The Journal of Supercomputing (2021) 77:11597–11624
https://doi.org/10.1007/s11227-021-03742-3

A novel cloud workflow scheduling algorithm based on stable matching game theory

Zhao-hong Jia[1,2] · Lei Pan[1,2] · Xiao Liu[3] · Xue-jun Li[2]

Accepted: 13 March 2021 / Published online: 27 March 2021

Abstract
Workflow scheduling is one of the most popular and challenging problems in cloud computing. However, among the studies on cloud workflow scheduling, very few consider the fairness among workflow tasks which could significantly delay the workflows and hence deteriorates user satisfaction. In this paper, we propose a workflow scheduling algorithm based on stable matching game theory to minimize workflow makespan and ensure the fairness among the tasks. The local optimization methods based on critical path and task duplication are developed to improve the performance of the algorithm. In addition, a novel evaluation metric is proposed to measure the fairness among workflow tasks. Comprehensive experiments are conducted to compare the performance of the proposed algorithm with other four representative algorithms. Experimental results demonstrate that our algorithm outperforms the other compared algorithms in terms of all three performance metrics under different workflow applications.

Keywords Cloud computing · Workflow scheduling · Game theory · Makespan · Fairness

Zhao-hong Jia and Lei Pan have contributed equally to this work.

✉ Xiao Liu
xiao.liu@deakin.edu.au

Zhao-hong Jia
zhjia@mail.ustc.edu.cn

Lei Pan
panlei_ahu@foxmail.com

Xue-jun Li
xjli@ahu.edu.cn

1 Key Lab of Intelligent Computing and Signal Processing of Ministry of Education, Hefei, China

2 School of Computer Science and Technology, Anhui University, Hefei 230039, China

3 School of Information Technology, Deakin University, Geelong, Australia

1 Introduction

Cloud computing [1] is a service that efficiently provides a large number of computing resources and automated resource management through software. Users can access unlimited resources through the network on a pay-as-you-go fashion [2] without being restricted by time and space. Moreover, cloud resources have many outstanding characteristics such as elasticity, on-demand provisioning and virtualization. Furthermore, there are three major types of cloud services, viz. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [3]. Workflows can represent real-world complex applications such as in the areas of physics, bio-informatics, weather forecasting and astronomy [4, 5]. Most of the workflow scheduling problems in clouds aims at makespan minimization which is known to be NP-complete in the cloud environment [6]. Proper scheduling can effectively reduce energy consumption of the cloud data center, increase resource utilization, reduce workflow makespan and improve user satisfaction. Therefore, how to schedule workflows efficiently on cloud workflow scheduling is of great significance.

The scheduling algorithms can be generally divided into heuristic and meta-heuristic. The former is able to provide approximate solutions with polynomial time complexity. However, the latter, based on iterative search, can improve solution quality with more computation time. Heuristic algorithms can be further divided into list, duplication and clustering scheduling. Meta-heuristic algorithms mainly include genetic algorithm (GA), particle swarm optimization algorithm (PSO), memetic algorithm (MA) and so on.

However, most cloud workflow scheduling algorithms focus on the common objectives such as minimizing the total cost or makespan of the whole workflow. In reality, there are many workflows such as video surveillance, object tracking [7] and face recognition [8] where specific tasks in the workflow have their own objectives such as minimum response time or maximum processing speed. Given a specific scheduling algorithm, if the current best resources (e.g., with the maximum bandwidth and maximum processing speed) are always assigned to the tasks according to their priorities, some tasks may fail to meet the customers' requirements, which is unfair. Therefore, unfair allocation of resources will lead to a significant decrease on the satisfaction of some task objectives, which in turn will affect customer satisfaction in the cloud services. Therefore, the fairness among the workflow tasks needs to be considered along with the global workflow objectives [9–11]. However, there are few studies on task fairness in workflow scheduling. Inspired by the widely adoption of stable matching game theory (SMGT) in balancing fairness and competition among tasks, this paper proposes a workflow scheduling algorithm in heterogeneous cloud environments based on SMGT [12, 13] to minimize workflow makespan and ensure the fairness among the tasks. To the best of our knowledge, this is the first attempt of SMGT on workflow scheduling problems and has achieved relatively good results. Specifically, in order to improve the efficiency, two local optimization methods based on critical path [14] and task duplication [15–17] are incorporated into the algorithm. The target of our algorithm is to distribute multiple cloud workflows with different sizes to virtual machines (VMs) with different processing capacities in the cloud to minimize the workflow makespan and maximize the fairness among the workflow tasks.

The major contributions of this paper are as follows:

- The fairness of workflow tasks is considered and a novel scheduling algorithm based on SMGT is proposed to balance between the global objectives of the workflows and the fairness of the workflow tasks.
- Local optimization methods based on critical path and task duplication are designed to improve the efficiency of the algorithm.
- A novel evaluation metric is proposed to measure the fairness among workflow tasks.
- The proposed algorithm is compared with other four representative algorithms through comprehensive experiments on four typical workflow structures.

The remainder of the paper is organized as follows. Section 2 reviews the workflow scheduling of Cloud in recent years. Section 3 defines the workflow model, the cloud model and the problem formulation. Section 4 describes the proposed algorithm, as well as an example, in detail. The experimental results and the analysis on the results are provided in Sect. 5. Section 6 concludes the paper.

2 Literature review

In recent years, a lot of studies on workflow scheduling in the cloud have been proposed. Generally speaking, cloud workflow scheduling algorithms can be divided into two categories including heuristics and meta-heuristics (see Table 1). Heterogeneous earliest finish time (HEFT) [14] is one of the most popular heuristic algorithms. It minimized the completion time of the workflow by assigning an upward rank value to all tasks and insertion approach. In addition, another method of prioritizing tasks with the sum of upward and downward rank values was proposed to calculate the rank value of tasks called Critical-Path-on-a-Processor (CPOP). At the stage of processor selection, priority is incorporated to schedule the critical tasks. Experimental results have verified the superiority of the two methods in solution quality and search speed. Based on HEFT, many improved algorithms have been proposed. Samadi et al. [18] presented an enhancement heterogeneous earliest finish

Table 1 Classification of workflow scheduling algorithms

	Objective				
	Makespan	Cost	Energy	Flowtime	Fairness
Heuristics	[14, 18–29]	[20, 21, 24]	[22]		
Meta-heuristics	[31–34, 36]	[31–36]	[35]	[31, 32]	
Game theory	[39, 40]	[38–40]	[37]		[39]

time algorithm (E-HEFT) taking into account the users' financial constraints and load balancing between VMs to minimize the makespan of workflow. Cao et al. [19] presented the Top-k strategy, called K-HEFT, where, in each assignment, the *k* most schedulable tasks are selected, according to their priority, to be allocated, in order to reduce the minimum completion time of the workflow.

Considering the constraint of deadline, the variability of performance of VMs and the delay of instance acquisition, Sahni et al. [20] proposed a heuristic algorithm to minimize the cost and makespan. Zheng et al. [21] proposed three different algorithms. The first algorithm, called CFMax (Change From Maximum), generates schedule using algorithm HEFT and then redistributes tasks based on cost reduction. The second, called CRR (cost–runtime ratio), considers makespan reduction based on the result of CFMax. The third, CBT-MD (Compete between tasks by Manhattan distance) and CBT-ED (compete between tasks by Euclidean distance) conduct global evaluation through two different measurements on pre-schedule. All of these algorithms can reduce the execution cost of workflow without violating the constraint of deadline. Moreover, CBT-MD and CBT-ED perform best among these algorithms, meaning that the reassigned schedule by the global evaluation helps to achieve better results. Wu et al. [22] proposed a soft error-aware task scheduling approach for workflows in dynamic voltage and frequency scaling cloud data centers. The approach can generate energy-efficient task schedules for workflows under the constraints of reliability and completion time. Ijaz and Munir [23] proposed a list scheduling heuristic with optimized duplication to reduce the makespan of workflow. The proposed algorithm improves the performance measurements and ensures the same time complexity as those of the existing algorithms. Zhang et al. [24] first described the workflow scheduling problem as an integer programming problem. They proposed a job-prioritization scheme that maintains the priorities of tasks with precedence constraints and assigns priorities to tasks without precedence constraints based on their importance. They finally designed a uniform spare budget-splitting strategy that splits the spare budget uniformly, resulting in better extra demand on average and improvement on makespan of workflow. Djigal et al. [25] proposed an improved predict priority task scheduling algorithm with two phases: task prioritization and processor selection. The algorithm reduced the makespan of workflow significantly by looking ahead in two phases. Geng et al. [26] proposed a new algorithm based on task duplication and task grouping to minimize the total execution time of workflow. Kumar et al. [27] proposed a novel algorithm based on the granularity of tasks in a workflow to minimize the makespan and maximize the average VM utilization. Gupta et al. [28] calculate the dynamic threshold of tasks through min–max normalization to ensure the makespan and the cloud resource utilization of each workflow. Min–min [29] is one of the common benchmark workflow scheduling algorithms that can intuitively reflect the performance of different algorithms.

At present, most heuristic algorithms are focusing on optimizing one single objective with certain constraints, which is ideal [30]. To solve the multi-objective problems of workflow scheduling, which are closer to reality, meta-heuristic algorithms have exhibited their superiorities. Taking the costs of both data transmission and computation into consideration, Wu et al. [31] proposed a revised discrete PSO to schedule the workflows. In order to minimize three objectives, i.e., flowtime, makespan and resource usage cost, Kaur and Kadam [32] extended the original bacteria foraging algorithm and a new fitness assignment method and bacteria selection strategy are incorporated to optimize multiple objectives, simultaneously. However, their method considers independent tasks, which is unsuitable for workflow scheduling problems. Therefore, on the basis of PSO, Hu et al. [33] proposed a multi-objective scheduling algorithm to minimize makespan and cost of workflow, simultaneously, with considering the constraint of reliability. Moreover, in the encoding method, both the execution location of tasks and the order of tasks for data transmission are taken into consideration. Huang et al. [34] added a heuristic algorithm called simplified swarm optimization to PSO, where dynamic parameters are designed to adjust local and global search abilities of the algorithm. Ding et al. [35] presented a cost-effective scheduling strategy based on PSO for multi-workflow under the constraints of deadline in Fog computing. Alsmady et al. [36] proposed a Memetic algorithm where hill climbing local search is adopted as an extra operator for Genetic Algorithm to improve solutions during global search. The experimental results showed that the MA is able to decrease the makespan and cost of the workflow at the same time.

Compared with heuristics, meta-heuristic algorithms have advantageous in finding better solutions to workflow scheduling problems. However, the computation time of meta-heuristic algorithms is much higher than that of heuristics in general. As a result, when response time is needed or problem scale is larger, the meta-heuristic algorithms are not suitable enough. Therefore, the heuristic algorithms are preferable for addressing the studied problem with one single objective. Game theory (GT) mainly focuses on the strategic interaction among rational decision-makers and is widely applied in all fields of logic, systems science and so on. Considering the reliability of balanced tasks, Yang et al. [37] presented a task scheduling algorithm based on the cooperative game model so that the complexity of the proposed algorithm is reduced, as well as the efficiency being ensured. To solve the task scheduling in computational grid, Gao et al. [38] treated the grid load-balancing problem as non-cooperative game and proposed a GT-based algorithm to minimize the cost of the grid. The experiment results demonstrate that the game-based algorithm has a desirable capability to resolve the task scheduling problem. Wang et al. [39] proposed a multi-objective workflow scheduling algorithm based on dynamic game-theoretic model to minimize makespan and total cost and maximize system fairness in terms of workload distribution among heterogeneous cloud VMs. Sujana et al. [40] formulated the multi-objective scheduling of workflows as a new sequential cooperative game with two objectives of minimizing the execution time and the economic cost on two constraints.

Although GT has shown advantages in workflow scheduling, there are still very few studies considering the issue of task fairness. Hence, in this paper, we apply the stable matching algorithm, one of the models in GT, to solve cloud workflow scheduling with the aim to simultaneously minimize the makespan of workflow and maximize the fairness of workflow tasks.

3 Problem description

To describe the studied problem clearly, the workflow model, the cloud model and the problem formulation are provided in this section. The major notations are firstly given in Table 2.

3.1 Workflow model

A workflow model is usually represented by a directed acyclic graph (DAG) [41–44]. An example of the studied workflow model is illustrated as Fig. 1. In Fig. 1, $DAG = (T, E)$, where $T = \{t_0, t_1, \dots, t_{n-1}\}$ denotes a set of n tasks and E represents the dependencies between tasks. The predecessors set and the successors set of t_i are denoted by $pre(t_i)$ and $succ(t_i)$, respectively. Each task with no predecessor is known as entry task t_{entry}, and each task with no successor is known as exit task t_{exit}. It is notable that there may be more than one entry task or more than one exit task in a DAG model. If all predecessors of one task are completed and the data from

Table 2 List of notations

Notation	Definition
$level$	Level of workflow
n	Number of tasks
id	Index of task
t_i	Task t_i of workflow
s_i	Size of t_i
$TT_{i,j}$	Size of data transmission from t_i to t_j
$preference(t_i)$	Preference array of task t_i
$pre(t_i)$	Predecessor set of t_i
$succ(t_i)$	Successor set of t_i
$T_{trans}(t_i, t_j)$	Transmission time from t_i to t_j
$AST(t_i)$	Actual start time of t_i
$AFT(t_i)$	Actual finish time of t_i
$ET(t_i, VM_k)$	Execution time of t_i on VM_k
$FT(t_i, VM_k)$	Finish time of t_i on VM_k
CP	The set of tasks on critical path
m	Number of VMs
p_k	Processing capacity of VM_k
$threshold(VM_k, l)$	Number of tasks allowed to be performed on VM_k in level l
$preference(VM_k)$	Preference array of VM_k
$waiting(VM_k)$	Waiting array of VM_k
$avail(VM_k)$	Available time of VM_k
$B(VM_k, VM_l)$	Bandwidth between VM_k and VM_l
$makespan$	Makespan of workflow
$utilization$	Utilization of VM

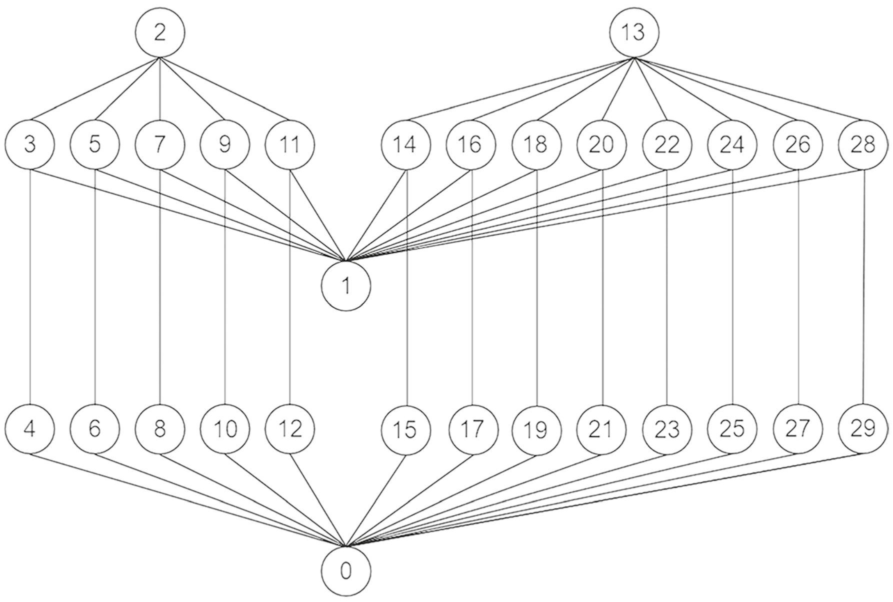

Fig. 1 An example of CyberShake workflow with 30 tasks

predecessors arrive, the task can be started to execute. The size of data transmission between tasks can be represented by a matrix denoted by *TT* as follows:

$$TT_{i,j} = \begin{bmatrix} 0 & TT_{0,1} & \cdots & TT_{0,n-1} \\ TT_{1,0} & 0 & \cdots & TT_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ TT_{n-1,0} & \cdots & \cdots & 0 \end{bmatrix} \tag{1}$$

where $TT_{i,j} = 0$ if there are no constraint relationships between t_i and t_j or $i = j$. The element $TT_{i,j}$, $(0 \le i \le n-1, 0 \le j \le n-1)$ represents the size of data transmission from t_i to t_j.

3.2 Cloud model

Assume the infrastructure to be a service (IaaS) cloud system that consists of a set of m VMs [45], denoted by $V = \{VM_0, VM_1, \ldots, VM_{m-1}\}$. All the VMs are independent of each other and have different processing capabilities. Let B denote a matrix representing the bandwidth between different VMs. Note that this matrix is unnecessarily symmetric. That is, $B(VM_k, VM_l)$ may not be equal to $B(VM_l, VM_k)$. Moreover, it is obvious that $B(VM_k, VM_l) = 0$, if $k = l$. The transmission time, denoted by $T_{trans}(t_i, t_j)$, from t_i executed on VM_k to t_j on VM_l, is determined by the size of data transmission $TT_{i,j}$ from t_i to t_j and bandwidth between VM_k and VM_l. Note that $T_{trans}(t_i, t_j) = 0$, if the two tasks are executed on the same VM. Specifically, $T_{trans}(t_i, t_j)$ is calculated by Eq. (2).

$$T_{trans}(t_i, t_j) = \frac{TT_{i,j}}{B(VM_k, VM_l)} \tag{2}$$

The start time of t_i on VM_k, denoted by $ST(t_i, VM_k)$, is calculated as Eq. (3).

$$ST(t_i, VM_k) = \begin{cases} 0 & t = t_{entry} \\ \max\limits_{t_j \in pre(t_i)} \{FT_j + T_{trans}(t_i, t_j)\} & t \neq t_{entry} \end{cases}. \tag{3}$$

The execution time of t_i on VM_k, denoted by $ET(t_i, VM_k)$, is defined as the ratio between the size of the task and the processing capacity of VM_k and shown as Eq. (4).

$$ET(t_i, VM_k) = \frac{s_i}{p_k}. \tag{4}$$

The finish time of t_i on VM_k, denoted by $FT(t_i, VM_k)$, formulated by Eq. (5).

$$FT(t_i, VM_k) = ST(t_i, VM_k) + ET(t_i, VM_k). \tag{5}$$

3.3 Problem formulation

The makespan of workflow, which is used to evaluate the performance of the proposed algorithm, equals to the maximum completion time of all tasks in the workflow [46–49]. Let $MS(VM_k)$ denote the makespan of VM_k, $0 \le k \le m-1$, then the makespan of workflow is defined as Eq. (6):

$$makespan = \max(MS(VM_k)), 0 \le k \le m-1. \tag{6}$$

In addition, the following metrics, based on makespan, to measure the performance of algorithms more intuitively and be formulated as follows.

(1) Scheduling length ratio (SLR) [50, 51]. In order to avoid large differences in makespan caused by different parameters, it is necessary to normalize the makespan to a lower bound. This metric is the ratio of the makespan of workflow to the sum of the minimum computation time of those tasks in CP. SLR is calculated as Eq. (7).

$$SLR = \frac{makespan}{\sum_{i=0}^{|CP|-1}\left(min\sum_{k=0}^{m-1} ET(t_i, VM_k)\right)}. \tag{7}$$

(2) Average VM utilization (AVU) [52, 53]. It is defined as the ratio of execution time of all tasks on one VM to the makespan of workflow. The utilization rate of VM_k is formulated as Eq. (8):

$$VU(VM_k) = \frac{\sum ET(t_i, VM_k)}{makespan}, i \in VM_k.waiting. \tag{8}$$

And the average utilization of all VMs is defined as Eq. (9):

$$AVU = \frac{1}{m}\sum_{k=0}^{m-1} VU(VM_k). \tag{9}$$

(3) Variance of fairness (*VF*). This is a new metric designed to measure the fairness between tasks. It is defined based on the satisfaction of each task. The ratio of the actual execution time *AET* and the expected execution time *EET* of each task is calculated as the satisfaction of each task *S*. It is formulated as Eq. (10).

$$S_i = \frac{AET_i}{EET_i} \tag{10}$$

where *EET* is the execution time of the task on the fastest VM. Obviously, the higher the value of *S*, the better the task satisfaction. The degree of deviation of all task satisfaction from their average is used to measure the fairness of all tasks. The variance of fairness is formulated is as follows:

$$VF = \frac{1}{n}\sum_{i=0}^{n-1}(M - S_i)^2 \tag{11}$$

where *M* is the average of all task satisfaction. Apparently, the smaller its value, the fairer the algorithm is for all tasks.

4 Proposed algorithm

The proposed algorithm, named SM-CPTD, is based on SMGT. SM-CPTD is designed to effectively coordinate individual and population goals while balancing VMs load and resource utilization. In addition, methods for selection and generating the preference list are presented. To improve solution quality, two local optimization methods, based on critical path and task duplication, are also incorporated into the algorithm. There are three main stages in the proposed algorithm and are shown in Algorithm 1. Firstly, determining the critical path (DCP) is designed to determine the critical path of the DAG. Secondly, the SMGT is applied to generate the pre-schedule scheme. Finally, the pre-schedule scheme is improved by the local optimization strategy based on task duplication (LOTD).

Algorithm 1 Algorithm SM-CPTD

```
Input: workflow $DAG$, VM pool $V$, CCR
Output: Schedule plan $S$ of $DAG$
1: Call Algorithm DCP.
2: Call Algorithm SMGT.
3: Call Algorithm LOTD.
```

4.1 Local optimization based on critical path

The method in HEFT [14] is utilized to calculate the critical path of a DAG. In this method, based on the average computation and communication time, each task is assigned a rank during the process of traversing the entire DAG from the exit tasks to the entry tasks. The rank of t_i, representing the length of the longest path from t_i to the exit task, is recursively defined as Eq. (12):

$$rank(t_i) = \overline{W_i} + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank(t_j)) \tag{12}$$

where $\overline{W_i}$ denotes the average computation time for t_i, and $\overline{c_{i,j}}$ denotes the average communication time of $edge(t_i, t_j)$. For the exit tasks, $rank(t_{exit}) = \overline{W_{exit}}$. Each task with the largest rank in the same level is selected to construct the critical path. Then, the tasks in the critical path are scheduled on the VM that can obtain the earliest completion time because reducing the total execution time of the jobs in the critical path can effectively decrease the makespan of a DAG [14, 15].

DCP is described as follows.

Algorithm 2 Algorithm DCP

```
1: Initialize $CP$ with an empty set.
2: Calculate the rank of $t_{exit}$.
3: for each task $t_i$ in the DAG except the exit task do
4:     Calculate the rank of $t_i$ by Eq. (8).
5: end for
6: for each level $l$ in the DAG do
7:     Select the task with the maximum rank in level $l$ and add the task into $CP$.
8: end for
```

4.2 Stable matching game theory

SMGT is one of the latest theories of modern economics, which can achieve a high degree of integration of fairness and efficiency in allocation public resources. Moreover, it has been successfully applied to solve different types of problems, such as the matrimonial problem, students' school choice problem and so on. Furthermore, SMGT has shown significant advantages in helping participants generate a strict preference order at a lower cost.

A stable matching model in workflow scheduling problem consists of two sets of participants, i.e., the set T of tasks and the set V of VMs, where all individuals in T have a preference matrix for all individuals in V, and all individuals in V have a preference matrix for all individuals in T. The preference matrix of a task is generated based on the ascending order of its finish time on different VMs. Similarly, the preference matrix of a VM for tasks is generated according to the ascending order of their finish time on this VM. The goal is to find a stable match that allows each individual to get the most desired object as possible.

Because of the particularity of workflow scheduling problem, the hierarchical matching method is employed in this paper. First, the tasks in level l are assigned to the most preferred VM according to their preference matrix. In addition, to avoid lots of tasks being assigned to the fastest VM, it is necessary to set the threshold of VMs. That is, task is allowed to be assigned to VM_k if $threshold(k, l) > 0$; otherwise, the task with the largest preference value in the waiting list of VM_k is eliminated and reassigned. The reassigned task selects another VM according to task's preference matrix. Repeat the above steps until all tasks in level l are assigned. $threshold(k, l)$ is calculated based on the number of tasks in each level in the DAG and the processing capacity of each VM so that the more powerful VM can handle more tasks. As a result, the high-speed VM will handle more tasks than the low-speed machine. The threshold of VM_k in level l is given by Eq. (13).

$$threshold(k, l) = \frac{\sum_{v=0}^{l-1} n_v}{\sum_{i=0}^{m-1} p_i} \times p_k \tag{13}$$

where n_l denotes the number of tasks in level l.

Since the order of task assignment has little effect on the result of SMGT, the priority of tasks is ignored in the proposed algorithm. Moreover, once a task t_i is assigned, $ACT(t_i)$ and $AFT(t_i)$ are updated, and preference values for the other tasks are assigned. The SMGT is described as follows.

Algorithm 3 Algorithm SMGT

```
1: Call DCP to determine critical path.
2: for $l$ = 0,1,...,level-1 do
3:    vmPreference($l$) /* Generate the preference array for VMs. */
4:    $task \leftarrow levelTask(l)$. /* The tasks set of the $l^{th}$ level. */
5:    Assign the critical task $t$ in the $l^{th}$ level to $vm$ with the fastest processing speed.
6:    $vm$.waiting.add($t$).
7:    Update($t$). /* Update $t.ACT$ and $t.AFT$. */
8:    $task$.remove($t$).
9:    while $task.size > 0$ do
10:      taskPreference($task$). /* Generate the preference matrix for $task$. */
11:      for $j = 0$; $j <$ $task$.get(0).preference.length; $j$++ do
12:         $u \leftarrow$ $task$.get(0).preference.get($j$). /* $u$ represents a VM.*/
13:         threshold($u$,$i$). /*Calculate the threshold of $u$. */
14:         if $u$.waiting.size $<$ $u$.threshold then
15:            $u$.waiting.add($task$.get(0)).
16:            Update($task$.get(0)).
17:            $task$.remove($task$.get(0)).
18:            Break.
19:         end if
20:         if $u$.waiting.size $=$ $u$.threshold then
21:            $p \leftarrow position(task$.get(0),$u)$. /* Locate $task$.get(0) in $u$'s preference matrix. */
22:            $b \leftarrow largeTask(u)$. /* The task's index with the largest preference value in
               waiting of $u$. */
23:            $q \leftarrow position(task_b,u)$. /* Locate $task_b$ in preference matrix of $u$. */
24:            if $p < q$ then
25:               Replace $b$ by $task$.get(0).
26:               Update $task$.get(0).
27:               $task$.remove($task$.get(0)).
28:               $task$.add($b$).
29:               Break.
30:            else
31:               Continue.
32:            end if
33:         end if
34:      end for
35:   end while
36: end for
```

4.3 Local optimization based on task duplication

The essence of task duplication is to use idle time of VMs to reduce the makespan. That is, the redundant tasks are utilized to reduce the communication time between tasks. Furthermore, in order to avoid excessive waste of space, only consider duplicating tasks in the first level in DAG. Specifically, task t_i can be duplicated and assigned to another VM if both the following two rules are satisfied: (1) t_i can only be duplicated and placed to the VMs where its successors are assigned to. (2) The completion time of the other tasks in the waiting list of the selected VM cannot increase after the duplication of t_i is assigned to this VM.

LOTD is described as Algorithm 4.

Algorithm 4 Algorithm LOTD

```
1: Call DCP and SMGT to generate pre-schedule $S$.
2: for $k = 0, 1, ..., m-1$ do
3:     $t \leftarrow$ the first task in the waiting list of $VM_k$.
4:     if $t.AST$ != 0 then
5:         $minST \leftarrow \infty$, where $minST$ denotes the start time of $t$ after the duplication of its
           predecessor.
6:         $minPredecessor \leftarrow$ the index of the task with $minST$.
7:     end if
8:     for each predecessor $p$ of $t$ do
9:         if $p$ is replicated then
10:            Calculate the start time of $t$, denoted by $st$.
11:        end if
12:        if $st < minST$ then
13:            $minPredecessor \leftarrow p$
14:            $minST \leftarrow st$
15:        end if
16:    end for
17:    if $minST < t.AST$ then
18:        Duplicate and assign the task with $minPredecessor$ to $VM_k$.
19:        Update $AST$ and $AFT$ of each task in DAG.
20:    end if
21: end for
```

4.4 Case study

To illustrate the above process more clearly, an example with CyberShake workflow is provided as shown in Fig. 1. In Fig. 1, there are 30 tasks, denoted by $T = \{t_0, t_1, t_2, \dots, t_{29}\}$, with four levels and 52 edges between the tasks. The sizes of tasks are shown in Table 3, where the rank of each task is calculated according to Eq. (8). The rank in bold in Table 3 is represented the task with the largest rank per level. The matrix TT representing the size of data transmission between tasks is calculated as: $TT_{i,j} = s_{t_i} \times CCR$, where $CCR = 0.4$, and CCR denotes the ratio of

Table 3 Sizes and ranks of tasks

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Size	50	40	46	37	37	39	44	40	44	31
Rank	**7.14**	5.71	**44.69**	22.23	14.67	**35.34**	**27.41**	24.23	16.09	19.55
	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}
Size	30	43	33	42	46	45	46	46	41	49
Rank	13.24	22.6	13.85	34.61	25.66	16.3	25.86	16.5	25.45	17.11
	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}
Size	40	39	47	46	37	43	42	38	39	42
Rank	23.21	15.08	26.06	16.5	23.42	15.89	23.42	14.87	23.62	15.69

i size e rank
non tornano

the average communication cost to the average computation cost. The processing capacity of the five VMs is 5, 8, 7, 9 and 6, respectively. The bandwidths between VMs are shown in Table 4.

According to Table 3, the critical path is $\{t_2, t_5, t_6, t_0\}$. Two cases, that is, level-0 and level-1, are provided and shown in Tables 5 and 6. The task that need to be reassigned is shown in bold in Table 6. Since there is no data transmission from predecessors to the tasks in level-0, only in the case of finish time of tasks, the array of preference for all VMs is the same.

The Gantt chart of the solution obtained by the proposed algorithm SM-CPTD is shown as Table 7. In Table 7, symbols '$*$' and '$\sim$' represent the idle time slot of VM and the duration of the assigned task in the solution, respectively. Moreover, duplicated tasks are shown in boldface.

4.5 Complexity analysis

Here, we present the complexity analysis for SM-CPTD. There are three main procedures in SM-CPTD, namely determining critical path, stable matching game theory and task duplication. Let n, l and m denote the number of total tasks, the total number of levels in the workflow and the number of active VMs, respectively. In Algorithm 2, the time complexity of calculating the rank value for each task and determining the critical path is $O(n^2)$. In Algorithm 3, the loop from Step 2 to Step 35 repeats l times. Step 3 requires $O(n)$. In the ideal case, that is, each task is assigned to the best VM, the loop from Step 9 to Step 34 requires $O(nm)$. However, in the worst case, each matched task is always rejected later, causing n matches to succeed totally, whose time complexity requires $O(n^2)$. It is obvious that the time complexity of Algorithm 4 is $O(nm)$. Therefore, the overall time complexity of SM-CPTD is $O(n^2l)$.

5 Simulation and analysis

In this section, comprehensive experiments are conducted to measure the performance of SM-CPTD with the other four algorithms and to verify the effectiveness of the proposed strategies.

Table 4 Bandwidth between VMs

	VM_1	VM_2	VM_3	VM_4	VM_5
VM_1	0	7	8	9	6
VM_2	5	0	8	7	5
VM_3	7	6	0	8	4
VM_4	7	8	6	0	5
VM_5	9	7	6	4	0

Table 5 Results of level-0

	VM.preference	VM.threshold
VM_1	2, 13	1
VM_2	2, 13	1
VM_3	2, 13	1
VM_4	2, 13	1
VM_5	2, 13	1

Unassigned task	Task	Task.preference	VM	VM.waiting	Task.AST	Task.AFT
2, 13	2		3	2	0	5.11
13	13	2, 3, 5, 1, 4	2	13	0	5.25

Table 6 Results of level-1

	VM.preference	VM.threshold
VM_1	22, 16, 14, 26, 18, 20, 11, 24, 7, 5, 3, 9, 28	3
VM_2	11, 7, 5, 3, 9, 22, 14, 16, 26, 18, 20, 24, 28	4
VM_3	11, 22, 7, 16, 14, 5, 3, 26, 18, 20, 9, 24, 28	3
VM_4	22, 16, 14, 26, 18, 20, 24, 11, 7, 5, 3, 9, 28	4
VM_5	22, 16, 14, 11, 26, 7, 18, 5, 20, 3, 24, 9, 28	3

Unassigned task	Task	Task.preference	VM	VM.waiting	Task.AST	Task.AFT
3, 5, 7, 9, 11, 14, 16, 18, 20, 22, 24, 26, 28	5		4	2, 5	5.11	9.44
3, 7, 9, 11, 14, 16, 18, 20, 22, 24, 26, 28	3	2, 4, 3, 1, 5	2	13, 3	7.36	11.99
7, 9, 11, 14, 16, 18, 20, 22, 24, 26, 28	7	3, 4, 5, 1, 2	3	7	8.11	13.83
9, 11, 14, 16, 18, 20, 22, 24, 26, 28	9	4, 1, 5, 2, 3	4	2, 5, 9	9.44	12.89
11, 14, 16, 18, 20, 22, 24, 26, 28	11	5, 1, 2, 4, 3	5	11	8.71	15.88
14, 16, 18, 20, 22, 24, 26, 28	14	1, 4, 2, 3, 5	1	14	8.45	17.65
16, 18, 20, 22, 24, 26, 28	16	2, 4, 3, 5, 1	2	13, 3, 16	11.99	17.74

c’è un errore sull’assegnazione della task2 o sulla vm 3/4

Table 6 (continued)

Unassigned task	Task	Task.preference	VM	VM.waiting	Task.AST	Task.AFT
18, 20, 22, 24, 26, 28	18	4, 3, 5, 2, 1	4	2, 5, 9, 18	12.89	17.44
20, 22, 24, 26, 28	20	3, 5, 4, 2, 1	3	7, 20	13.83	19.54
22, 24, 26, 28	22	4, 5, 2, 1, 3	4	2, 5, 22, 18	9.44	14.67
24, 26, 28, **9**	24	5, 4, 2, 1, 3	5	11, 24	15.88	22.05
26, 28, **9**	26	2, 4, 3, 1, 5	2	13, 3, 16, 26	17.74	22.99
28, **9**	28	4, 1, 3, 5, 2	1	14, 28	17.65	25.45
9	9	4, 3, 2, 5, 1	3	7, 20, 9	19.54	23.97

Table 7 Gantt chart of SM-CPTD

VM1	0 ~ 8.4	8.4 ~ 17.6	17.6 ~ 25.4	25.4 ~ 34.4	
	$\mathbf{t}_{13}$	t_{14}	t_{28}	t_{15}	
VM2	0 ~ 5.25	5.25 ~ 7.36	7.36 ~ 11.99	11.99 ~ 17.74	17.74 ~ 22.99
	$\mathbf{t}_{13}$	*	t_3	t_{16}	t_{26}
	22.99 ~ 27.61	27.61 ~ 31.74	31.74 ~ 37.49		
	t_4	t_{12}	t_{23}		
VM3	0 ~ 6.57	6.57 ~ 12.29	12.29 ~ 18	18 ~ 22.43	22.43 ~ 26.71
	$\mathbf{t}_2$	t_7	t_{20}	t_9	t_{10}
	26.71 ~ 33.29	33.29 ~ 39.43			
	t_{17}	t_{25}			
VM4	0 ~ 5.11	5.11 ~ 9.44	9.44 ~ 14.67	14.67 ~ 19.22	19.22 ~ 24.11
	$\mathbf{t}_2$	t_5	t_{22}	t_{18}	t_6
	24.11 ~ 27.07	27.07 ~ 31.51	31.51 ~ 36.96	36.96 ~ 41.18	41.18 ~ 45.83
	*	t_1	t_{19}	t_{27}	*
	45.83 ~ 51.39				
	t_0				
VM5	0 ~ 7.67	7.67 ~ 14.83	14.83 ~ 21	21 ~ 28.33	28.33 ~ 34.83
	$\mathbf{t}_2$	t_{11}	t_{24}	t_8	t_{21}
	34.83 ~ 41.83				
	t_{29}				

5.1 Experimental settings

The proposed algorithm is tested on four benchmark workflows in Fig. 2, namely Montage, CyberShake, LIGO and Epigenomics [54]. The parameters in this article and the range of values are shown in Table 8, which can be referred to [27]. The sizes of tasks, processing capacities of VMs and bandwidth between VMs are randomly generated by uniform distribution. The range of *CCR* indicates the transition of workflows from computation-intensive to data-intensive with

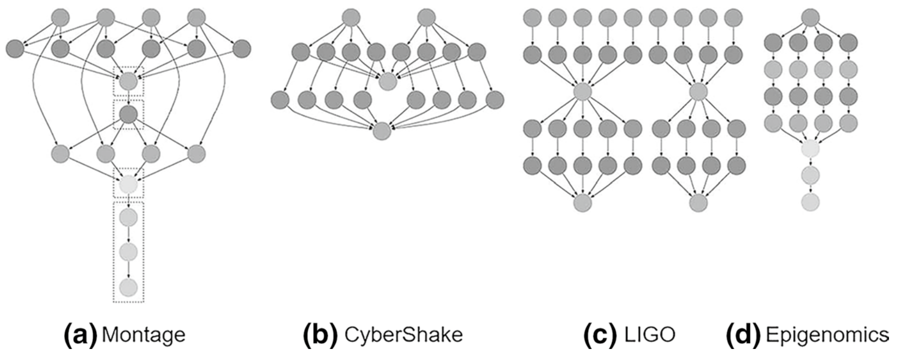

Fig. 2 Structures of the workflows in the experiments

Table 8 Parameters and range of values

Parameter	Range of value
Workflow type	Montage/Cyber-Shake/LIGO/Epigenomics
Scale of workflow	small/medium/large
Size of tasks	500 ~ 700 (MIP)
Number of VMs	5/10/50
Processing capacities of VMs	10 ~ 20 (MIPS)
Bandwidth between VMs	20 ~ 30 (Mbps)
CCR	0.4 ~ 2.0

the increase in *CCR*. Moreover, all algorithms are implemented and executed under Eclipse R2019 version 4.13.0 on an Intel core i7-9750H CPU @ 2.60GHz, 2.60GHz and 8GB RAM on Microsoft Windows 10 Professional 64-bit operating system.

5.2 Experimental results

The experimental results are discussed from two aspects, that is, measuring the performance of SM-CPTD and verifying the effectiveness of the proposed strategies. The detailed experimental results are shown in Figs. 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12. In order to reduce the impact of experimental uncertainties, each algorithm is run 10 times on each instance to obtain the average results.

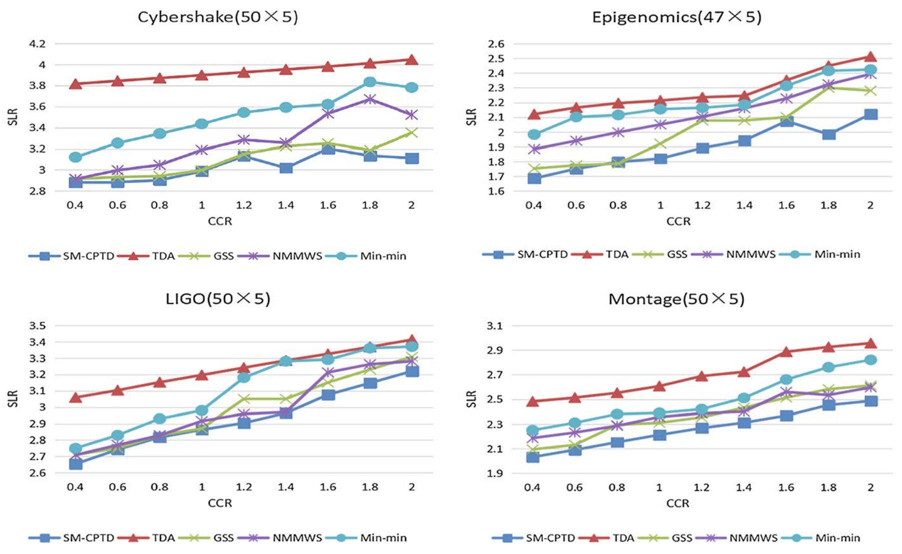

Fig. 3 Comparison of *SLR* in CCRs and small-scale workflows

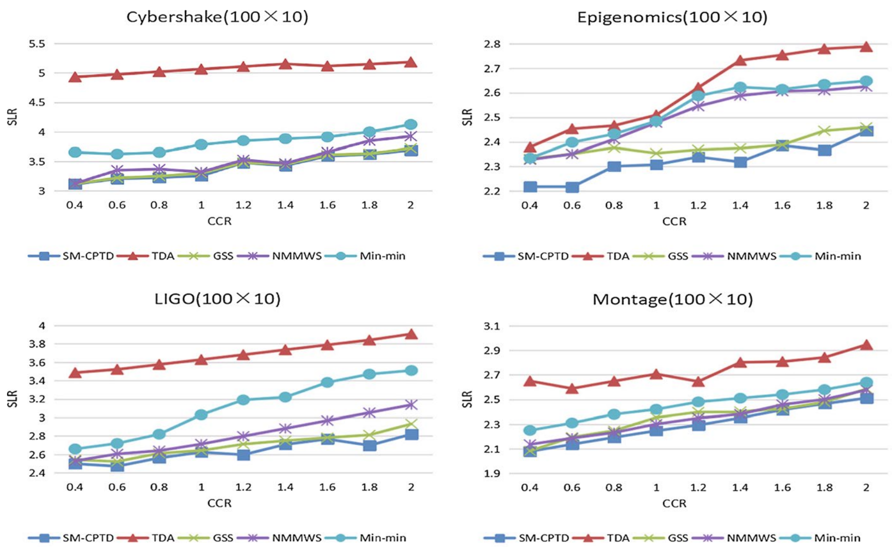

Fig. 4 Comparison of *SLR* in CCRs and medium-scale workflows

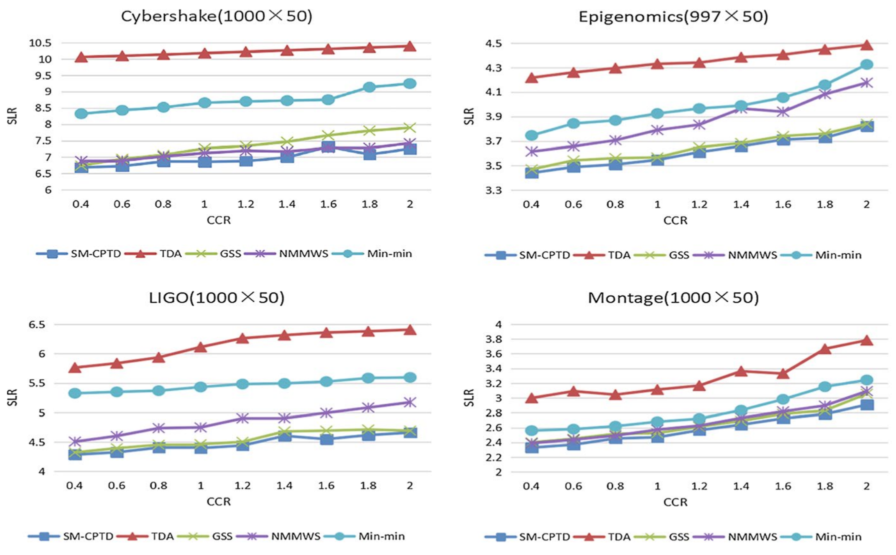

Fig. 5 Comparison of *SLR* in CCRs and large-scale workflows

5.2.1 Verification of SM-CPTD

To comprehensively evaluate the performance of SM-CPTD, TDA [26], GSS [27], NMMWS [28] and min–min [29] are selected as the comparative algorithms. In addition, the time complexity of TDA, GSS, NMMWS and min–min is $O(n^3)$, $O(n^2m)$, $O(n^3ml)$ and $O(n^2m)$, according to Geng et al. [26], Kumar et al. [27], Gupta

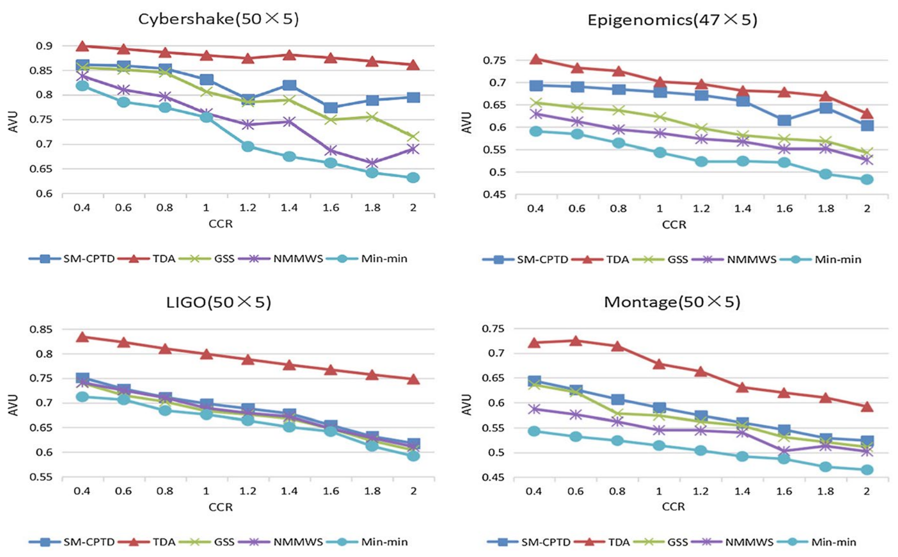

Fig. 6 Comparison of *AVU* in CCRs and small-scale workflows

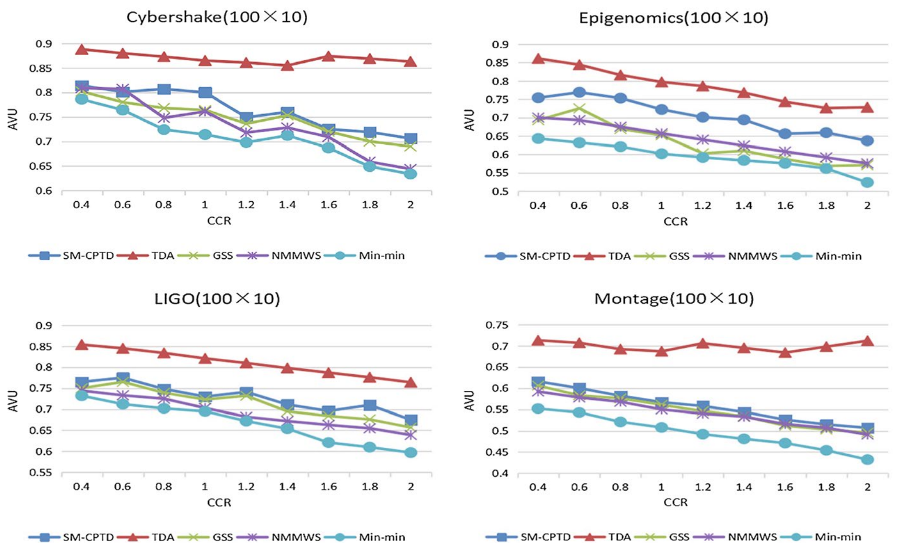

Fig. 7 Comparison of *AVU* in CCRs and medium-scale workflows

et al. [28] and Maheswaran et al. [29], respectively. Clearly, our proposed SM-CPTD algorithm (with a time complexity of $O(n^2l)$) has the similar time complexity as min–min which is generally regarded as one of the most efficient scheduling algorithms for benchmarking purpose.

To investigate how the performance of algorithms impacted by CCR and m, the numbers of VMs for small, medium and large workflows are set to 5, 10 and 50,

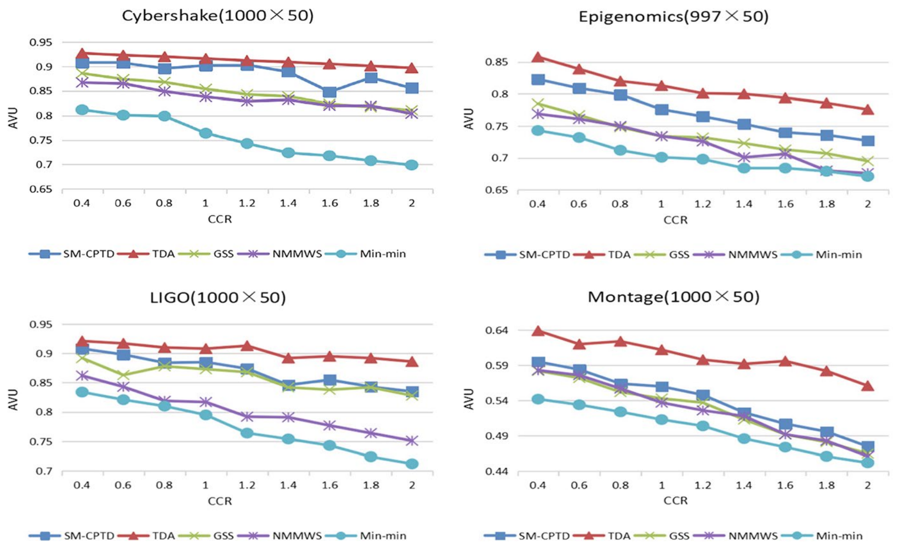

Fig. 8 Comparison of *AVU* in CCRs and large-scale workflows

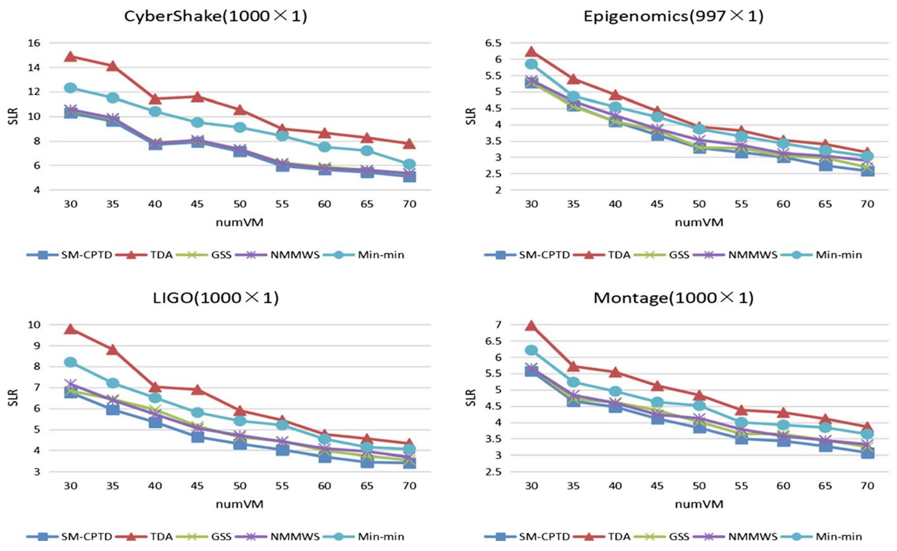

Fig. 9 Comparison of *SLR* in large-scale workflows and different numbers of VMs

respectively, when the influence of CCR on each algorithm is considered. Similarly, when considering the impact of *m*, the value of CCR is set to 1. Meanwhile, to study the *VF* of different algorithms in different workflows, we set *m* as 50 and CCR as 1. The evaluation results are shown in Figs. 3, 4, 5, 6, 7, 8, 9, 10 and 11.

It can be apparently observed from Figs. 3, 4 and 5 that the values of *SLR* obtained by all algorithms increment as the values of CCR increase from 0.4 to 2,

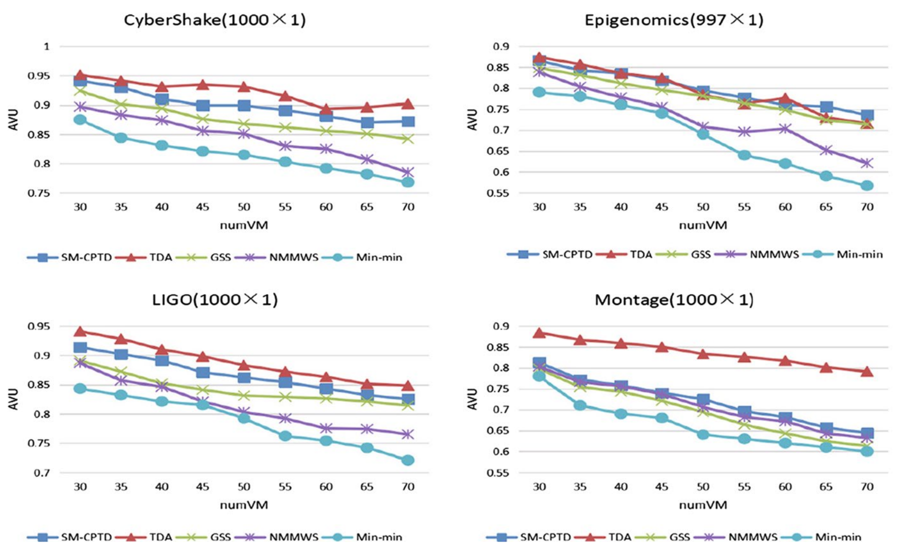

Fig. 10 Comparison of *AVU* in large-scale workflows and different numbers of VMs

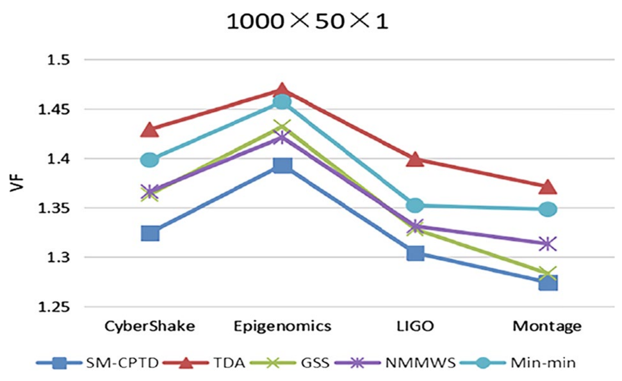

Fig. 11 Comparison of *VF* in different large-scale workflows

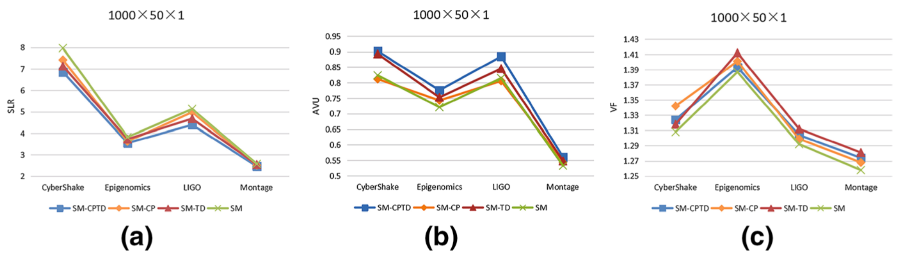

Fig. 12 Comparison of *SLR*, *AVU* and *VF* for different methods

which prolongs the transmission time of data between tasks. As a result, the completion time of workflows increases. However, SM-CPTD shows the average improvement of 20.93% over TDA, 2.66% over GSS, 5.25% over NMMWS and 11.55% over min–min in terms of *SLR*, respectively. That is, SM-CPTD obtains the minimum *SLR*, corresponding to the minimum makespan, on instances of different scales of workflows among all compared algorithms. Moreover, the performance of GSS is comparable with that of NMMWS, and both they and min–min beat that of TDA in most cases.

In TDA, a large number of tasks are replicated to decrease the transmission time between tasks, which can lead to data redundancy and extra execution time of tasks. So it performs better in data-intensive workflows than in computation-intensive workflows. The performance of NMMWS depends on the sizes of tasks and the computing capability of VMs. Specifically, NMMWS under-performs in the case of small-sized tasks because NMMWS is hard to obtain fine batching results.

From Figs. 6, 7 and 8, it can be seen that the values of *AVU* of all algorithms decrease with the increase in CCR values, which causes that the VMs have to wait for longer time to transmit data. Moreover, distinct from the results in *SLR*, TDA obtains the best results in terms of *AVU*. It is because that lots of replicated tasks processed in TDA can be filled into the idle period of VMs, which benefits the utilization of VMs. Although 9.42% worse than those of TDA, the *AVU* values of SM-CPTD are 4.07, 6.56, 12.89% better than those of GSS, NMMWS and min–min, respectively.

It can be found from Figs. 9 and 10 that for large-scale workflows, SM-CPTD is able to obtain the minimum value of *SLR* regardless of the number of VMs. Moreover, with the increase in numbers of VMs, the parallel computing capability of VMs is elevated to decrease the completion time of workflows, while the utilization of VMs decreases. Furthermore, although approximately 5.26% worse than TDA in terms of *AVU*, SM-CPTD obtains 2.98, 6.12 and 11.69% better utilization than GSS, NMMWS and min–min on instance groups with different numbers of VMs, respectively.

The comparison of *VF* in different large-scale workflows is shown in Fig. 11. The comparison results show that the proposed algorithm has better *VF* in all workflows compared with other algorithms.

5.2.2 Verification of the proposed methods

To verify the two local optimization methods presented in this paper, three other algorithms are also adopted as the comparative algorithms, namely SM, SM-CP and SM-TD. SM is the original stable matching algorithm without any local optimization method. SM-CP is the SM with local optimization method of critical path and SM-TD is the SM with local optimization method of task duplication. Without loss of generality, the algorithms are tested on large-scale workflows, we set *m* to 50 and CCR to 1. The evaluation results are shown in Fig. 12. It can be observed from Fig. 12a that both local optimization methods can reduce the makespan of workflow, and SM-TD is superior to SM-CP. In addition, the performance of the algorithm can be improved by 11.28% when the two methods are both applied. From Fig. 12b, it can be found that since *AVU* is related to the makespan of workflow, as the makespan decreases, the value of *AVU* increased as well. Therefore, compared with SM, SM-CP obtains better results in terms of *AVU*. Moreover, SM-TD utilizes the idle time of VMs, so that the results of AVU obtained by SM-TD are higher than those obtained by SM-CP. When the two local optimization methods are both adopted, SM-CPTD achieves the best utilization with an increase of about 7.69%. It can be seen from Fig. 12c that SM obtains the smallest *VF* value among all workflows. It means that the original stable matching algorithm can effectively balance the fairness of each task. However, the fairness of some tasks is affected by the two local optimization methods. As a result, the value of *VF* of the other three algorithms is larger than that of SM.

In conclusion, our proposed algorithm SM-CPTD shows better performance than all the other algorithms. Moreover, the two local optimization methods, DCP and LOTD, are able to reduce the makespan of workflow effectively. In addition, when the value of CCR is set to one, the ranges of the average running time of SM-CPTD for small-scale, medium-scale and large-scale workflows with four different structures are 10–20, 30–40, 1200–1400 ms, respectively. Therefore, SM-CPTD is highly efficient and can be applied in online workflow scheduling scenarios.

6 Conclusions

Considering the requirement of task fairness in real-world workflow applications, a novel scheduling algorithm based on stable matching game theory, called SM-CPTD, is proposed for minimizing the workflow makespan and maximizing the task fairness. In addition, two local optimization methods based on critical path and task duplication, respectively, are presented and incorporated into SM-CPTD to reduce the completion time of critical tasks and take full advantage of the free VMs. The experimental results in terms of *SLR*, *AVU* and *VF* demonstrated that the proposed algorithm can effectively reduce the makespan of workflow, obtain better utilization of VMs and ensure the fairness of the tasks, simultaneously.

In the future, multiple objectives such as cost and energy consumption can be taken into consideration. Meanwhile, it is of great interest to investigate the workflow scheduling in the mobile edge computing environment where computation tasks are required to be offloaded to either the edge server or the cloud server before they are processed.

Acknowledgements This work is supported by the National Natural Science Foundation under Grants 71971002 and 61872002, the Humanity and Social Science Youth Foundation of Ministry of Education of China under Grant 15YJC630041 and the Natural Science Foundation of Anhui Provincial Department of Education under Grant KJ2015A062.

References

1. Mukherjee D, Nandy S, Mohan S, Al-Otaibi YD, Alnumay WS (2021) Sustainable task scheduling strategy in cloudlets. Sustain Comput: Inform Syst 30:100513

2. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Fut Gener Comput Syst 25(6):599–616
3. Wu Z, Liu X, Ni Z, Yuan D, Yang Y (2013) A market-oriented hierarchical scheduling strategy in cloud workflow systems. J Supercomput 63(1):256–293
4. Deelman E, Gannon D, Shields M, Taylor I (2009) Workflows and e-science: an overview of workflow system features and capabilities. Fut Gener Comput Syst 25(5):528–540
5. Liu X, Chen J, Liu K, Yang Y (2008) Forecasting duration intervals of scientific workflow activities based on time-series patterns. In: 2008 IEEE 4th International Conference on eScience. IEEE, pp 23–30
6. Darbha S, Agrawal DP (1998) Optimal scheduling algorithm for distributed-memory machines. IEEE Trans Parallel Distrib Syst 9(1):87–95
7. Xie Y, Zhu Y, Wang Y, Cheng Y, Xu R, Sani AS, Yuan D, Yang Y (2019) A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. Fut Gener Comput Syst 97:361–378
8. Huang B, Li Z, Tang P, Wang S, Zhao J, Hu H, Li W, Chang V (2019) Security modeling and efficient computation offloading for service workflow in mobile edge computing. Fut Gener Comput Syst 97:755–774
9. Shih CS, Wei JW, Hung SH, Chen J, Chang N (2013) Fairness scheduler for virtual machines on heterogonous multi-core platforms. ACM Sigapp Appl Comput Rev 13(1):28–40
10. Rezaeian A, Naghibzadeh M, Epema DHJ (2019) Fair multiple-workflow scheduling with different quality-of-service goals. J Supercomput 75(2):746–769
11. Jang J, Jung J, Hong J (2019) K-LZF: an efficient and fair scheduling for edge computing servers. Fut Gener Comput Syst 98:44–53
12. Sethuraman J, Teo CP, Qian L (2006) Many-to-one stable matching: geometry and fairness. Math Oper Res 31(3):581–596
13. Zhang Y, Cui L, Zhang Y (2017) A stable matching based elephant flow scheduling algorithm in data center networks. Comput Netw 120:186–197
14. Topcuoglu H, Hariri S, My Wu (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
15. Xian-Fu M, Wei-Wei L (2010) A dag scheduling algorithm based on selected duplication of precedent tasks. J Comput-Aided Des Comput Graph 22(6):1056–1062
16. Geng X, Xu G, Fu X, Zhang Y (2012) A task scheduling algorithm for multi-core-cluster systems. JCP 7(11):2797–2804
17. Chen W, Xie G, Li R, Bai Y, Fan C, Li K (2017) Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. Fut Gener Comput Syst 74:1–11
18. Samadi Y, Zbakh M, Tadonki C (2018) E-heft: enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing. In: 2018 International Conference on High Performance Computing and Simulation (HPCS). IEEE, pp 601–609
19. Tian-mei zi C, Heng-zhou Y, Zhi-dan H (2018) K-heft: a static task scheduling algorithm in clouds. In: Proceedings of the 3rd International Conference on Intelligent Information Processing, pp 152–159
20. Sahni J, Vidyarthi DP (2015) A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput 6(1):2–18
21. Zheng W, Qin Y, Bugingo E, Zhang D, Chen J (2018) Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds. Fut Gener Comput Syst 82:244–255
22. Wu T, Gu H, Zhou J, Wei T, Liu X, Chen M (2018) Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud. J Syst Arch 84:12–27
23. Ijaz S, Munir EU (2019) Mopt: list-based heuristic for scheduling workflows in cloud environment. J Supercomput 75(7):3740–3768
24. Zhang H, Zheng X, Xia Y, Li M (2019) Workflow scheduling in the cloud with weighted upward-rank priority scheme using random walk and uniform spare budget splitting. IEEE Access 7:60359–60375
25. Djigal H, Feng J, Lu J, Ge J (2020) IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 32(5):1057–1071
26. Geng X, Mao Y, Xiong M, Liu Y (2019) An improved task scheduling algorithm for scientific workflow in cloud computing environment. Clust Comput 22(3):7539–7548

27. Kumar MS, Gupta I, Panda SK, Jana PK (2017) Granularity-based workflow scheduling algorithm for cloud computing. J Supercomput 73(12):5440–5464
28. Gupta I, Kumar MS, Jana PK (2018) Efficient workflow scheduling algorithm for cloud computing system: a dynamic priority-based approach. Arab J Sci Eng 43(12):7945–7960
29. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF (1999) Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 59(2):107–131
30. Elsherbiny S, Eldaydamony E, Alrahmawy M, Reyad AE (2018) An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment. Egypt Inform J 19(1):33–55
31. Wu Z, Ni Z, Gu L, Liu X (2010) A revised discrete particle swarm optimization for cloud workflow scheduling. In: 2010 International Conference on Computational Intelligence and Security, IEEE, pp 184–188
32. Kaur M, Kadam S (2018) A novel multi-objective bacteria foraging optimization algorithm (MOBFOA) for multi-objective scheduling. Appl Soft Comput 66:183–195
33. Hu H, Li Z, Hu H, Chen J, Ge J, Li C, Chang V (2018) Multi-objective scheduling for scientific workflow in multicloud environment. J Netw Comput Appl 114:108–122
34. Huang CL, Jiang YZ, Yin Y, Yeh WC, Chung VYY, Lai CM (2018) Multi objective scheduling in cloud computing using Mosso. In: 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp 1–8
35. Ding R, Li X, Liu X, Xu J (2018) A cost-effective time-constrained multi-workflow scheduling strategy in fog computing. In: International Conference on Service-Oriented Computing. Springer, pp 194–207
36. Alsmady A, Al-Khraishi T, Mardini W, Alazzam H, Khamayseh Y (2019) Workflow scheduling in cloud computing using memetic algorithm. In: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT). IEEE, pp 302–306
37. Yang J, Jiang B, Lv Z, Choo KKR (2020) A task scheduling algorithm considering game theory designed for energy management in cloud computing. Fut Gener Comput Syst 105:985–992
38. Gao Z, Wang Y, Gao Y, Ren X (2018) Multi-objective non-cooperative game model for cost-based task scheduling in computational grid. arXiv preprint arXiv:1807.05506
39. Wang Y, Jiang J, Xia Y, Wu Q, Luo X, Zhu Q (2018) A multi-stage dynamic game-theoretic approach for multi-workflow scheduling on heterogeneous virtual machines from multiple infrastructure-as-a-service clouds. In: International Conference on Services Computing. Springer, pp 137–152
40. Sujana JAJ, Revathi T, Karthiga G, Raj RV (2015). Game multi objective scheduling algorithm for scientific workflows in cloud computing. In: 2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]. IEEE, pp 1–6
41. Zhang M, Li H, Liu L, Buyya R (2018) An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in clouds. Distrib Parallel Databases 36(2):339–368
42. Chen L, Li X, Ruiz R (2018) Idle block based methods for cloud workflow scheduling with preemptive and non-preemptive tasks. Fut Gener Comput Syst 89:659–669
43. Shishido HY, Estrella JC, Toledo CFM, Arantes MS (2018) Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds. Comput Electr Eng 69:378–394
44. Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY (2018) GA-ETI: an enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. J Comput Sci 6:318–331
45. Saharan S, Somani G, Gupta G, Verma R, Gaur MS, Buyya R (2020) QuickDedup: Efficient VM deduplication in cloud computing environments. J Parallel Distrib Comput 139:18–31
46. Manasrah AM, Ba Ali H (2018) Workflow scheduling using hybrid GA-PSO algorithm in cloud computing. Wirel Commun Mob Comput 2018
47. Li W, Xia Y, Zhou M, Sun X, Zhu Q (2018) Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. IEEE Access 6:61488–61502
48. Ismayilov G, Topcuoglu HR (2018) Dynamic multi-objective workflow scheduling for cloud computing based on evolutionary algorithms. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC companion). IEEE, pp 103–108
49. Adhikari M, Koley S (2018) Cloud computing: a multi-workflow scheduling algorithm with dynamic reusability. Arab J Sci Eng 43(2):645–660
50. Kumar MS, Gupta I, Jana PK (2017) Delay-based workflow scheduling for cost optimization in heterogeneous cloud system. In: 2017 10th International Conference on Contemporary Computing (IC3). IEEE, pp 1–6

51. Choudhary A, Gupta I, Singh V, Jana PK (2018) A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing. Fut Gener Comput Syst 83:14–26
52. Luo F, Yuan Y, Ding W, Lu H (2018) An improved particle swarm optimization algorithm based on adaptive weight for task scheduling in cloud computing. In: Proceedings of the 2nd International Conference on Computer Science and Application Engineering, pp 1–5
53. Mohanapriya N, Kousalya G, Balakrishnan P, Pethuru Raj C (2018) Energy efficient workflow scheduling with virtual machine consolidation for green cloud computing. J Intell Fuzzy Syst 34(3):1561–1572
54. Center SC (2014). Cybershake and epigenomics scientific workflow. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator