$

Definition 7 (**Direct Calculation Time (DCT)**). *The Direct Calculation Time DCT*(t_i) *represents the minimum time required to perform the task* t_i *and communicate from* t_i *to all its successors.*

$$DCT(t_i) = w^*(t_i) + \sum_{t_j \in succ(t_i)} c_{i,j}. \tag{6}$$

4.1 Task prioritizing phase

In MPEFT, we use the summed *DCT* overall offspring to prioritize the tasks:

$$rank_{AP}(t_i) = DCT(t_i) + \sum_{t_j \in OffspringSet(t_i)} DCT(t_j). \tag{7}$$

Contrary to the task ranks used in the prior works that only deal with the critical path, $rank_{AP}$ computes the optimal cost over all paths (AP) from a task to the exit task. More specifically, the rank $rank_{AP}$ (t_i) contains the computation and communication cost of all subsequent tasks starting from the task t_i to the exit task. The $rank_{AP}$ could provide a more accurate priority of tasks since it takes into account the cost of non-critical tasks, which could also contribute a lot to the value of makespan. The intuition is that in the case where the number of successors of a task is larger than the number of processors, and every successor consumes considerable runtime, the successive tasks cannot be fully paralleled onto the processors and the earlier execution of the preceded task could speed up the whole workflow. Even though sometimes the computation resources are adequate, the processors would not be always available during the running of the workflow, and thus the scheduling performance might still benefit from the proposed $rank_{AP}$.

```
Input: A workflow $G = (V, E)$ and the computation cost matrix $w$
Output: $rank_{AP}$ of all tasks
1:  Compute the DCT for each task in $G$ by (6)
2:  $rank_{AP}$ ←{}; $OffspringSet$ ←{} ▷ Initialize as empty dictionaries
3:  Get the entry task $t_{entry}$ from $G$
4:  CALCURANKFORTASK ($t_{entry}$) ▷ Recursively compute $rank_{AP}$
5:  return $rank_{AP}$
```

TABLE 3 Produced by Processor Selection Phase on the sample DAG in Figure 2

Task	$rank_{AP}$	OCT			CPS			k		
		p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3
t_1	332	48	38	53	t_2	t_2	t_2	0.3	0.3	0.3
t_2	96	35	19	35	t_9	t_9	t_9	1	1	1
t_3	65	28	22	27	t_7	t_7	t_7	1	1	1
t_4	106	38	19	36	t_9	t_9	t_9	1	1	1
t_5	55	32	19	32	t_9	t_9	t_9	1	1	1
t_6	47	23	18	30	t_8	t_8	t_8	1	1	1
t_7	31	21	7	16	t_{10}	t_{10}	t_{10}	1	1	1
t_8	23	18	7	16	t_{10}	t_{10}	t_{10}	1	1	1
t_9	32	20	7	16	t_{10}	t_{10}	t_{10}	1	1	1
t_{10}	7	0	0	0	-	-	-	-	-	-

```
 6:
 7:  function CALCURANKFORTASK ($t_i$)
 8:      if task $t_i$ is exit task then
 9:          $OffspringSet\ (t_i) \leftarrow \varnothing$
10:      else
11:          for $t_j \in succ\ (t_i)$ do
12:              if $OffspringSet\ (t_j) = \varnothing$ then
13:                  CALCURANKFORTASK ($t_j$)
14:              end if
15:
         $OffspringSet\,(t_i) \leftarrow \bigcup_{t_j \in succ\,(t_i)} OffspringSet\,(t_j)$
16:          end for
17:      end if
18:
     $rank_{AP}\,(t_i) = DCT\,(t_i) + \sum_{t_j \in OffspringSet\,(t_i)} DCT\,(t_j)$
19:  end function
```

Algorithm 1. Task Priority Calculation

When calculating the $rank_{AP}$, we use the minimum computation cost w^* rather than $\bar{w}$ in the prior algorithms. It is by design and aims to align the task prioritizing with the processor selection since in the processor selection, the *EFT* comprising *AFT* is utilized, and the w^* is a better estimator for *AFT* than $\bar{w}$. We elaborate the calculation of $rank_{AP}$ in Algorithm 1.

4.2 Processor selection phase

The processor selection strategy of MPEFT is built on that of PEFT. During this phase, the Optimistic Cost Table (OCT) (8) is first calculated. It is defined recursively and only the entry task needs to be considered. For the exit task, the $OCT\,(t_{exit}, p_k) = 0$ for all processors $p_k \in P$.

$$OCT\,(t_i, p_k) = \max_{t_j \in succ\,(t_i)} \left[\min_{p_w \in P} \left\{ OCT\left(t_j, p_w\right) + w\left(t_j, p_w\right) + c_{i,j}^{k,w} \right\} \right] \tag{8}$$

where $c_{i,j}^{k,w} = c_{i,j}$ if $p_w \neq p_k$, else $c_{i,j}^{k,w} = 0$. When calculating the *OCT*, we additionally record the immediate successor (critical) tasks on the critical path, namely the Critical Path Successor (CPS) table:

$$CPS\,(t_i, p_k) = \arg\max_{t_i \in succ\,(t_i)} \left[\min_{p_w \in P} \left\{ OCT\left(t_j, p_w\right) + w\left(t_j, p_w\right) + c_{i,j}^{k,w} \right\} \right]. \tag{9}$$

The assignment of tasks is determined by the Modified Earliest Start Time (MEFT):

$$MEFT\,(t_i, p_k) = EFT\,(t_i, p_k) + OCT\,(t_i, p_k) \times k\,(t_i, p_k). \tag{10}$$

where $k\,(t_i, p_k)$ is a rational weight controlling the impact of the critical path on the processor selection. Every $k\,(t_i, p_k)$ is computed by

$$k\,(t_i, p_k) = \begin{cases} rank_{AP}\,(CPS\,(t_i, p_k)) / \sum_{t_j \in succ\,(t_i)} \left(rank_{AP}\left(t_j\right) + c_{i,j} \right), \text{if } |succ\,(t_i)| > |P| + 1 \\ 1, \text{otherwise} \end{cases} \tag{11}$$

Note that $CPS(t_i, p_k) \in succ\,(t_i)$, and thus the weight $k\,(t_i, p_k) \leq 1$. In other words, the MEFT only diminishes the impact of the critical path, and will degenerate to PEFT if the cost of the critical path dominates the whole time of a task and its offspring. When the number of processors is one larger than the successors of a

TABLE 4 Schedule Produced by the MPEFT Algorithm in Each Iteration on the sample DAG in Figure 2.

Step	Ready list	Task selected	EFT			MEFT			Processor selected
			p_1	p_2	p_3	p_1	p_2	p_3	
1	t_1	t_1	14	36	**9**	28.3	27.3	**24.8**	p_3
2	t_2, t_3, t_4, t_5, t_6	t_4	31	**26**	**26**	69	**45**	62	p_2
3	t2, t_3, t_5, t_6	t_2	40	46	**27**	75	65	**62**	p_3
4	t_3, t_5, t_6	t_3	**32**	39	46	**60**	61	73	p_1
5	t_5, t_6, t_7	t_5	44	39	**37**	76	**58**	69	p_2
6	t_6, t_9, t_7	t_6	45	55	**36**	68	73	**66**	p_3
7	t_9, t_7, t_8	t_9	70	**55**	72	90	**62**	88	p_2
8	t_7, t_8	t_7	**39**	70	66	**60**	77	82	p_1
9	t_8	t_8	**58**	66	67	76	**73**	83	p_2
10	t_{10}	t_{10}	98	**73**	93	98	**73**	93	p_2

That the bold values indicate some optimal values in the second stage of the algorithm.

task, the successors can be paralleled onto multiple processors, and thus MEFT also adopts the original PEFT.

Recall that $rank_{AP}$ corresponds to the total time over all paths from a task to the exit task. The k (t_i, p_k) evaluates the impact of the critical path by calculating the proportion of the critical path in the total cost over all paths. The smaller fraction of time the critical path occupies, the lower impact it will have.

Compared with the existing scheduling algorithms, MEFT based allocation strategy selects the processor to optimize the cost of the critical path, only when the critical path is vitally important to the final makespan. If there are several comparably costly paths from a task to the exit task, different selections of the processor might not change the whole time of the offspring tasks too much. Subsequently, when selecting the processor, the assignment of a task should pay more attention to the total time of its ahead tasks, i.e., the EFT of the task. For instance, in Figure 2, assigning the task t_2 to the processor p_3 that has the smallest EFT will produce a better scheduling plan.

4.3 Detailed description of the MPEFT algorithm

The pseudocode of the MPEFT algorithm is shown in Algorithm 2. MPEFT will first calculate the *OCT*, *CPS* and $rank_{AP}$ values of all tasks, and then create a ready list with a single entry task. The $rank_{AP}$ value is processed in the order as described in Algorithm 1. Line four will take out the task with the highest $rank_{AP}$ value in the current ready list for processor allocating. Lines 5–12 will select the appropriate processor for the task. The line six calculates the *EFT*, and lines 7–10 calculate *k*, the importance of the critical successor of the current task. Line 12 calculates *MEFT* for the processor selection. Line 15 will put the new ready tasks into the ready-list. Repeat the procedure until all tasks have been executed, and the scheduling plan and the overall computation cost are obtained.

```
Input: A workflow $G = (V, E)$ and the computation cost matrix $w$
Output: A scheduling plan with its makespan
1:  Compute $rank_{AP}$ and $OCT$ $CPS$ table for all tasks
2:  Create Empty ready list $L$ and put entry task as initial task
3:  while $L$ is NOT Empty do
4:      $t_i \leftarrow$ the task with highest $rank_{ap}$ from $L$
5:      for $p_j \in P$ do
6:          Compute $EFT$ ($t_i$, $p_j$) value using insertion-based scheduling policy
7:          if $|succ\ (t_i)| < |P| + 1$ then
8:              $k \leftarrow 1$
9:          else
10:             Compute $k$ by (11)
11:         end if
12:         $MEFT\ (t_i, p_j) = EFT\ (t_i, p_j) + OCT\ (t_i, p_j) \times k$
13:     end for
14:     Assign task $t_i$ to the processor $p_j$ that minimize MEFT of task $t_i$
15:     Update $L$
16: end while
```

Algorithm 2. The MPEFT Algorithm

In terms of algorithm complexity, we denote by v, e, p the number of tasks, edges, and processors, respectively. In the task priority calculation stage, the time complexity of computing $rank_{AP}$ is $O\ (v^2{+}e)$, and the computational complexity of *OCT* and *CPS* is $O\ (p\ (v + e))$; in the processor allocation stage, the complexity is $O\ (v^2 p)$.

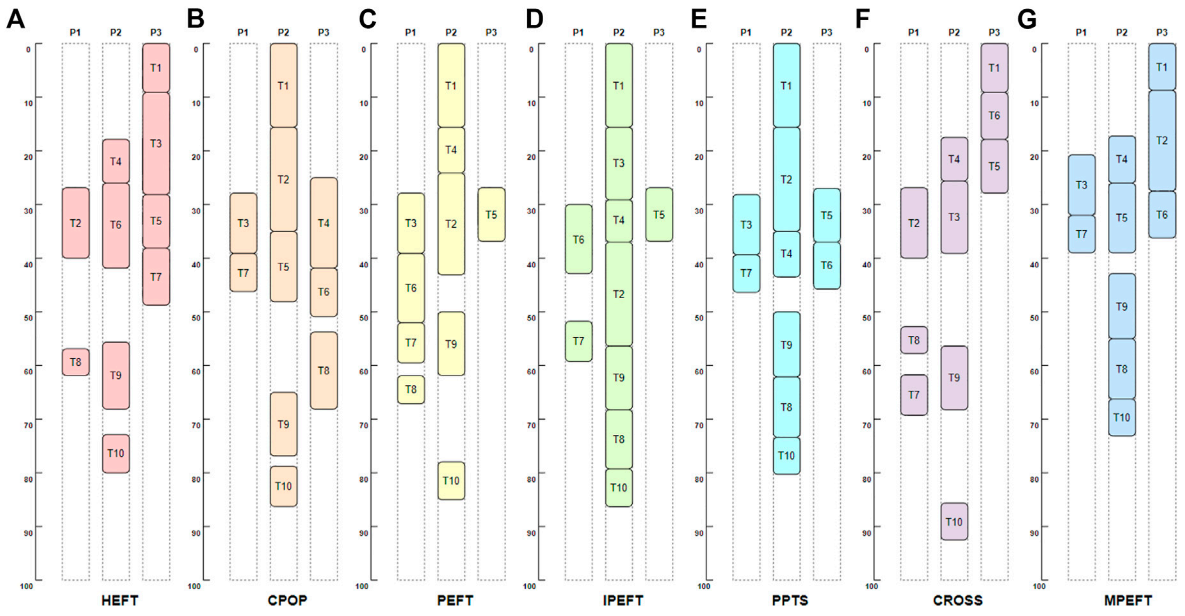


FIGURE 3
Schedules of the sample DAG in Figure 2 with **(A)** HEFT (makespan = 80), **(B)** CPOP(makespan = 86), **(C)** PEFT (makespan = 85), **(D)** IPEFT (makespan = 95), **(E)** PPTS (makespan = 80), **(F)** CROSS(makespan = 93) and **(G)** MPEFT (makespan = 73).

So the overall complexity is O (v^2+e + v^2p). In the extremely dense DAG graph, e is equal to v^2, and the overall time complexity is O (v^2p), which is consistent with the HEFT algorithm.

4.4 Case study

We analyze the workflow scheduling example shown in Example 2 as a case study. In task prioritizing phase, the $rank_{AP}$ for each task by Eq. 7 is shown in Table 3. The task priority list is (t_1, t_4, t_2, t_3, t_5, t_6, t_9, t_7, t_8, t_{10}).

Table 3 also shows the calculation results of OCT, CPS and k respectively. The value of k of the task t_1 is .3. It indicates that the number of direct successor tasks of t_1 is more than the number of current processors, and its critical path occupies only 30% of the total time cost. When selecting a processor for task t_1, the impact of the critical path is determined according to its CPS and k.

Table 4 shows the scheduling results of each round in MPEFT. The comparison of the scheduling scheme of MPEFT with other list-based scheduling algorithms is shown in Figure 3. The MPEFT algorithm schedules the task t_1 to the processor p_3, while other algorithms except HEFT consider the processor p_2 to be a better choice for t_1. However, these algorithms focusing on optimizing task scheduling on the longest path starting from t_1 to the exit task would not guarantee global optimization. This is because the assignment of t_1 on p_2 will cause a delay of t_1 compared to p_3, and its non-critical successor tasks are also affected by the delay of task t_1, which leads to an increased maskpan. In the MPEFT algorithm, the influence of the critical successor of t_1 is evaluated by the controlling weight k. The value .3 of k decreases the impact of the critical-path-based metric OCT on the processor selection, which in turn argues the importance of the value of EFT. It makes the algorithm select p_3 with the minimum EFT for t_1, ensuring the earliest finish of t_1 as well as all its successors. Though the critical path from t_1 might take a longer time to complete, the total time cost of the workflow, the makespan, in MPEFT is smaller than the prior algorithms.

5 Experiment

In this paper, we compare our MPEFT with other list-based scheduling algorithms using the metrics the Scheduling Length Ratio (SLR), the Number of Occurrences of Better Quality of Solutions (NOBQS), makespan standard deviation and running time of the algorithms. Since DAGs may have very different topologies, the SLR is often used to represent the scheduling efficiency of the algorithm, which is defined as follows:

$$SLR = \frac{makespan}{\sum_{t_i \in CP} min_{p_j \in P}\left(w\left(t_i, p_j\right)\right)} \qquad (12)$$

where CP is the critical path from the entry to exit task.

TABLE 5 Parameters used in simulation studies.

Parameter	Description	Values
n	the number of nodes in the DAG	[50, 100, 200, 300, 400, 500, 600, 700, 800]
fat	the width of the DAG	[.1, .4, .8]
density	density of the DAG	[.2, .8]
regularity	the regularity of the distribution of nodes at every level of the DAG	[.2, .8]
jump	the maximum number of levels spanned by edges in the DAG	[2, 5, 8]
ccr	the ratio of communication cost to computation cost in DAG	[.1, .5, 1, 5, 10, 20]
heterogeneity	the heterogeneity factor for processor speeds	[.1, .25, .5, 1, 2]
p	the number of processors	[4, 8, 16, 32]

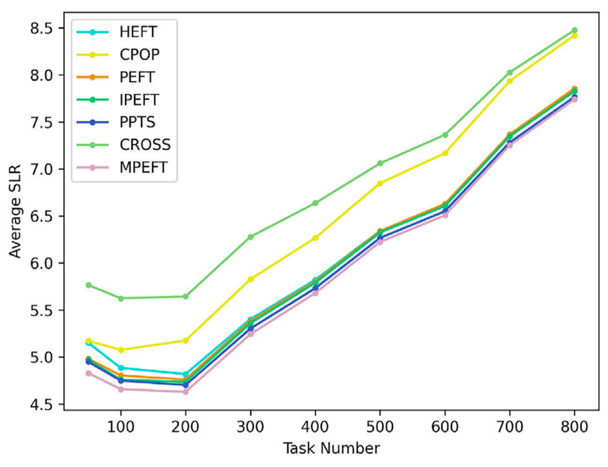


FIGURE 4
Average SLR for random graphs with respect to DAG size.

The NOBQS shows the comparison results of the scheduling schemes of different algorithms on all DAG graphs in the form of a table, including better, worse and the same. This paper will compare the MPEFT algorithm with several baselines, including HEFT (Topcuoglu et al., 2002), CPOP (Topcuoglu et al., 2002), PEFT (Arabnejad and Barbosa, 2013), IPEFT (Zhou et al., 2017), PPTS (Djigal et al., 2019) and CROSS (Madhura et al., 2021).

Makespan standard deviation describes the robustness of the algorithms. The makespan standard deviation $\sigma_{makespan}$ is defined as 13. The smaller the makespan standard deviation, the stronger the robustness of the algorithm.

$$\sigma_{makespan} = \sqrt{\frac{1}{N_r} \times \sum_{i=1}^{N_r} \left(makespan_i - Avg_{makespan}\right)^2} \tag{13}$$

All experiments were performed on a computer with a 40-core Intel(R) Xeon(R) Gold 5218R 2.10 GHz processor and 32 GB of RAM. We use a simulator, which is realized by *Python*, to conduct the comparative experiments. The code of the proposed MPEFT is released on the Website https://github.com/MengQiaolan/DAG_Scheduling.

5.1 Randomly generated application graphs

5.1.1 Random graph generator

To comprehensively evaluate the effectiveness of the algorithm on different graphs, we use a random graph generator to obtain graphs with different topologies. The random graph generator generates multiple DAG graphs with the given input parameters (frs69wq, 2012). Related parameters are described as follows:

n: the number of computation nodes in the DAG (i.e., tasks in workflow).
fat: the width of the DAG, the maximum number of tasks that can be executed concurrently. A small value will lead to a thin DAG (e.g., chain) with a low task parallelism, while a large value induces a fat DAG (e.g., fork-join) with a high degree of parallelism.
density: determines the number of dependencies between tasks of two consecutive DAG levels.
regular: the regularity of the distribution of tasks between the different levels of the DAG.
jump: the maximum number of levels spanned by inter-task communications. This allows to generate DAGs with execution paths of different lengths.

Regarding the computation cost and data transfer cost, we specify them by setting the following parameters:

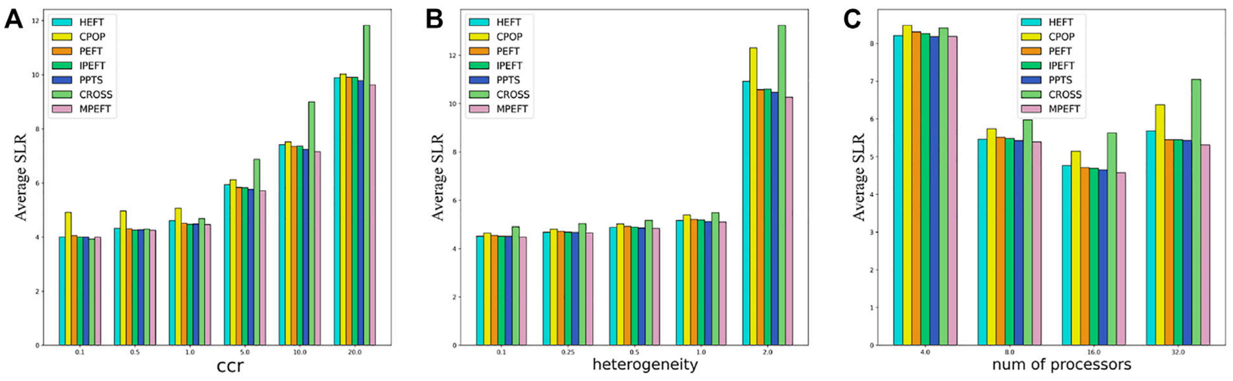


FIGURE 5
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for random DAGs.

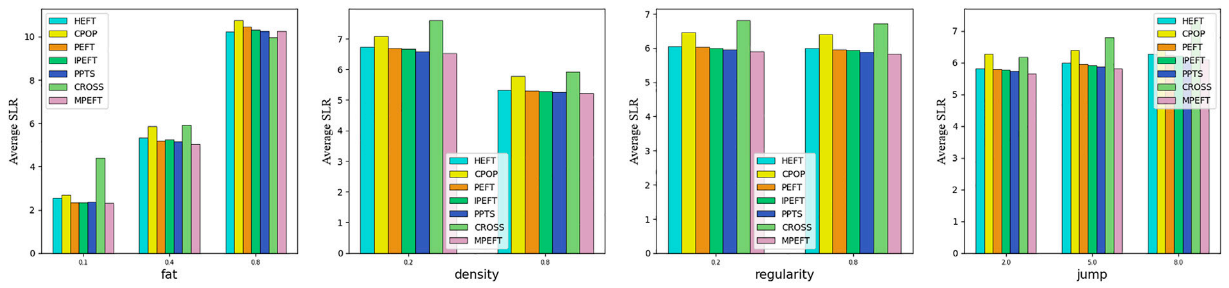


FIGURE 6
Average SLR as function of fat, density, regularity and jump for random DAGs.

communication-to-computation ratio (ccr): the ratio of the sum of edge weights to the sum of the task computation cost in a DAG.

heterogeneity: the *heterogeneity factor* for processor speeds. A high heterogeneity value indicates higher heterogeneity and different computation costs among processors. A low value indicates that the computation costs on processors for a given task are almost equal.

The average computation cost of a task t_i in a given graph $\overline{w_i}$ is selected randomly from a uniform distribution with range $[0, 2\overline{w_{DAG}}]$, where $\overline{w_{DAG}}$ is the average computation cost of a given graph that is obtained randomly. The computation cost of each task t_i on each processor p_j is randomly set from the range of the following equation:

$$\overline{w_i} \times \left(1 - \frac{heterogeneity}{2}\right) \le w_{i,j} \le \overline{w_i} \times \left(1 + \frac{heterogeneity}{2}\right) \tag{14}$$

In this experiment, we use the parameter in Table 5 to generate workflow scheduling scenarios randomly. These parameters generate a total of 38,880 different parameter combinations, and for each parameter combination, 10 DAG graphs are randomly generated, so this part has a capacity of 388,800 DAGs for experiments.

5.1.2 Result

Figure 4 shows the average SLR of all algorithms for different DAG sizes. The CROSS algorithm performed worst because of its downward priority calculation method. The CPOP algorithm performed badly because it pays too much attention to the critical path. It may work well in extreme cases where most tasks of workflow are on the critical path. There is a gap between the performance of HEFT, PEFT, IPEFT, PPTS and MPEFT, when the number of tasks is less than 400. When the number of tasks is greater than 400, the average SLRs of these five algorithms are

TABLE 6 The NOBQS of the Scheduling Algorithms for random DAGs.

		MPEFT	CPOP	HEFT	PEFT	IPEFT	PPTS	CROSS	Combined (%)
MPEFT	better	*	81%	62%	63%	45%	55%	73%	63
	equal		3%	4%	19%	23%	8%	1%	10
	worse		16%	34%	18%	32%	37%	26%	27
CPOP	better	16%	*	30%	26%	21%	24%	46%	27
	equal	3%		2%	2%	3%	2%	1%	2
	worse	81%		68%	72%	76%	74%	53%	71
HEFT	better	34%	68%	*	45%	39%	35%	75%	49
	equal	4%	2%		2%	7%	4%	2%	3
	worse	62%	30%		53%	54%	61%	23%	48
PEFT	Better	18%	72%	53%	*	26%	42%	66%	46
	equal	19%	2%	2%		18%	10%	1%	9
	worse	63%	26%	45%		56%	48%	34%	45
IPEFT	Better	32%	76%	54%	56%	*	48%	73%	56
	equal	23%	3%	7%	18%		9%	1%	10
	worse	45%	21%	39%	26%		43%	26%	34
PPTS	Better	37%	74%	61%	48%	43%	*	74%	56
	equal	8%	2%	4%	10%	9%		1%	6
	worse	55%	24%	35%	42%	48%		25%	38
CROSS	better	26%	53%	23%	34%	26%	25%	*	31
	equal	1%	1%	2%	1%	1%	1%		1
	worse	73%	46%	75%	66%	73%	74%		68

however not much different. This may be due to their similar task priority calculation methods. The MPEFT algorithm outperforms the other four algorithms in every task number.

Figure 5 shows the influence of ccr, heterogeneity, and the number of processors. When the ccr is low, there is no obvious difference between these algorithms except CPOP. With the increase in ccr, the MPEFT algorithm begins to show better performance, and the gap from the baselines becomes more obvious. This shows that MPEFT is more suitable for scenarios with large data transmission between tasks since MPEFT considers all subsequent data transmission costs of the task in the task prioritization stage.

In contrast, other algorithms ignore the data transmission costs other than the longest path from the task to the exit task. When the ccr is low, these costs can be ignored, while as the ccr increases, the data transmission requirements between tasks become larger, and the benefits of MPEFT in task sorting will become clear. Similarly, the performance of MPEFT also improves significantly compared with other algorithms under different heterogeneities and numbers of processor conditions.

Figure 6 shows the influence of fat, density, regularity and jump on the algorithm performance. It can be found that the structural parameters of the DAG have no obvious impact on the scheduling performance.

Table 6 shows the NOBQS results. As can be seen, MPEFT performs better than CPOP, HEFT, PEFT, IPEFT, PPTS and CROSS on most DAG graphs. Compared to IPEFT, the best-performing algorithm among other algorithms, MPEFT also performs worse on only 32% of the DAGs.

5.2 Real-world application graphs

We evaluate the performance of workflows that appear in some practical applications, including epigenomics, montage, ligo and cybershake. The workflow for these applications is presented in Figure 7, and for the details,

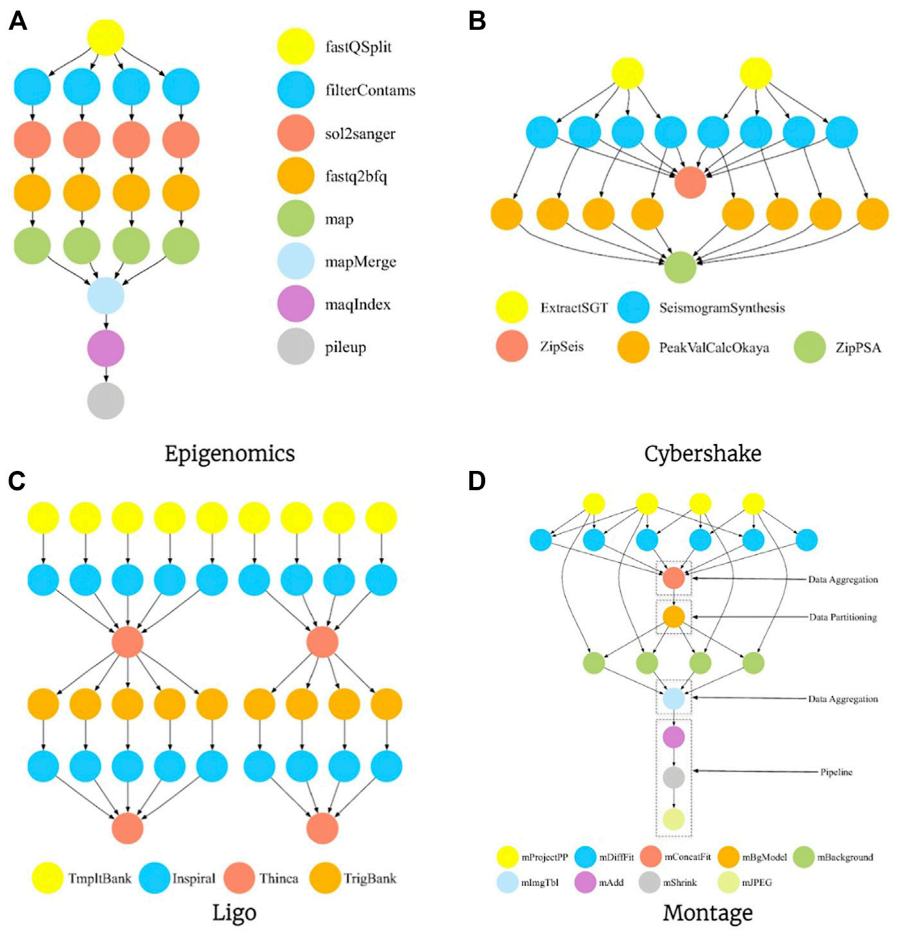


FIGURE 7
The structures of four classes of scientific workflow.

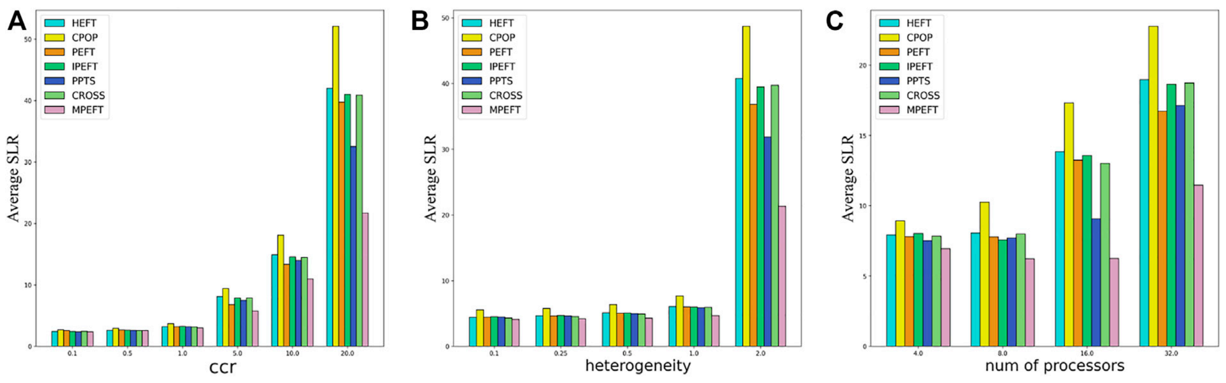


FIGURE 8
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for Epigenomic Workflows.

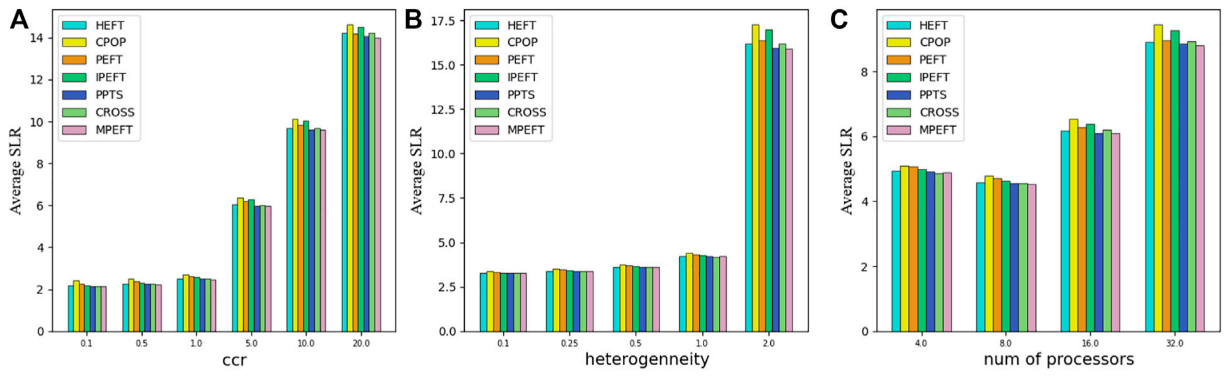


FIGURE 9
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for Cybershake Workflows.

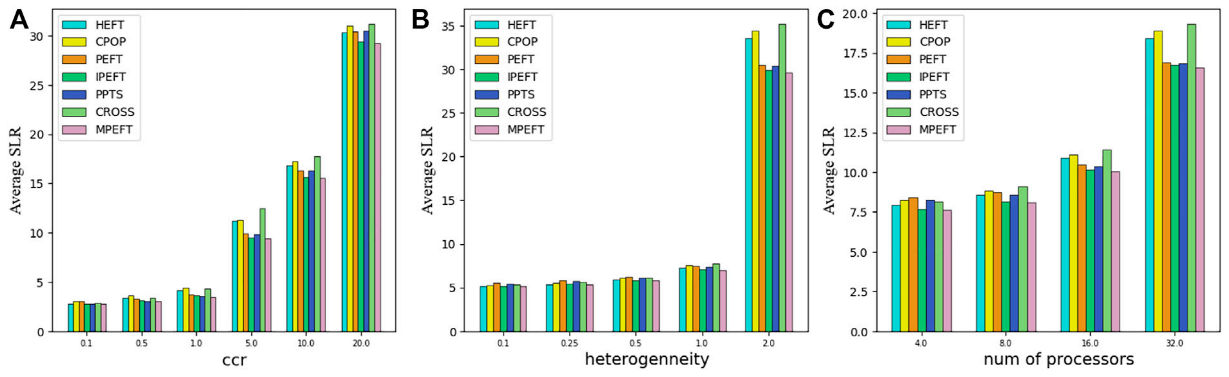


FIGURE 10
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for Ligo Workflows.

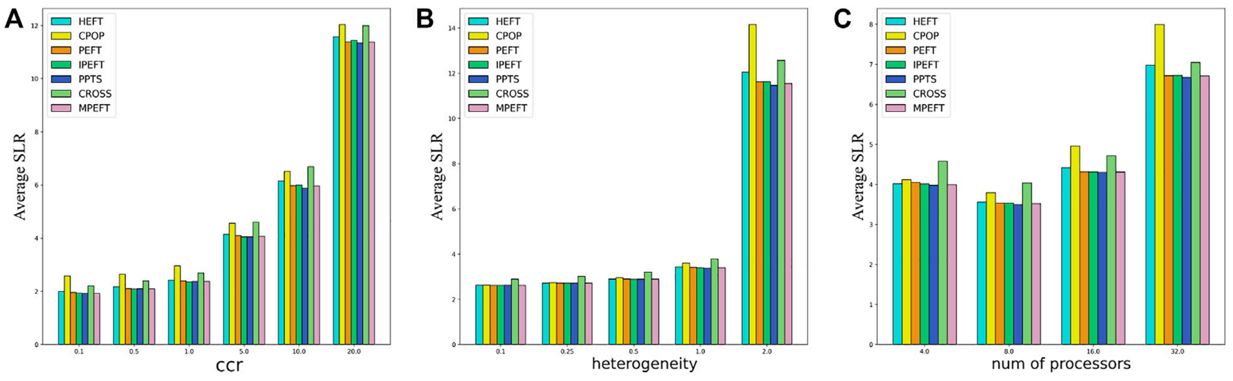


FIGURE 11
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for Montage Workflows.

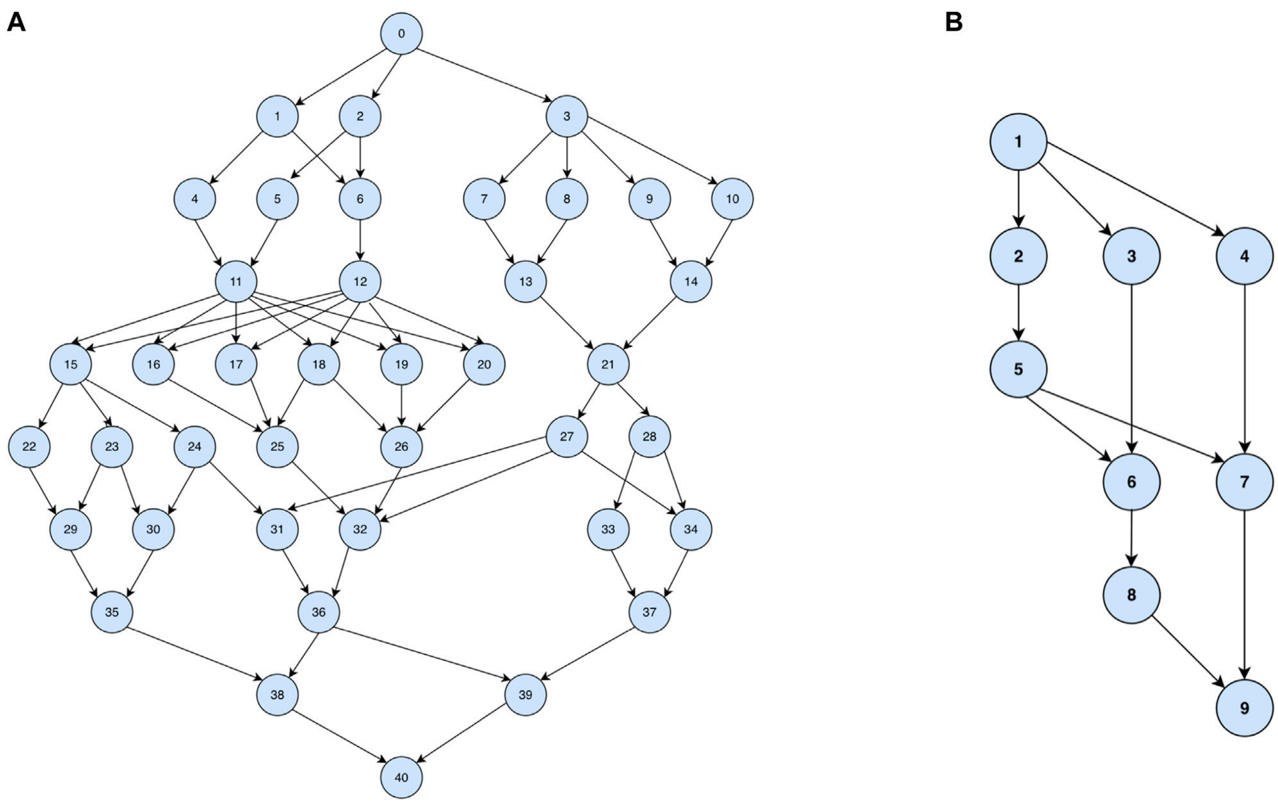


FIGURE 12
DAG of molecular dynamic code **(A)** and Gaussian elimination **(B)**.

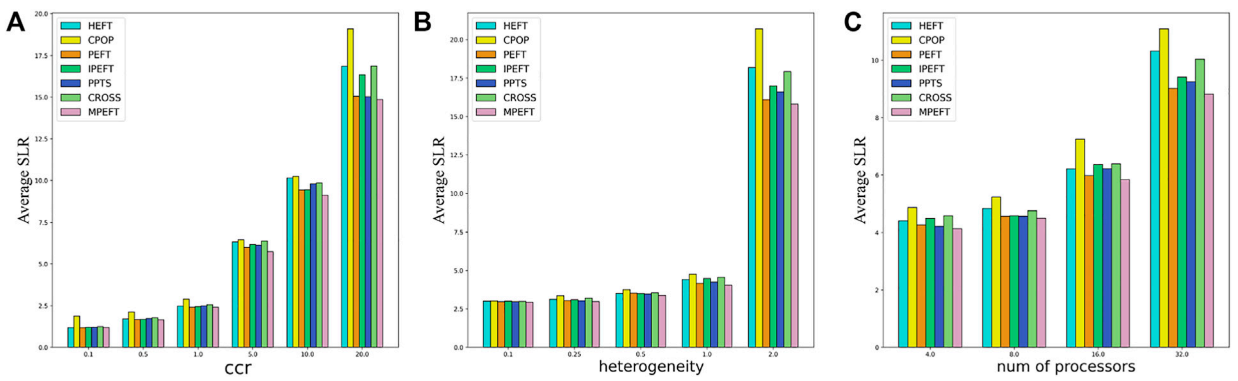


FIGURE 13
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for Molecular Dynamics Workflows.

please refer to (Djigal et al., 2020). In the experiment, we used epigenomics, montage, ligo and cybershake workflows containing 50, 100, 200 tasks to evaluate the algorithms. And we also evaluated the performance of our algorithm on workflows of Gaussian elimination and molecular dynamics code.

5.2.1 Epigenomic workflow

Figure 7A depicts the structure of an epigenomic workflow with 20 tasks. Figure 8A shows the average SLR under different ccr conditions. When ccr <5, the performance of the five algorithms is relatively close, and the MPEFT algorithm performs slightly better. However, when ccr is

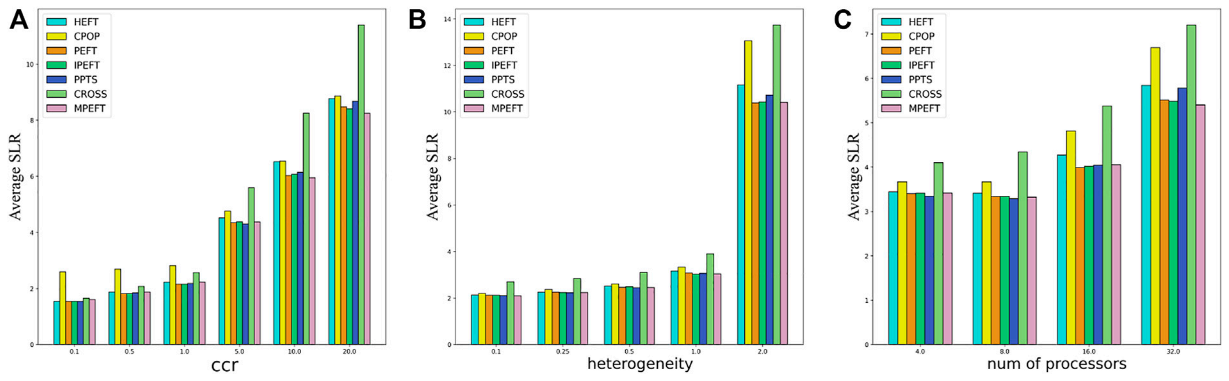


FIGURE 14
Average SLR as function of ccr **(A)**, heterogeneity **(B)** and number of processors **(C)** for Gaussian Elimination Workflows.

greater than 5, the MPEFT outperforms other algorithms significantly, especially when ccr is equal to 20. For various heterogeneities, MPEFT also outperforms other algorithms and improves significantly when heterogeneity is 2.0. Finally, MPEFT performs well for different numbers of processors. As shown in Figure 8C, by analyzing the characteristics of the Epigenomic workflow, it can be found that there is no critical path but several paths of the same process. Therefore, the algorithms such as CPOP, HEFT, *etc.* Tend to perform worse under this workflow since they pay more attention to the critical path. The MPEFT reduces the impact of the critical path on the allocated processor by calculating the k value and achieves better results.

5.2.2 Cybershake workflow

In the simulated scheduling of cybershake workflow, our algorithm outperforms other list scheduling algorithms in most scenarios, as shown in Figure 9. As can be seen from Figure 7B, the workflow has no obvious critical path and the paths are short, so the scheduling performance based on the critical path is not satisfied. Among them, the performance of PPTS is close to that of MPEFT. This is because the PCM of PPTS takes into account the case where the current task and immediate successor tasks are placed on the same processor. In a DAG with a flat structure, PCM can optimize the scheduling strategy. Since there is no obvious critical path, the processor allocation strategy of MPEFT is similar to that of HEFT, that is, the successor tasks are not considered when assigning tasks.

5.2.3 Ligo workflow

In Ligo workflow, MPEFT has better scheduling efficiency, as shown in Figure 10. The structure of Ligo is shown in Figure 7C. When scheduling the task represented by the yellow node in the first row, since there is only one successor node, MPEFT will schedule based on the critical path. When scheduling the task represented by the red node in the third row, since it has many paths to the exit task, MPEFT will schedule it to the node that can complete the task as soon as possible, speeding up the execution of all immediate successor tasks.

5.2.4 Montage workflow

The Montage Workflow is the worst-performing workflow type for the MPEFT algorithm in our experiments, as shown in Figure 11. On this Workflow, the performance of the five algorithms is very close in general except CPOP and CROSS. Compared with other algorithms, MPEFT only improves by 1%–3% in most scenarios. And MPEFT performs slightly worse than PPTS. Because tasks in the latter part of this kind of workflow are connected one by one, which results in almost no difference in $rank_{AP}$ of predecessors of these tasks. The same is true for other algorithms. Thus all algorithms achieve similar results in the task prioritizing phase.

5.2.5 Molecular dynamics code

Figure 12A depicts the structure of Molecular Dynamic Code workflow. The experimental results of Molecular Dynamics Code are shown in Figure 13. MPEFT also outperforms the other four algorithms in different parameter environments. As with the previous experimental results, MPEFT performs better and better with increasing ccr. when ccr = .1, MPEFT performs very close to HEFT, PEFT, and IPEFT. But when ccr = 20, the MPEFT performs significantly better than other algorithms. Figure 13B shows that the MPEFT algorithm outperforms most algorithms in different heterogeneity situations. Especially when heterogeneity = 2.0, compared with HEFT, CPOP, PEFT, IPEFT, PPTS and CROSS algorithms, there are 13.1%,

TABLE 7 Experimental results in terms of makespan standard deviation on real-world workflows.

Algorithm	Cybershake ($\times10^4$)			Epigenomics ($\times10^6$)			Ligo ($\times10^5$)			Montage ($\times10^3$)		
	50	100	200	50	100	200	50	100	200	50	100	200
HEFT	**2.126**	3.325	4.427	**2.090**	2.445	8.333	3.478	3.550	4.394	8.458	10.89	18.43
CPOP	2.454	3.687	5.552	3.045	4.738	10.543	3.505	3.879	5.121	8.238	10.90	18.94
PEFT	2.148	3.360	4.248	2.160	2.454	8.232	3.492	3.580	4.397	**7.886**	9.393	16.12
IPEFT	2.170	3.365	4.257	2.228	2.498	7.958	3.410	3.299	4.191	7.898	**9.348**	16.11
PPTS	2.172	3.351	4.225	2.161	2.395	8.034	3.521	3.657	4.426	7.897	9.351	**15.99**
CROSS	2.182	**3.320**	**4.199**	2.097	**2.303**	8.225	**3.398**	3.410	4.22	8.293	10.52	17.76
MPEFT	2.156	3.354	4.233	2.137	2.466	**6.643**	3.400	**3.295**	**4.03**	7.910	9.365	16.08

That the bold values indicate the best results of the experiments.

TABLE 8 Experimental Results in Terms of Average Running Time (in usec) on Real-World Workflows.

Algorithm	Cybershake			Epigenomics			Ligo			Montage		
	50	100	200	50	100	200	50	100	200	50	100	200
HEFT	**445**	**969**	**1,458**	**427**	**1,046**	1,622	**484**	1,057	**1,502**	**498**	**1,163**	**1,694**
CPOP	473	1,058	1,540	466	1,083	**1,545**	496	**1,047**	1,506	552	1,305	1803
PEFT	555	1,185	1783	520	1,237	1711	581	1,235	1772	596	1,360	2072
IPEFT	645	1,409	2063	656	1,419	2091	723	1,473	1952	774	1,523	2,164
PPTS	560	1,208	1725	539	1,238	1738	577	1,189	1,578	673	1,471	1971
CROSS	509	1,034	1,501	489	1,187	1,687	524	1,103	1,561	589	1,270	1872
MPEFT	616	1,320	1818	693	1,410	1962	1,045	2035	2,624	809	1,641	2,204

That the bold values indicate the best results of the experiments.

23.6%, 1.9%, 7.0%, 4.8% and 11.8% improvements, respectively. Under the condition of a different number of processors, the performance of the MPEFT algorithm is slightly better than other algorithms. These result shows that MPEFT still performs well even when the workflow structure is chaotic and irregular.

5.2.6 Gaussian elimination

Figure 12B gives the graph of Gaussian Elimination workflow for the special case of $m = 4$, where m is the dimension of the matrix. On the Gaussian Elimination workflow, our algorithm also does not perform particularly well. As shown in Figure 14, only when the ccr is 10 and 20, the MPEFT algorithm has obvious advantages. Under most other conditions, the MPEFT algorithm only outperforms the CPOP algorithm. By analyzing the structural characteristics of Gaussian Elimination workflow, we believe that this is due to the particularity of the critical path of this type of workflow. In the Gaussian Elimination workflow, its critical path is very clear and important. However, since there are many other paths from the entry task to the exit task, the cost is not much lower than the critical path. The MPEFT algorithm may incorrectly estimate the importance of the critical path due to the influence of other paths.

5.3 Comparison of robustness

We carried out experiments on the makespan standard deviation of the algorithms on four scientific workflows under number of tasks, and the results are shown in Table 7. CPOP algorithm has the worst robustness because it relies heavily on critical paths of workflows. HEFT and CROSS did not perform well on montage workflows. The second half of the montage workflow is a single chain structure as shown in Figure 7 (d), that is, the critical path is determined. HEFT and CROSS lack consideration of critical path, resulting in poor performance. PEFT, IPEFT, PPTS and MPEFT have similar makespan standard deviations in most scenarios. And on epigenomics workflows, MPEFT has an obvious advantage when the number of tasks is large.

5.4 Comparison of running times

The average running time of the algorithm for the number of four scientific workflow tasks is shown in Table 8. HEFT, CPOP, and CROSS perform similarly in performance. Because they do not consider different processors when calculating priorities, the performance at this stage is better than other algorithms. The other four algorithms have similar performance because they calculate priority based on the processor.

6 Conclusion

This paper proposes a new list-based static scheduling algorithm based on a heterogeneous platform, MPEFT. First, MPEFT will sort all tasks in the workflow based on priority, which is determined by the cost of all offspring of the task. Then, MPEFT will allocate the sorted tasks to the corresponding processors in turn based on the number of processors, the number of successors, and the priority of successors. Compared with the current list scheduling algorithm based on the critical path, MPEFT solves the problem that excessive attention to the critical path leads to reduced scheduling efficiency. Experiments on SLR and NOBQS show that MPEFT performs well in terms of makespan compared with other list-based scheduling algorithms. And through makespan standard deviation and running time experiments, MPEFT also has good robustness and operational efficiency.

The current optimization objective of MPEFT is limited to makespan. Our one future research is to expand MPEFT into a multi-objective optimization algorithm, such as optimization for resource utilization.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

The authors would like to thank the reviewers for their helpful comments.

Funding

This work is supported by the National Key R&D Program of China (2021YFB2104800) and the National Science Foundation of China (62177002).

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Arabnejad, H., and Barbosa, J. G. (2013). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distributed Syst.* 25, 682–694.

Boeres, C., Filho, J. V., and Rebello, V. E. F. (2004). "A cluster-based strategy for scheduling task on heterogeneous processors," in *16th symposium on computer architecture and high performance computing* (IEEE), 214–221.

Chen, H., Zhu, X., Liu, G., and Pedrycz, W. (2018). Uncertainty-aware online scheduling for real-time workflows in cloud service environment. *IEEE Trans. Serv. Comput.* 14, 1167–1178. doi:10.1109/tsc.2018.2866421

Djigal, H., Feng, J., Lu, J., and Ge, J. (2020). Ippts: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. *IEEE Trans. Parallel Distributed Syst.* 32, 1057–1071. doi:10.1109/tpds.2020.3041829

Djigal, H., Feng, J., and Lu, J. (2019). Task scheduling for heterogeneous computing using a predict cost matrix. In Proceedings of the 48th International Conference on Parallel Processing: Workshops, 1–10.

Duan, Q., Quynh, N. V., Abdullah, H. M., Almalaq, A., Do, T. D., Abdelkader, S. M., et al. (2020). Optimal scheduling and management of a smart city within the safe framework. *IEEE Access* 8, 161847–161861. doi:10.1109/access.2020.3021196 frs69wq (2012). *Daggen: A synthetic task graph generator*. Available at: https://github.com/frs69wq/daggen (Accessed December 18, 2022).

Houssein, E. H., Gad, A. G., Wazery, Y. M., and Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. *Swarm Evol. Comput.* 62, 100841. doi:10.1016/j.swevo.2021.100841

Hu, B., Cao, Z., and Zhou, M. (2019). Scheduling real-time parallel applications in cloud to minimize energy consumption. *IEEE Trans. Cloud Comput.* 10, 662–674. doi:10.1109/tcc.2019.2956498

Li, H., Wang, B., Yuan, Y., Zhou, M., Fan, Y., and Xia, Y. (2021a). Scoring and dynamic hierarchy-based nsga-ii for multiobjective workflow scheduling in the cloud. *IEEE Trans. Automation Sci. Eng.* 19, 982–993. doi:10.1109/tase.2021.3054501

Li, H., Wang, D., Zhou, M., Fan, Y., and Xia, Y. (2021b). Multi-swarm co-evolution based hybrid intelligent optimization for bi-objective multi-workflow scheduling in the cloud. *IEEE Trans. Parallel Distributed Syst.* 33, 2183–2197. doi:10.1109/tpds.2021.3122428

Li, J. (2020). Resource optimization scheduling and allocation for hierarchical distributed cloud service system in smart city. *Future Gener. Comput. Syst.* 107, 247–256. doi:10.1016/j.future.2019.12.040

Madhura, R., Elizabeth, B. L., and Uthariaraj, V. R. (2021). An improved list-based task scheduling algorithm for fog computing environment. *Computing* 103, 1353–1389. doi:10.1007/s00607-021-00935-9

NoorianTalouki, R., Shirvani, M. H., and Motameni, H. (2022). A heuristic-based task scheduling algorithm for scientific workflows in heterogeneous cloud computing platforms. *J. King Saud University-Computer Inf. Sci.* 34, 4902–4913. doi:10.1016/j.jksuci.2021.05.011

Pham, T.-P., and Fahringer, T. (2020). Evolutionary multi-objective workflow scheduling for volatile resources in the cloud. *IEEE Trans. Cloud Comput.* 10, 1780–1791. doi:10.1109/tcc.2020.2993250

Sulaiman, M., Halim, Z., Waqas, M., and Aydın, D. (2021). A hybrid list-based task scheduling scheme for heterogeneous computing. *J. Supercomput.* 77, 10252–10288. doi:10.1007/s11227-021-03685-9

Tong, Z., Chen, H., Deng, X., Li, K., and Li, K. (2020a). A scheduling scheme in the cloud computing environment using deep q-learning. *Inf. Sci.* 512, 1170–1191. doi:10.1016/j.ins.2019.10.035

Tong, Z., Deng, X., Chen, H., Mei, J., and Liu, H. (2020b). Ql-heft: A novel machine learning scheduling scheme base on cloud computing environment. *Neural Comput. Appl.* 32, 5553–5570. doi:10.1007/s00521-019-04118-8

Topcuoglu, H., Hariri, S., and Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. parallel distributed Syst.* 13, 260–274. doi:10.1109/71.993206

Tuli, S., Casale, G., and Jennings, N. R. (2021). Mcds: Ai augmented workflow scheduling in mobile edge cloud computing systems. *IEEE Trans. Parallel Distributed Syst.* 33, 1–2807. doi:10.1109/tpds.2021.3135907

Wang, H., and Sinnen, O. (2018). List-scheduling versus cluster-scheduling. *IEEE Trans. Parallel Distributed Syst.* 29, 1736–1749. doi:10.1109/tpds.2018.2808959

Wang, Y., and Zuo, X. (2021). An effective cloud workflow scheduling approach combining pso and idle time slot-aware rules. *IEEE/CAA J. Automatica Sinica* 8, 1079–1094. doi:10.1109/jas.2021.1003982

Wu, Q., Zhou, M., and Wen, J. (2021). Endpoint communication contention-aware cloud workflow scheduling. *IEEE Trans. Automation Sci. Eng.* 19, 1137–1150. doi:10.1109/tase.2020.3046673

Wu, Q., Zhou, M., Zhu, Q., Xia, Y., and Wen, J. (2019). Moels: Multiobjective evolutionary list scheduling for cloud workflows. *IEEE Trans. Automation Sci. Eng.* 17, 166–176. doi:10.1109/tase.2019.2918691

Zheng, X., Li, M., and Guo, J. (2021). Task scheduling using edge computing system in smart city. *Int. J. Commun. Syst.* 34, e4422. doi:10.1002/dac.4422

Zhou, N., Qi, D., Wang, X., Zheng, Z., and Lin, W. (2017). A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. *Concurrency Comput. Pract. Exp.* 29, e3944. doi:10.1002/cpe.3944