Information Sciences 531 (2020) 31–46

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments

Longxin Zhang[a], Liqian Zhou[a,*], Ahmad Salah[b]

[a] College of Computer Science, Hunan University of Technology, Zhuzhou, 412007 China
[b] Faculty of Computers and Informatics, Zagazig University, 1 El-zera Square, Zagazig, Sharkia, 44519, Egypt

ARTICLE INFO

Article history:
Received 16 November 2019
Revised 22 March 2020
Accepted 15 April 2020
Available online 11 May 2020

Keywords:
Cloud computing
Deadline
Directed acyclic graph (DAG)
Makespan
Resource management

ABSTRACT

Data centers for cloud computing must accommodate numerous parallel task executions simultaneously. Therefore, data centers have many virtual machines (VMs). Minimizing the scheduling length of parallel task sets becomes a critical requirement in cloud computing systems. In this study, we propose an efficient priority and relative distance (EPRD) algorithm to minimize the task scheduling length for precedence constrained workflow applications without violating the end-to-end deadline constraint. This algorithm consists of two processes. First, a task priority queue is established. Then, a VM is mapped for a task in accordance with its relative distance. The proposed method can effectively improve VM utilization and scheduling performance. Extensive rigorous experiments based on randomly generated and real-world workflow applications demonstrate that the resource reduction rate and scheduling length of the EPRD algorithm significantly surpass those of existing algorithms.

1. Introduction

In recent years, the rapid development and wide application of cloud computing have changed people's daily work and life. For instance, software as a service is widely used in the data centers of Google, Twitter, Facebook, Alibaba, and other major IT companies. The pay-as-you-go business model and service-oriented paradigm in cloud computing allow users to customize resources and services freely in accordance with their needs. Amazon's cloud computing revenue reached USD 12.2 billion in 2016. Cloud services are one of the fastest-growing areas in computing; companies are adopting cloud services at a fast rate, and the global market is expected to exceed USD 270 billion by 2020 [1]. Scalable infrastructure and processing engine technologies that support cloud services, such as the Google File System [2], the MapReduce [3] programming model, the distributed File System Hadoop [4] developed by Apache Foundation, and the in-memory computing framework Spark [5] developed by the AMP Lab of University of California, Berkeley, have largely affected how application services run.

The basic feature of cloud computing is that computing and storage tasks scattered across many personal computers are handled using centralized data centers. The resource allocation problem in cloud computing is proven to be NP-complete [6]. In general, the optimal algorithm can be used to obtain the optimal solution, and approximation or heuristic algorithms are

* Corresponding author.
E-mail addresses: longxinzhang@hnt.edu.cn (L. Zhang), zhouliqian@hut.edu.cn (L. Zhou), ahmad@zu.edu.eg (A. Salah).

https://doi.org/10.1016/j.ins.2020.04.039
0020-0255/

designed to obtain a feasible solution. Applications with deadline constraints, such as media streaming applications, smart city traffic forecasts, and online banking systems, maximize cloud computing to obtain results. Some applications, such as the national weather center's hurricane forecasts, can result in disastrous consequences if deadlines are missed.

Resource management in cloud computing has elicited varied research attention. Many existing studies have achieved good results. High-performance computing has been playing an increasingly important role in the information field. In general, several virtual resources are managed by cloud computing. Arunarani et al. [7] comprehensively surveyed task scheduling strategies and performance metrics in cloud computing systems. Many parallel tasks should be executed on many-core processors in data centers on a daily basis. Thus, some important task scheduling problems, such as system performance optimization and energy saving, should be considered. Chen et al. [8] proposed optimized SpGEMM kernels to improve large-scale applications in supercomputing centers. By utilizing the high computational power of GPUs to accelerate in-memory cluster computing, Chen et al. [9] designed an easy programming model in heterogeneous CPU-GPU computing systems. Li [10] addressed the parallel task scheduling problem with energy and time constraints and gave the lower bounds of optimal solutions. Maqsood et al. [11] studied task scheduling and mapping strategy to minimize energy consumption by considering the communication overhead in computer systems. A bi-layered parallel training architecture was designed by Chen et al. [12] to accelerate the training process of large-scale convolutional neural networks. The knapsack-based bin-packing algorithm was proposed to obtain joint optimization in two dimensions, namely, energy consumption and network load. Wang et al. [13] developed parallelism-aware scheduling strategies based on existing task scheduling algorithms to realize the task scheduling problem in polynomial time. The combination problem is transformed into a binary programming problem. On the basis of reinforcement learning, Islam et al. [14] designed a framework under several task scheduling strategies in multiple core systems. Chen et al. [15] developed a periodicity-based parallel time series prediction to handle the massive historical datasets in distributed systems. In their framework, voltage and frequency can be scaled intelligently to match the features of tasks and processors. A profile-based efficient power-aware framework was presented by Qureshi [16] to achieve a good tradeoff among the cost of virtual machines (VMs), CPU utilization, load balance, and power usage in data centers. Several characteristics of application workflow requirements that involve CPU, memory size, network bandwidth, and power budget constraints are considered while assigning application workloads to the cloud center.

In certain cloud systems, such as Hadoop, the mapping task jobs perform with low efficiency due to the input data that are in a remote memory system instead of local storage. Selvitopi et al. [17] developed a scheduling model to balance between load balance and data locality in the map and reduce phases. Huang et al. [18] proposed a novel algorithm to solve the data skew problem in the MapReduce computing framework. Chen et al. [19] extended Flink to heterogeneous CPU-GPU clusters to achieve high-performance dataflow processing. A two-stage task scheduling algorithm based on the historical task scheduling log information was presented by Zhang and Zhou [20] to meet deadline and service quality constraints. Chen et al. [21] proposed a security-aware scheduling algorithm to reduce the makespan and monetary costs of workflow application while satisfying the security requirements in cloud computing by taking advantage of the security-sensitive intermediate data in the workflow. Although energy saving and system reliability enhancement are crucial metrics in most cases in a distributed computing system, they are also conflicting objects. A bi-objective genetic algorithm was developed in [22] to solve the parallel workflow task scheduling problem, the pursuit of which involves low energy consumption and high system reliability. Tong et al. [23] designed two heuristic algorithms, namely, biogeography-based optimization (BBO) and new hybrid migrating BBO, to address the parallel tasks with precedence constraints in cloud computing.

In previous studies, some excellent strategies have been proposed for parallel task sets under deadline constraints in cloud computing environments. However, the time complexity of these algorithms is high. In general, VM resources are essential in cloud computing. Improving VM resource utilization and simultaneously ensuring that task sets do not miss deadlines have considerable practical implications.

The contributions of this study are listed as follows:

- The resource management in cloud computing centers is formalized into a combinatorial optimization problem.
- An efficient parallel task scheduling algorithm is presented in cloud computing environments under end-to-end deadline constraints.
- Various workflow applications, including randomly generated and real-world workflow applications, and scientific evaluation methods are used to evaluate the performance of the proposed algorithm.

The remainder of the paper is organized as follows. Section 2 elaborates the related works of resource management in heterogeneous computing. Section 3 introduces some models used in this study and describes the problem to be resolved. Section 4 presents a novel heuristic algorithm to schedule a parallel task set with end-to-end deadline constraints to appropriate VM instances in cloud computing environments. Section 5 discusses the experiments conducted to assess the presented algorithm with state-of-art algorithms. Section 6 concludes the paper and mentions possible future works.

2. Related work

Xu et al. [24] presented a molecular genetic algorithm to minimize the makespan in heterogeneous computing systems. Ali et al. [25] proposed a contention-aware strategy to optimize computation and communication energy by applying an integrated method that includes dynamic voltage scaling and task mapping techniques for a real-time directed cyclic graph

(DAG) task set with precedence and deadline constraints. Zhang et al. [26] developed a novel contention-aware reliability scheme while satisfying deadline and energy constraints for parallel tasks in distributed systems.

In general, task scheduling is an NP-hard problem. Topcuoglu et al. [27] presented an efficient heuristic algorithm heterogeneous-earliest finish-time (HEFT) and mapped a parallel task set to a matched processor in heterogeneous computing systems. Mezmaz et al. [28] investigated parallel applications with precedence constraints and proposed a novel bi-objective genetic algorithm that considers makespan and energy consumption on cloud computing environments. The competing user loads, performance variations, failures, and other factors have an important impact on deadline-sensitive parallel applications. A workflow orchestrator for distributed systems architecture was designed in [29] on the basis of a least common denominator resource model. Durillo et al. [30] developed a bi-objective optimization by considering performance and energy saving. This approach is based on empirical models that depict real-world operations in heterogeneous computing environments. To attain the high-quality security needed in security-sensitive applications, such as bank systems, Tang et al. [31] developed a security-driven scheduling architecture to guarantee high quality of security and high performance while processing such parallel applications.

Considering the important influence of bandwidth constraint, security network latency, and other key economic aspects on the cloud model, a set of cost-efficiency algorithms has been proposed for deadline-constrained parallel task applications on public and private cloud centers [32]. Xiao et al. [33] developed a CASpMV framework for a supercomputer center to promote the parallel efficiency of SpMV in heterogeneous computing systems. The parallel K-Tree was proposed in [34] to obtain a multi-core solution of extreme clustering in high-performance applications. On the basis of a dynamic voltage and frequency scaling technique, the whale optimization algorithm (WOA), which is a joint optimal task scheduling scheme, was presented in [35] to achieve the tradeoff of task set complete time and energy consumption in mobile cloud computing. During WOA processing, three important factors, namely, the sequence of task execution, the position of task execution, and the operation voltage of mobile devices, are considered. Wu et al. [36] presented the minimal slack time and minimal distance (MSMD) algorithm to achieve a minimal task complete time for task applications with deadline constraints in cloud computing.

3. Models

In this section, application, system, and task models and some basic concepts are described.

3.1. Application model

A parallel application with end-to-end constraints can be modeled as a DAG. In the DAG, $G =< T, E >$, where T is the parallel task set and E is the communication edge among task nodes in the parallel tasks. An edge, $e_{ij} \in E$, represents the priority constraint between task nodes τ_i and τ_j. The parent node τ_i should be executed before its child node τ_j can be released. Weight w_i on task node τ_i denotes the execution time of that on VM. When VMs are equipped with a uniform configuration, the execution time of the task τ_i on VM vm_j will be the same. In each DAG, all the direct predecessors of task node τ_i are denoted as $parent(\tau_i)$, that is, $parent(\tau_i) = \{\forall\tau_p | e_{p,i} \in E\}$. All the direct successors of task node τ_i are denoted as $child(\tau_i)$, that is, $child(\tau_i) = \{\forall\tau_c | e_{i,c} \in E\}$. Two special task nodes exist in a DAG. The node without a parent node is called an entry node, that is, τ_{entry}. The node without a child node is called an exit node, that is, τ_{exit}. In our study, each DAG has only one entry node and one exit node. If more than one entry or exit node exists, then a node with zero computation cost can be added as a virtual entry or exit node. As shown in Fig. 1, this DAG has 11 task nodes, where τ_0 is τ_{entry} and τ_{10} is τ_{exit}.

The speed of a network is much faster than that of a hard disk in the cloud computing center. Thus, the communication cost between two adjacent nodes is negligible in this model. The edge that connects two adjacent nodes only represents the priority constraint. When each parallel task set as a DAG application arrives at the cloud computing center, release time T_R is required to prepare the entire task to be executed. Meanwhile, the application should guarantee the end-to-end deadline constraint T_D.

3.2. System model

The cloud computing center in this study can be modeled as a set of VM instances. A VM instance set is denoted as $VM = \{vm_0, vm_1, vm_2, \ldots, vm_n\}$. All the instances have different types. They are heterogeneous with different configurations. In other words, CPU cores, DRAM capacity, and hard memory are equipped with diverse configurations. Similar to most cloud computing centers, the file system used in this study is built with a shared storage design. The cost of data communication among VM instances is negligible. The bandwidth of a VM can be considered ideal. No communication contention occurs among VM instances. The notations about the application and system models used in this study are shown in Table 1. The computation cost of the tasks in Fig. 1 on different VMs is shown in Table 2.

Table 1
Notations used in this study.

Notation	Definition
DAG	Directed Acyclic Graph
G	A parallel tasks set
VM	A virtual machine instances set
T_C	The computation costs of tasks in critical path of the DAG
T_D	The end-to-end deadline of DAG application
τ_i	The task node i
vm_k	The virtual machine instance k
$w_{i,j}$	The computation cost of task node τ_i on vm_j
$t_{dr}(\tau_i, vm_j)$	The data ready time of task node τ_i on VM instance vm_j
makespan	The task completion time of the exit node in the DAG
$DRank(\tau_i)$	The downward rank value of τ_i
$URank(\tau_i)$	The upward rank value of τ_i
$child(\tau_i)$	The set of immediate successors of task node τ_i
$parent(\tau_i)$	The set of immediate predecessors of task node τ_i
$EST(\tau_i)$	The earliest start time of task node τ_i
$EAT(vm_i)$	The earliest available time of vm_i
$LFT(\tau_i)$	The latest finish time of task node τ_i
$FT(\tau_i)$	The finish time of task node τ_i
$maxslack$	The maximum slack time of τ_i in the DAG applicaiton
$Res(\tau_i, vm_j)$	The relative distance of task node τ_i on vm_j
SLR	Scheduling length ratio
RRR	Resource reduction ratio
MRR	Makespan reduction ratio

Table 2
Computation cost of tasks on different VMs of the DAG application in Fig. 1.

Tasks	vm_0	vm_1	vm_2
τ_0	8	5	9
τ_1	12	8	13
τ_2	11	10	14
τ_3	16	15	17
τ_4	12	9	11
τ_5	10	12	15
τ_6	14	11	13
τ_7	11	16	12
τ_8	13	14	10
τ_9	5	10	8
τ_{10}	15	16	18

3.3. Preliminaries

3.3.1. Task earliest start time $\mathrm{EST}(\tau_i)$ and task latest finish time $\mathrm{LFT}(\tau_i)$

The earliest time of task $\tau_i \in T$ on VM vm_j denotes the latest finish time of all of the predecessors of task τ_i plus its execution time on VM. The earliest start time of the task τ_i can be calculated recursively as follows:

$$EST(\tau_i) = \begin{cases} T_R & if\ \tau_i = \tau_{entry}, \\ \max\limits_{\tau_k \in parent(\tau_i)} \{EST(\tau_k) + w_k\} & otherwise. \end{cases} \tag{1}$$

Similarly, the latest finish time of a given task $\tau_i \in T$ can be defined as the latest start time of τ_i's successors. The expression can be expressed as follows:

$$LFT(\tau_i) = \begin{cases} T_R + T_D & if\ \tau_i = \tau_{exit} \\ \min\limits_{\tau_k \in child(\tau_i)} \{LFT(\tau_k) - w_k\} & otherwise. \end{cases} \tag{2}$$

3.3.2. Earliest available time $\mathrm{EAT}(\mathrm{vm_n})$ of a VM instance

For the VM instance vm_n, the available time is 0 when no task is assigned to vm_n. In Eq. (3), if some tasks are scheduled on vm_n, that is, $i > 0$, then the earliest available time of vm_n depends on the data ready time of $t_{dr}(\tau_i, vm_n)$, the earliest finish time $EFT(\tau_{i-1,n})$ of τ_{i-1} on vm_n (where $EFT(\tau_{i-1,n}) = EST(\tau_{i-1,n}) + w_{i-1,n}$), and the computing cost $w_{i,n}$ of task τ_i on VM instance vm_n. The earliest available time of a VM instance is defined as follows:

$$\mathrm{EAT}(vm_n) = \begin{cases} 0 & i = 0, \\ \max\{EFT(\tau_{i-1,n}), t_{dr}(\tau_i, vm_n)\} + w_{i,n} & i > 0. \end{cases} \tag{3}$$

3.3.3. Task ready time (t_{dr})

The data ready time of task τ_i on VM instance vm_j denotes that all the data that are dependent on the task are received. Ready time t_{dr} can be defined as follows:

$$t_{dr}(\tau_i, vm_k) = \max_{\tau_j \in parent(\tau_i)} \left\{ FT\left(\tau_j, vm_k\right) \right\}, \tag{4}$$

where $FT(\tau_i, vm_j)$ denotes the finish time of task τ_i on vm_j.

3.3.4. Task rank

For a given DAG, the feasible priority queue should guarantee the topology of the DAG. The $URank(\tau_i)$ can be expressed as follows:

$$\mathrm{URank}(\tau_i) = \max_{\tau_j \in child(\tau_i)} \left\{ URank(\tau_j) \right\} + \bar{w}_i, \tag{5}$$

where $\bar{w}_i$ is the average computation cost of τ_i on different VMs. In particular, the *URank* of the exit node is equal to its computing cost on VM, namely, $URank(\tau_{exit}) = \bar{w}_{\tau_{exit}}$.

The downward rank *DRank* of a given task $\tau_i \in T$ can be computed recursively from the exit node of a DAG, as follows:

$$\mathrm{DRank}(\tau_i) = \max_{\tau_j \in parent(\tau_i)} \left\{ \bar{w}_j + DRank(\tau_i) \right\}. \tag{6}$$

In particular, the *DRank* of the entry node equals zero, namely, $DRank(\tau_{entry}) = 0$.

3.3.5. Task maximum slack time (maxslack)

The task maximum slack time *maxslack* is used to measure the length of the task's free time before causing a violation of the deadline constraint. When the *maxslack* of τ_i is equal to zero, that is, $maxslack = 0$, τ_i must be started at its earliest start time. Otherwise, a deadline violation will occur. In other words, a large *maxslack* indicates a considerable time it can afford to delay the start time of τ_i.

The definition of *maxslack* is expressed as follows:

$$maxslack(\tau_i) = LFT(\tau_i) - (EST(\tau_i) + \bar{w}(\tau_i)). \tag{7}$$

3.3.6. Task priority

The task priority in the DAG is established to maintain the precedence constraint of parallel tasks. On the basis of the *DRank* introduced above, tasks with low *DRank* values have higher priorities than tasks with high *DRank* values. When two tasks have the same *DRank* value, the task with the lower *maxslack* has a higher priority than the other task. If the *DRank* and *maxslack* values of the two tasks are the same, then the priority is assigned in accordance with the numbers of the tasks. On this basis, the task with the smaller number has a higher priority than the other task with the large number in this case.

3.3.7. Relative distance

The relative distance between the available time of the VM instance VM_j and the data ready time of task τ_i on the VM instance VM_j is used to indicate the close degrees of ready time of the candidate task. Candidate task τ_i in a priority queue is assigned to an appropriate VM instance following the rule of minimal relative distance among task τ_i on each VM instance, instead of $EFT(\tau_i)$ or $EST(\tau_i)$. The definition of relative distance is expressed as follows:

$$\mathrm{Res}(\tau_i, vm_j) = \begin{cases} EAT(vm_j) & if\ EAT(vm_j) < t_{dr}(\tau_i, vm_j), \\ t_{dr}(\tau_i, vm_j) & otherwise. \end{cases} \tag{8}$$

3.3.8. Virtual machine bounds

As proven in [36], the number of VM instances M should be used for a given parallel task with a deadline constraint T_D by complying with the following expression:

$$\left\lceil \frac{T_{seq}}{T_D} \right\rceil \le M \le |V| - Level(\tau_{exit}), \tag{9}$$

where T_{seq} is the sequential execution time, $|V|$ is the number of task nodes, and $level(\tau_{exit})$ is the topological level of the exit node τ_{exit}.

Table 3
Task priority of DAG application for Fig. 1.

Tasks	EST	LFT	*maxslack*	CP	*DRank*	Priority
τ_0	0	11.67	4.33	√	0.00	1
τ_1	8	23.67	5.33		7.33	3
τ_2	8	25.33	6.33		7.33	4
τ_3	8	27.67	4.33	√	7.33	2
τ_4	22	36.00	6.33		19.00	6
τ_5	16	36.00	5.33		18.33	5
τ_6	28	40.67	4.67		23.33	8
τ_7	24	40.67	4.33	√	23.33	7
τ_8	33	48.33	5.33		30.67	9
τ_9	39	48.33	4.33	√	36.33	10
τ_{10}	44	65.00	4.33	√	44.00	11

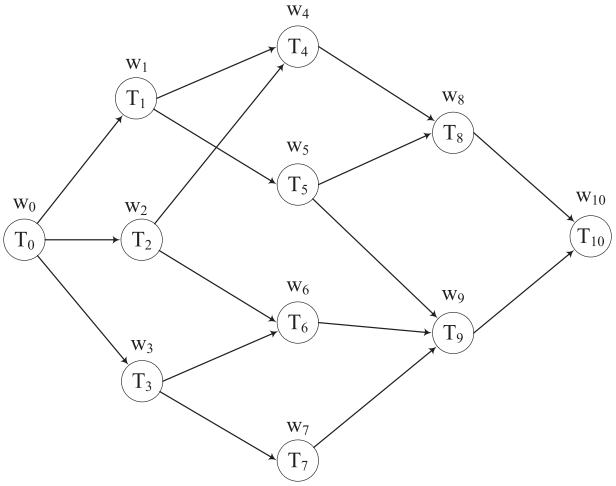

Fig. 1. Example of DAG with an end-to-end deadline constraint $T_D = 80$.

3.3.9. Problem description

On the basis of the above-mentioned descriptions, the problem to be solved in this study is to assign each task in the parallel task set with an end-to-end deadline constraint to an appropriate VM instance. In this manner, the total task set complete time is minimization. The formulation is given as follows:

$$\begin{aligned} &\text{Minimize}: \quad \mathrm{FT}(\tau_{\text{exit}}) \\ &\textit{Subject to}: \ FT(\tau_{exit}) \le T_D, \end{aligned} \tag{10}$$

$$\textit{and} \ \ \forall \tau_i \in \mathrm{T}, \sum_{j=1}^{n} M(\tau_i, vm_j) = 1. \tag{11}$$

where $FT(\tau_{exit})$ is the complete time of task τ_{exit}. Eq. (11) guarantees that each task can only be executed once on one VM.

3.3.10. Motivation example

On the basis of the aforementioned definitions, *EST, LFT, maxslack, DRank*, priority, and whether task τ_i is on the critical path of the DAG are shown in Table 3.

For the given DAG application, the critical path is composed of τ_0, τ_3, τ_7, τ_9, and τ_{10}. On the basis of the previous priority queue establishment criteria, the task queue sort is arranged in descending order in accordance with the task's *DRank* value. Tasks with the same *DRank* value are sorted in ascending order in accordance with *maxslack*. Therefore, the priority queue of DAG in Fig. 1 is $\{\tau_0, \tau_3, \tau_1, \tau_2, \tau_5, \tau_4, \tau_7, \tau_6, \tau_8, \tau_9, \tau_{10}\}$.

In the priority establishment stage, the MSMD algorithm based on the combination of the topological level and *maxslack* value is used. The topological level can be used to satisfy the precedence constraint of parallel tasks in DAG but cannot be utilized to build a priority queue. Furthermore, an improved relative space distance $Res(\tau_i, vm_j)$ is used when mapping the candidate task τ_i to the most suitable VM instance. At time 0, the three VMs are available, that is, $EST(vm_i) = 0, (i = 1, 2, 3)$. All task nodes in a critical path of DAG are assigned to VM instance vm_0. Using the MSMD algorithm, the schedule of the DAG in Fig. 1 is shown in Fig. 2(a). The makespan is 72. The priority of our algorithm is different from that of MSMD. Specifically, τ_4 is assigned to vm_2, instead of vm_0. The final schedule result is shown in Fig. 2(b). The makespan is 59, which is 13 units less than that of MSMD.

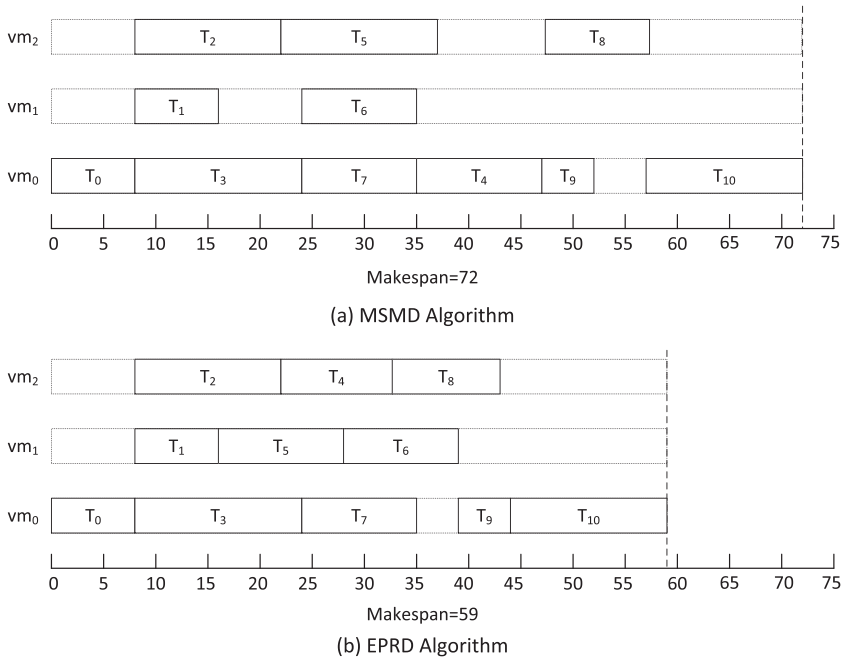

Fig. 2. Schedule result of two algorithms in accordance to task graph in Fig. 1.

4. Efficient priority and relative distance scheduling algorithm

In this section, the efficient priority and relative distance (EPRD) scheduling algorithm is presented for the problem given in Section 3.3.9.

Algorithm 1 provides an overview of the heuristic algorithm in this study. In Step 1 of Algorithm 1, the *URank* value

Algorithm 1: Schedule overview.

Input: A DAG $G =< T, VM >$, T_R, T_D.
Output: A schedule S of T on VMs while satisfying end-to-end deadline constraint.

1 compute *URank* of $\tau_i \in T$ by traversing the graph from the exit node;
2 compute *DRank* of $\tau_i \in T$ by traversing the graph from the entry node;
3 compute *EST*, *LFT*, *maxslack* for each node;
4 sort tasks in a non-decreasing order by *DRank* value for each task and build a priority queue $Queue_{URank}$;
5 n $\leftarrow \left\lceil \frac{T_{seq}}{T_D} \right\rceil$;
6 **while** $FT(\tau_i) \leq T_R + T_D$ **do**
7 $\quad vm[n] \leftarrow \{\emptyset\}$;
8 $\quad FT(\tau_{exit}) \leftarrow EPRD(T, n, Queue_{URank}, VM)$;
9 $\quad n \leftarrow n + 1$;
10 **end**

for each task node is calculated. In Step 2, the *DRank* value for each task node is computed. Steps 1–2 are followed to find the critical path of the DAG. The *EST, LFT*, and *maxclack* of each node are obtained in Step 3. The efficient priority queue of parallel tasks is established in Step 4 on the basis of the non-increasing order of *DRank* and ascending orders of *maxslack*. In Step 5, the number of VM instances is obtained with Eq. (9). Steps 6–10 are the loop body, the purpose of which is to map each task to a suitable VM instance. In Step 8, the EPRD algorithm is invoked.

The EPRD algorithm is presented in Algorithm 2. This algorithm aims to choose a feasible VM instance for each candidate task in the priority queue $Queue_{\text{URank}}$. The head task τ_i in the priority queue is selected in Step 2. If task τ_i belongs to the critical path of the DAG, then task τ_i is assigned to the VM instance vm_0 following an insert technique strategy. This process occurs in Step 4. All tasks in the critical path are scheduled to be executed on vm_0. The relative distance $Res(\tau_i, vm_0)$ on the vm_0 of the other task that does not exist in the critical path is calculated. $Res(\tau_i, vm_0)$ is stored temporarily to variable *resValue* in Step 6. The candidate VM instance is marked as vm_0 temporarily in Step 7. In the inner loop of Steps 8–12, the other VM instances are compared with vm_0 on the value of relative distance. The maximum relative distance and the

Algorithm 2: EPRD.

Input: A DAG $G =< T, VM >$, T_R, T_D, $Queue_{URank}$.
Output: $FT(\tau_{exit})$

```
1  while the priority queue Queue_URank is not empty do
2      τ_i ← the head node in Queue_URank;
3      if τ_i ∈ CP then
4          assign τ_i to vm_0;
5      else
6          resValue ← Res(τ_i, vm_0);
7          candidateVM ← vm_0;
8          for j ← 1 to m − 1 do
9              if Res(τ_i, vm_j) < resValue then
10                 Res(τ_i, vm_j) ← resValue;candidateVM ← j;
11             end
12         end
13         if candidateVM = 0 then
14             assign τ_i to vm_0 base on insert technique
15         end
16         assign τ_i to vm[candidateVM] based on insert technique
17     end
18 end
```

corresponding VM instance number are preserved. Following the previous operations, the candidate task τ_i is assigned to the corresponding VM instance in Steps 13–16.

We let N be the number of tasks in the workflow and M the number of VMs. In Algorithm 1, *URank* and *DRank* of the task set are calculated in Steps 1 and 2, respectively. They both have a time complexity of $O(|M| \times |N|)$. The time required in Step 4 is $O(|N| \times log_2|N|)$. In Algorithm 2, Steps 8–12 are the most time-consuming, they need $O(|M| \times |N|)$ time. Therefore, the time required for Steps 6–10 of Algorithm 1 is $O(|N|^2 \times |M|)$. Thus, the time complexity of Algorithm 1 is $O(|N|^2 \times |M|)$.

5. Experiments

A comprehensive evaluation and analysis of the proposed algorithm is conducted across a wide set of metrics, as presented in this section. Three state-of-the-art algorithms, namely, HEFT, MOHEFT, and MSMD, are introduced to compare with the proposed algorithm.

Topcuoglu et al. [27] proposed an efficient scheduling algorithm called HEFT. HEFT can achieve the objectives of high performance and low complexity in heterogeneous computing systems. The HEFT algorithm with a deadline constraint (denotes as HEFT_D) consists of two stages. In the first stage, the scheduling priority of tasks in the workflow is determined according to their *URank*. Then, a priority task scheduling queue is established with these priority tasks. In the second stage, the candidate task in the priority queue is mapped to the VM instance that has the earliest finish time.

MOHEFT [30] refers to the multi-objective HEFT. MOHEFT aims to predict the execution time and energy consumption of workflow and finding tradeoff solutions between makespan and energy consumption in a cloud center using historical data.

MSMD, which is a heuristic task scheduling algorithm in cloud computing centers, was presented by Wu et al. [36]. MSMD has two goals. One goal is to find the minimum amount of resources required while satisfying the end-to-end deadline of a given DAG application. The other goal is to minimize the scheduling length of DAG under specified VM instances.

5.1. Experimental setup

Randomly generated DAGs are used to evaluate the performance of our developed algorithm. Four real-word benchmarks are also used to test our presented algorithm with other start-of-art algorithms.

5.1.1. Randomly generated DAG

Without loss of generality, randomly generated DAG applications are exploited in the experiments. The random DAG applications are generated on the basis of the following parameters:

- DAG size: The number of task nodes in a DAG application. The size of the DAG ranges from 50 nodes to 300 nodes.
- Computing cost: The execution time of each task node in a DAG application. The computing cost of a task node follows a uniform distribution with a mean value of 15.
- Average in/out degree: The input/output edge of each task node. The average in/out degree varies from 2 to 10.
- Number of VM instances: The VM instances required in the task scheduling. The bound of VM instances complies with Eq. (9).

5.1.2. Real-world benchmark

Four types of popular applications, namely, CyberShake, Epigenoic (GENOME), Interferometer Gravitational Wave Observatory (LIGO), and Montage, which are provided by Pegasus Workflow Generator [37], are used in the following experiments.

The characteristics of these benchmarks are different from one another. The CyberShake is a parallel workflow application. Montage and GENOME applications combine the characteristics of sequential and parallel tasks. Other parallel applications include LIOG. However, some critical task nodes of LIOG even have a large number of child nodes. The detailed characteristics of the real-world benchmarks are elaborated in [38]. The number of task nodes in each application used in our experiments ranges from 50 nodes to 500 nodes. In accordance with the feasibility of task scheduling, the end-to-end deadline should be greater than its T_C because the four benchmarks do not supply their end-to-end deadlines T_D. Thus, $T_D > T_C$. Four different types of end-to-end deadlines, namely, $T_D = 1.5 \times T_C$, $T_D = 2.0 \times T_C$, $T_D = 2.5 \times T_C$, and $T_D = 3.0 \times T_C$, are chosen for the four applications to understand the effect of deadlines on algorithm performance.

The experiments are conducted on a Windows 10 workstation with an Intel Core i7-7700 quad-core CPU, 64 GB DRAM, and a 2 TB hard disk.

5.2. Performance metrics

5.2.1. Scheduling length ratio

The makespan, which is also called scheduling length, depends on the finish time of the exit task τ_{exit} in a DAG. The makespan is an essential metric when evaluating the performance of a cloud computing system. When running an algorithm for task scheduling, the number of VM instances considerably affects the makespan for a given DAG application because the makespan of a DAG application is usually high. The scheduling length ratio (SLR) metric is used to normalize the makespan. The SLR can be expressed as follows:

$$SLR = \frac{\text{makespan}}{\sum_{\tau_i \in CP} \min_{vm_j \in VM} \{w_{i,j}\}}, \tag{12}$$

where CP denotes the task nodes located in the critical path of the DAG.

5.2.2. Resource reduction ratio

The resource utilization is another important metric. The resource reduction ratio (RRR) is defined to indicate the extent of resource reduction from the upper bound of VM instances. The expression of RRR is defined as follows:

$$\text{RRR} = \frac{Upper\ Bound - Actual\ Resources\ Used}{Upper\ Bound}. \tag{13}$$

5.2.3. Makespan reduction ratio

The primary goal of EPRD is to minimize the scheduling length of DAG applications with appropriate VM instances under an end-to-end deadline constraint. An effective method to assess the makespan is time saving by comparing with the end-to-end deadline. The makespan reduction ratio (MRR) of a DAG application with a release time as T_R, and end-to-end deadline as T_D, is defined as follows:

$$MRR = \frac{T_R + T_D - SFT(\tau_{exit})}{T_D}. \tag{14}$$

5.3. Effect of random applications

The primary goal of these experiments is to evaluate the performance of EPRD and the three popular algorithms. Each data point in the experiment originates from the average value of SLR on the basis of algorithms running for 10 times. Figs. 3–6 show the average SLR of the four algorithms with $T_D = 1.5 \times T_C$, $T_D = 2.0 \times T_C$, $T_D = 2.5 \times T_C$, and $T_D = 3.0 \times T_C$. The four groups of algorithms use the same number of VM instances in the test of each random DAG application.

For all the comparisons of the four algorithms shown in Figs. 3–6, the average SLR increases as the number of tasks grows. Among the compared algorithms, the average SLR of MOHEFT is larger than that of the three other algorithms.

As illustrated in Fig. 3, the SLR value of algorithm MOHEFT is larger than that of the three other algorithms in the experimental comparison of several random graphs with a different number of nodes. The reason for this result is that the MOHEFT algorithm has two goals, namely, minimizing task set completion time and reducing energy consumption. When the same number of VMs is used, MOHEFT takes a slightly longer time than normal to complete the execution of the DAG applications.

When the number of nodes is 300, the average SLR of the EPRD algorithm decreases by 13.24%, 16.31%, and 6.35%, compared with that of HEFT_D, MOHEFT, and MSMD, respectively. With the increase in the deadline of applications, the SLR values of the four algorithms all rise. The increase in SLR for the four algorithms is insignificant because the other conditions are not changed. When the number of nodes in the DAG applications is below 300, the SLR values of the four algorithms are below 1.5. When the number of nodes reaches 300, the SLR values of these algorithms, except for EPRD, are greater than 1.5 under the condition of $T_D = 3.0 \times T_C$. EPRD performs well in terms of makespan reduction.

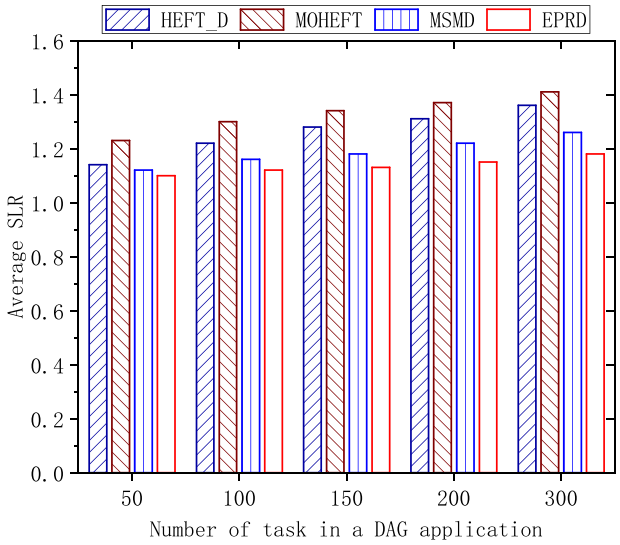

Fig. 3. Average SLR comparison for application with $T_D = 1.5 \times T_C$.

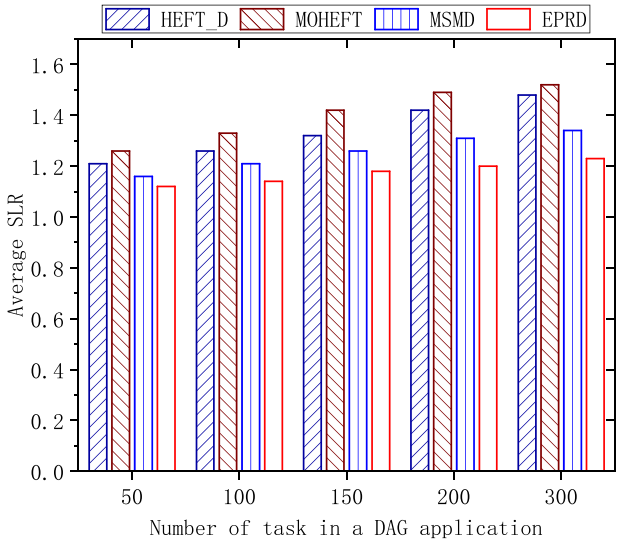

Fig. 4. Average SLR comparison for application with $T_D = 2.0 \times T_C$.

5.4. Experiment on real-world workflows

In addition to the random DAG, the performance of the algorithms is evaluated with real-world workflows, that is, Montage, Cybershake, Epigenomics, and Genome, as presented in this section. Large-scale workflows with low parallelism are common in the cases used in this section. Thus, some critical nodes in the DAG have large in/out degree values. The number of VMs used in the experiment is unlimited to the theoretical upper limit of Eq. (9) to ensure that all four algorithms can complete the execution of workflows before the specified deadline. In other words, each of the four algorithms may use a different number of VMs when testing the same workflow and meeting the same deadline constraint. Under such circumstances, the MRR cannot reflect the performance of the algorithm. Therefore, in the same test scenario, the RRR is added as the second metric. In addition, the data in the experiment are averaged by multiple running.

Figs. 7, 9, 11, and 13 show the comparison of average resource reduction rates of the four algorithms when the workflow's deadlines are 1.5, 2.0, 2.5, and 3.0 times its critical path length, respectively. Figs. 8, 10, 12, and 14 illustrate the comparison of the average MRRs of the four algorithms when the workflow's deadlines are 1.5, 2.0, 2.5, and 3.0 times its critical path length, respectively.

Figs. 7 and 8 show the comparison of the average RRR and MRR of the four algorithms when the end-to-end deadline of each DAG application is 1.5 times its critical path length. As illustrated in Fig. 7, a high average RRR value indicates a low

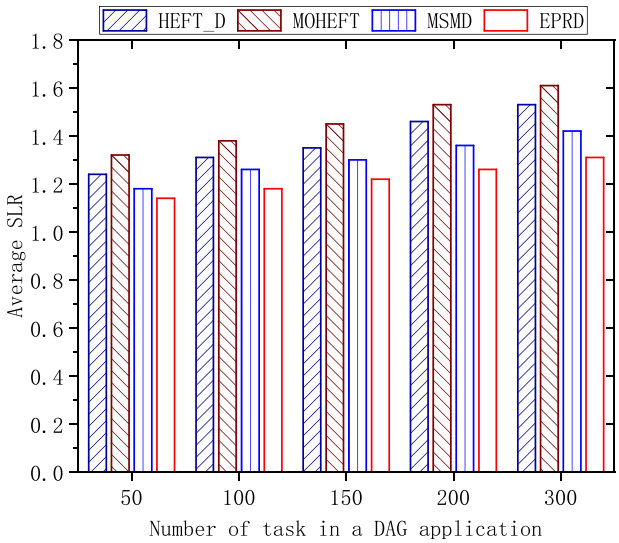

Fig. 5. Average SLR comparison for applications with $T_D = 2.5 \times T_C$.

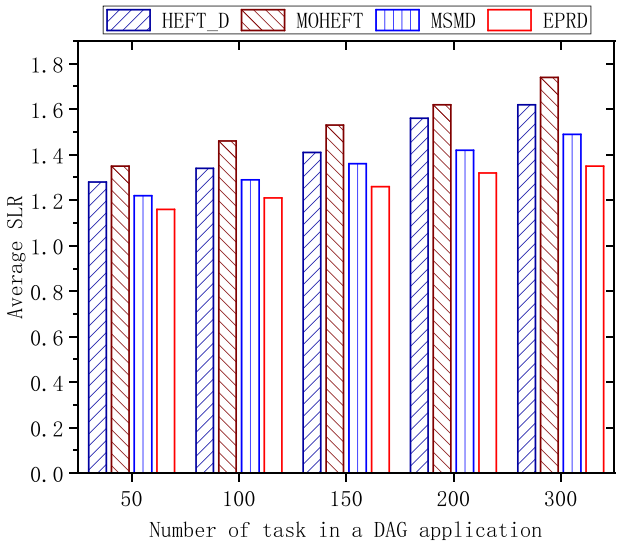

Fig. 6. Average SLR comparison for applications with $T_D = 3.0 \times T_C$.

number of VMs used in the test of the same workflow. In the CyberShake workflow, the average MRR of the four algorithms is relatively smaller than those of the three other workflows due to the high output or input degree of some nodes. In Fig. 7, MOHEFT exceeds HEFT_D, MSMD, and EPRD in terms of average MRR by 4.1%, 3.71%, and 0.6%, respectively. However, MOHEFT uses 20.41%, 31.6%, and 32.9% more VMs than HEFT_D, MSMD, and EPRD to achieve this performance, respectively. Testing with large-scale workflow usually requires the use of thousands of VMs. One percentage point more resources corresponds to 12 more VMs. With the same or fewer numbers of VMs than typical, the EPRD algorithm consistently outperforms HEFT_D and MSMD in the experimental comparisons with the four real-world workflows.

For all of the four real-world applications with different end-to-end deadline constraints (that is, $T_D = 1.5 \times T_C$, $2.0 \times T_C$, $2.5 \times T_C$, and $3.0 \times T_C$), the EPRD algorithm performs more steadily than HEFT_D and MSMD in terms of MRR without using additional VMs. MOHEFT can reduce the makespan rate more than the three other algorithms. However, MOHEFT does so at the expense of consuming a larger number of VMs relative to the three other algorithms. Fig. 13 shows that MOHEFT consumes 4.46%, 5.53%, and 5.41% more VMs than HEFT_D, MSMD, and EPRD, respectively, for the LIGO application.

For the same workflow application, the EPRD algorithm generally has a higher MRR than the three other algorithms when EPRD consumes relatively fewer resources without violating the end-to-end deadline constraints.

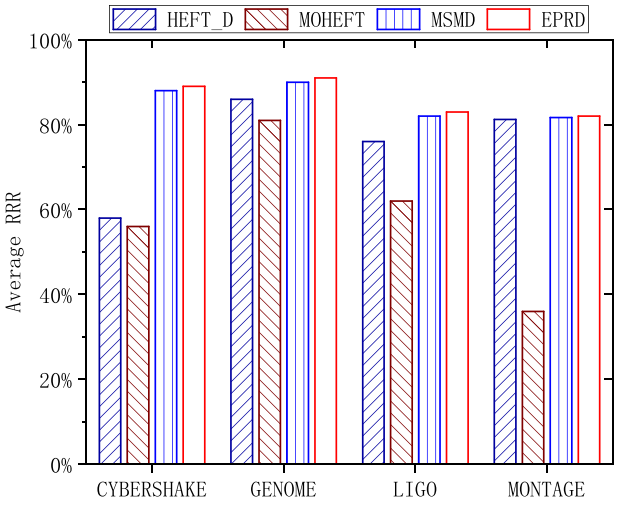

Fig. 7. Average RRR comparison for application with $T_D = 1.5 \times T_C$.

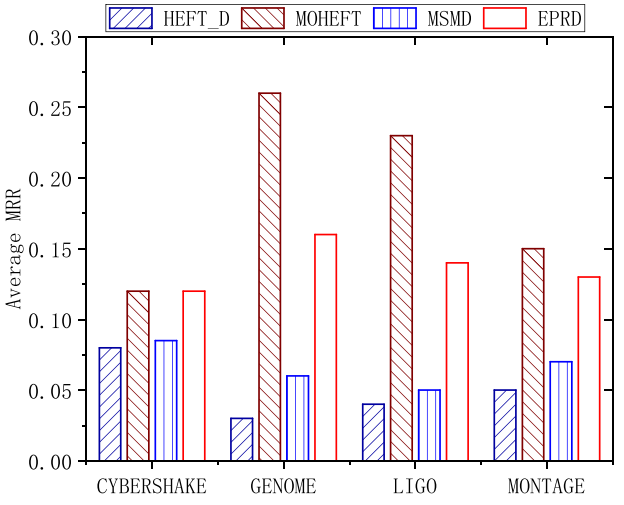

Fig. 8. Average MRR comparison for application with $T_D = 1.5 \times T_C$.

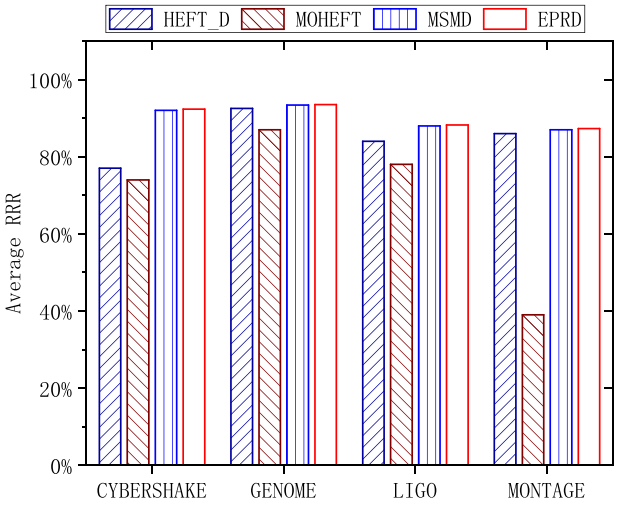

Fig. 9. Average RRR comparison for application with $T_D = 2.0 \times T_C$.

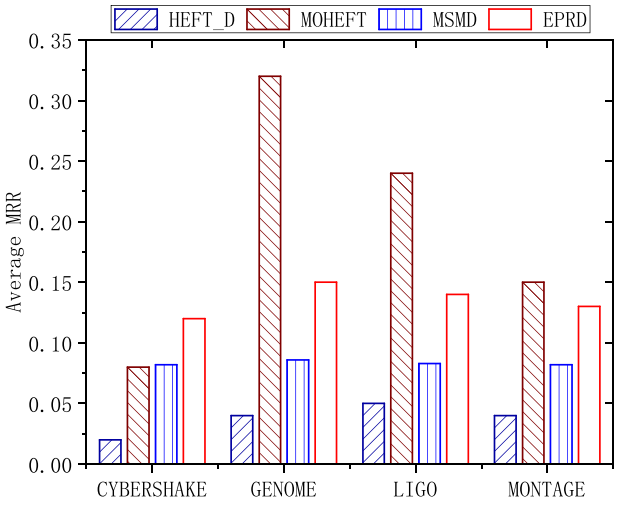

Fig. 10. Average MRR comparison for application with $T_D = 2.0 \times T_C$.

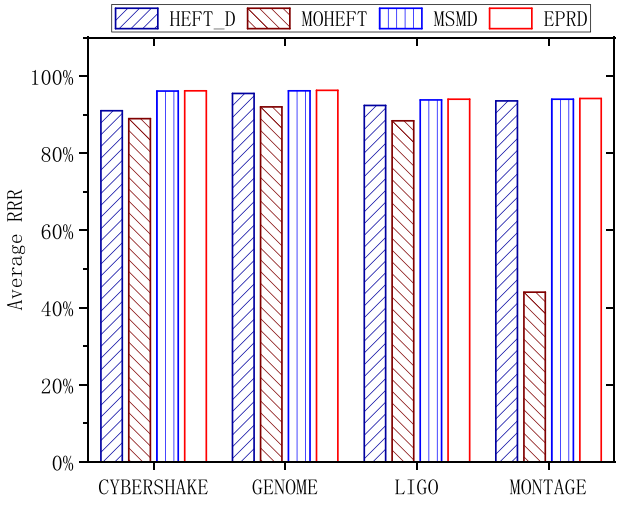

Fig. 11. Average RRR comparison for application with $T_D = 2.5 \times T_C$.

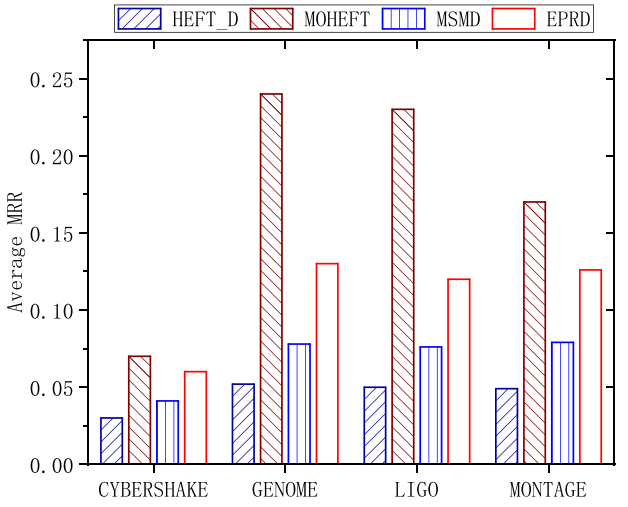

Fig. 12. Average MRR comparison for application with $T_D = 2.5 \times T_C$.

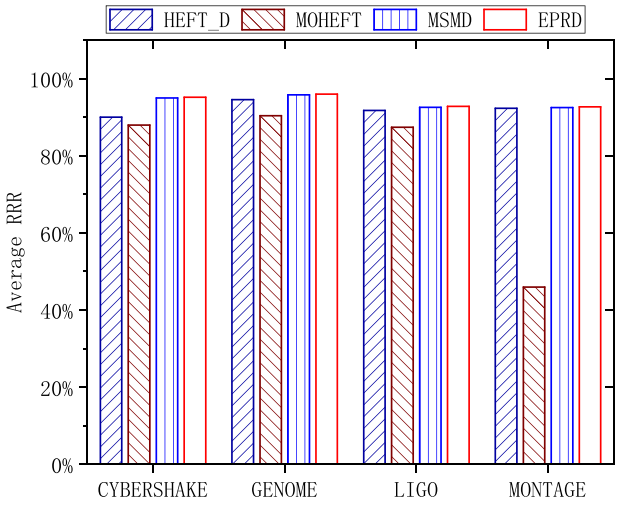

Fig. 13. Average RRR comparison for application with $T_D = 3.0 \times T_C$.

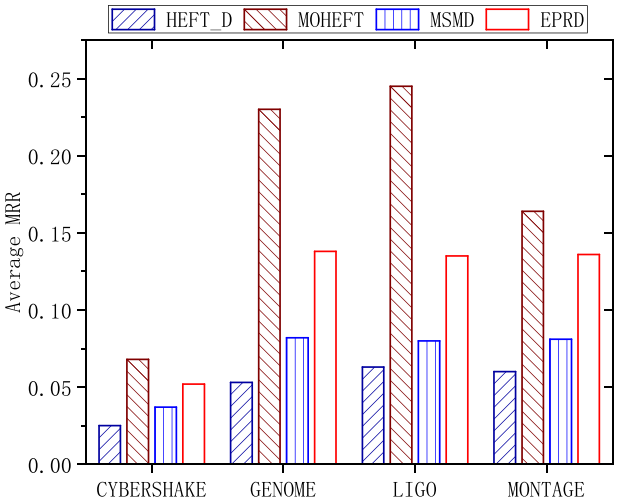

Fig. 14. Average MRR comparison for application with $T_D = 3.0 \times T_C$.

6. Conclusion

This study aims to incorporate VM utilization and minimization of the complete time for parallel task sets while satisfying its end-to-end deadline constraints. The execution sequence of each task in the workflow application is determined by its *DRank* and *maxslack* values. Thereafter, each task is efficiently assigned to the VM in accordance with its relative distance.

The experimental results reveal that the proposed algorithm, namely, ERPD, is comparable to the three other popular algorithms. The benchmarks used in the experiment include random workflow applications and real-world workflow applications, namely, Montage, Cybershake, Epigenomics, and Genome. The experiment results show that ERPD significantly surpasses the three other algorithms in terms of RRR and MRR.

Energy consumption and security are two crucial metrics in cloud computing. One planned future work is to study energy saving and security enhancement for precedence-constrained workflow applications in cloud computing while guaranteeing service quality.

Declaration of Competing Interest

The authors declare that they do not have any financial or nonfinancial conflict of interests

CRediT authorship contribution statement

Longxin Zhang: Investigation, Conceptualization, Methodology, Software, Writing - original draft, Data curation. **Liqian Zhou:** Software, Validation, Supervision, Writing - review & editing. **Ahmad Salah:** Data curation, Software, Validation, Writing - review & editing.

Acknowledgment

This work was partially funded by the National Key R&D Program of China (Grant No. 2018YFB1003401), the National Natural Science Foundation of China (Grant Nos. 61702178, 61672224, 61871432), the Natural Science Foundation of Hunan Province (Grant No. 2019JJ50123, 2018JJ4063, 2018JJ4068, 2019JJ60054, 2019JJ60008), the Key Program of Education Bureau of Hunan Province (No. 17A052), the Research Foundation of Education Bureau of Hunan Province (No. 18C0528), and in part by China Scholarship Council (No. 201808430297).

References

[1] [Online], Available. (http://marketresearchmedia.com/?p=839, Jan. 2014).
[2] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, ACM SIGOPS Oper. Syst. Rev. 37 (5) (2003) 29–43.
[3] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[4] K. Shvachko, H. Kuang, S. Radia, R. Chansler, et al., The hadoop distributed file system., in: MSST, vol. 10, 2010, pp. 1–10.
[5] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets., HotCloud 10 (10-10) (2010) 95.
[6] M.R. Garey, D.S. Johnson, Computers and intractability, 1979.
[7] A. Arunarani, D. Manjula, V. Sugumaran, Task scheduling techniques in cloud computing: aliterature survey, Future Gener. Comput. Syst. 91 (2019) 407–415.
[8] Y. Chen, K. Li, W. Yang, G. Xiao, X. Xie, T. Li, Performance-aware model for sparse matrix-matrix multiplication on the sunway TaihuLight supercomputer, IEEE Trans. Parallel Distrib. Syst. 30 (4) (2019) 923–938.
[9] C. Chen, K. Li, A. Ouyang, K. Li, FlinkCL: an OpenCL-based in-memory computing architecture on heterogeneous CPU-GPU clusters for big data, IEEE Trans. Comput. 67 (12) (2018) 1765–1779.
[10] K. Li, Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment, Future Gener. Comput. Syst. 82 (2018) 591–605.
[11] T. Maqsood, N. Tziritas, T. Loukopoulos, S.A. Madani, S.U. Khan, C.-Z. Xu, A.Y. Zomaya, Energy and communication aware task mapping for MPSoCs, J. Parallel Distrib. Comput. 121 (2018) 71–89.
[12] J. Chen, K. Li, K. Bilal, K. Li, S.Y. Philip, et al., A bi-layered parallel training architecture for large-scale convolutional neural networks, IEEE Trans. Parallel Distrib. Syst. 30 (5) (2018) 965–976.
[13] B. Wang, Y. Song, J. Cao, X. Cui, L. Zhang, Improving task scheduling with parallelism awareness in heterogeneous computational environments, Future Gener. Comput. Syst. 94 (2019) 419–429.
[14] F.M.M. ul Islam, M. Lin, L.T. Yang, K.-K.R. Choo, Task aware hybrid DVFS for multi-core real-time systems using machine learning, Inf. Sci. 433 (2018) 315–332.
[15] J. Chen, K. Li, H. Rong, K. Bilal, K. Li, S.Y. Philip, A periodicity-based parallel time series prediction algorithm in cloud computing environments, Inf. Sci. 496 (2019) 506–537.
[16] B. Qureshi, Profile-based power-aware workflow scheduling framework for energy-efficient data centers, Future Gener. Comput. Syst. 94 (2019) 453–467.
[17] O. Selvitopi, G.V. Demirci, A. Turk, C. Aykanat, Locality-aware and load-balanced static task scheduling for mapreduce, Future Gener. Comput. Syst. 90 (2019) 49–61.
[18] X. Huang, L. Zhang, R. Li, L. Wan, K. Li, Novel heuristic speculative execution strategies in heterogeneous distributed environments, Comput. Electr. Eng. 50 (2016) 166–179.
[19] C. Chen, K. Li, A. Ouyang, Z. Zeng, K. Li, GFlink: an in-memory computing architecture on heterogeneous CPU-GPU clusters for big data, IEEE Trans. Parallel Distrib. Syst. 29 (6) (2018) 1275–1288.
[20] P. Zhang, M. Zhou, Dynamic cloud task scheduling based on a two-stage strategy, IEEE Trans. Autom. Sci. Eng. 15 (2) (2017) 772–783.
[21] H. Chen, X. Zhu, D. Qiu, L. Liu, Z. Du, Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds, IEEE Trans. Parallel Distrib. Syst. 28 (9) (2017) 2674–2688.
[22] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, Inf. Sci. 379 (2017) 241–256.
[23] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization, Soft Comput. (2018) 1–20.
[24] Y. Xu, K. Li, L. He, L. Zhang, K. Li, A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 26 (12) (2015) 3208–3222.
[25] H. Ali, U.U. Tariq, Y. Zheng, X. Zhai, L. Liu, Contention & energy-aware real-time task mapping on NoC based heterogeneous MPSoCs, IEEE Access 6 (2018) 75110–75123.
[26] L. Zhang, K. Li, W. Zheng, K. Li, Contention-aware reliability efficient scheduling on heterogeneous computing systems, IEEE Trans. Sustain. Comput. (3) (2018) 182–194.
[27] H.R. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[28] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.-G. Talbi, A.Y. Zomaya, D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, J. Parallel Distrib. Comput. 71 (11) (2011) 1497–1508.
[29] L. Ramakrishnan, J.S. Chase, D. Gannon, D. Nurmi, R. Wolski, Deadline-sensitive workflow orchestration without explicit resource control, J. Parallel Distrib. Comput. 71 (3) (2011) 343–353.
[30] J.J. Durillo, V. Nae, R. Prodan, Multi-objective energy-efficient workflow scheduling using list-based heuristics, Future Gener. Comput. Syst. 36 (2014) 221–236.
[31] T. Xiaoyong, K. Li, Z. Zeng, B. Veeravalli, A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems, IEEE Trans. Comput. 60 (7) (2011) 1017–1029.
[32] R. Van den Bossche, K. Vanmechelen, J. Broeckhove, Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds, Future Gener. Comput. Syst. 29 (4) (2013) 973–985.
[33] G. Xiao, K. Li, Y. Chen, W. He, A. Zomaya, T. Li, CASpMV: a customized and accelerative SpMV framework for the Sunway TaihuLight, IEEE Trans. Parallel Distrib. Syst. (2019).

[34] A. Woodley, L.-X. Tang, S. Geva, R. Nayak, T. Chappell, Parallel k-tree: a multicore, multinode solution to extreme clustering, Future Gener. Comput. Syst. 99 (2019) 333–345.
[35] H. Peng, W.-S. Wen, M.-L. Tseng, L.-L. Li, Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment, Appl. Soft Comput. 80 (2019) 534–545.
[36] H. Wu, X. Hua, Z. Li, S. Ren, Resource and instance hour minimization for deadline constrained DAG applications using computer clouds, IEEE Trans. Parallel Distrib. Syst. 27 (3) (2015) 885–899.
[37] [Online] Available. (https://confluence.pegasus.isi.edu/display/pegasus/workflowgenerator, 2014).
[38] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: 2008 Third Workshop on Workflows in Support of Large-Scale Science, IEEE, 2008, pp. 1–10.