Cluster Comput (2019) 22:S693–S706
https://doi.org/10.1007/s10586-017-1189-5

DSM: a dynamic scheduling method for concurrent workflows in cloud environment

Shengjun Xue[1,2] · Yue Peng[1,2] · Xiaolong Xu[1,3] · Jie Zhang[3] · Chao Shen[1,2] · Feng Ruan[4]

Received: 26 July 2017 / Revised: 8 September 2017 / Accepted: 11 September 2017 / Published online: 22 September 2017

Abstract Cloud computing, emerged as a commercial service model, has been widely concerned in both industry and academia. Massive workflow applications could be performed simultaneously on the cloud platforms, which significantly benefits from the elasticity and convenience of cloud computing. However, it is still a challenge to schedule virtualized resources for the concurrent workflows in cloud environment, with limited high-performance resources in a timesaving and efficient manner. In view of this challenge, a dynamic scheduling method for concurrent workflows, named as *DSM*, in cloud environment is proposed to satisfy the various resource requirements of the workflows. Technically, a time overhead model for the workflows and a resource utilization model for cloud datacenter are presented. Then a relevant dynamic scheduling method is designed based on critical path lookup, which aims at minimizing the makespan of workflows, and maximizing the resource utilization of the datacenter during the execution of the workflows. Extensive experimental evaluations demonstrate the efficiency and effectiveness of our proposed method.

Keywords Cloud computing · Workflow scheduling · Makespan · Resource utilization

✉ Xiaolong Xu
xlxu@nuist.edu.cn

Shengjun Xue
sjxue@163.com

Yue Peng
joyous_yp@163.com

Jie Zhang
jzhangchina@126.com

Chao Shen
shen_chao@126.com

Feng Ruan
ruanfeng@hotmail.com

1 Jiangsu Engineering Center of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing 210044, China

2 School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China

3 State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

4 School of Information and Control, Nanjing University of Information Science and Technology, Nanjing 210044, China

1 Introduction

Nowadays, cloud computing has been adopted as one of the most efficient compute paradigms, to deal with the explosive expansion of data. Cloud computing integrates abundant heterogeneous resources while improving the adaptability to commercialization [1]. In Cloud datacenters, physically configurable and fragmented underlying resources (i.e., networks, servers, etc.) are generally virtualized to a logically clustered virtual resource pool, which manages to realize dynamic resource provision for an increasing number of users [2,3]. Cloud computing provides convenient network services, which allows users to rent on-demand resources for their applications through Internet and charge in the pattern of "pay as you go" [4]. Such resource provision and charge manner contribute to reducing the operating costs of the IT industry [5]. Currently, a variety of applications are submitted to the cloud platforms for implementation, including the workflow applications.

Cloud workflow scheduling is an essential part of cloud computing technology, which directly affects the performance of the entire workflow system [6]. In order to provide the users good quality of cloud services, rapid execution

time for the cloud applications is necessary to be taken into consideration during workflow scheduling, which is often reflected in the quality of service (QoS) [7]. Taking minimal total execution time as the goal, task-level scheduling in the cloud environment splits the workflow application into several interrelated tasks. And these tasks need to be assigned to the virtual machines (VMs), which requires powerful methods and techniques to guarantee the time efficiency for the execution of workflow applications [8]. The workflow makespan is determined by the task dependency, as well as the service performance provided by the cloud providers [9]. The scheduling strategy has a widely impact on the execution time of the workflows, thus it is necessary to consider the goal of minimizing the makespan for the concurrent workflows.

In addition to the makespan, the performance of the cloud platforms is an indispensable part for quality measurement of cloud datacenters. Generally, resource utilization is a popular metrics for performance evaluation for cloud platforms [10]. Although cloud resources are unlimited, inefficient resource usage would lead to the durative expansion of the cloud datacenters and the significant increase of operating costs [11]. The efficient use of resources helps to prevent unnecessary expansion of data centers and greatly reduces the waste of a variety of physical resources. While satisfying the resource requirements of substantial tasks, cloud providers expect to take full advantage of active physical machines (PMs) for resource response, which is beneficial to reducing the operating costs of datacenter. In the process of workflow execution, high resource utilization also helps to reduce the total number of PMs occupied and then reduce the waiting time of users. Therefore, it is significant to conduct workflow scheduling with the goal of maximize resource utilization.

However, to the best of our knowledge, there are few of studies that succeed in obtaining good achievements in both makespan and resource utilization. With these observations, it is still a challenge to complete cloud workflow scheduling in time-saving and efficient manners. To tackle this problem, we focus on investigating a dynamic scheduling method for concurrent workflows, named as *DSM*, in cloud environment. Our contributions are three folds. Firstly, a time overhead model for the workflows and a resource utilization model for cloud datacenters are designed. Then a relevant dynamic scheduling method for concurrent workflows is devised to reduce the time overhead of workflows and maximize the resource utilization of the cloud datacenter during the workflow execution process. Finally, we carry out comprehensive experiments to verify the efficiency and effectiveness of our proposed method.

The remainder of this paper is summarized as follows: Related work is described in Sect. 2. Section 3 presents time overhead and resource utilization analysis in cloud environment. The proposed dynamic workflow scheduling method is described in detail in Sect. 4. Section 5 introduces the simulation experiment to test and verify proposed method. This paper is summarized in Sect. 6 and an outlook on possible future work is also given in this section.

2 Related work

After cloud computing integrated numerous heterogeneous resources, resource scheduling strategies are applied to organize and schedule the resources, which have important implications for making full use of resources. Meanwhile, workflow scheduling strategies are applied to dispatch tasks submitted to the cloud in the way of minimizing the total time for task execution from the user perspective. Both of them play important roles in cloud computing.

In order to provide the users good quality of cloud services, workflow scheduling methods for the cloud applications with rapid execution time have aroused a great deal of discussion [12]. There are a number of methods that have been designed, where various metrics are taken into account to measure the performance, such as load balancing, fairness and so on. Hence, these measurement tags should be analyzed with a certain necessity while designing an intelligent scheduling method, as described in [13–18].

In [13], the graph partitioning algorithm was used by Ahmad et al. to minimize communication between intermediate nodes during workflow execution, but the problem of workflow resource scheduling after partitioning was not set about, so that it is not applicable to the multi-resource cloud environment. For the sake of interminable time overhead and poor utilization, Chen brought up a workflow scheduling method on hybrid cuckoo search and decision tree which minimizes data dependent between tasks and selects optimal resources for tasks [14]. In [15], Liu et al. proposed a workflow scheduling method based on batch processing strategy to deal with the fierce competition of low-cost resources between tasks in the cloud environment, which achieves a good tradeoff between time and cost. Aiming at optimization of the accuracy in business workflow with time constraint, a reverse reduction optimization method based on deadline was developed to obtain the balance of time and accuracy within deadline in [16]. In [17], Doulamis et al. proposed a resource selection policy using spectral clustering, which maximizes the satisfaction of tasks' time requirements and resources' utilization efficiency synchronously. Kong et al. [18] proposed a dispersive belief propagation-based allocation method for multi-agent tasks to cope with constantly time-varying agents and tasks in dynamic cloud environments.

In addition to time, resource utilization in cloud computing continues to be highly concerned by all circles of society. In order to extricate cloud suppliers from poor resource utilization and heavy equipment cost, it is significant to develop

effective scheduling strategies with high resource utilization [19]. In order to improve the resource utilization of datacenters, scholars have conducted a variety of researches, as described in [20–24]. Some studies take both of the two into account as goals at the same time and strive to strike a balance.

A scheduling optimization algorithm based on task tolerance was proposed in [20], where resource utilization of computing node is maximized by adjusting the value of task tolerance put forward in the article. Cao et al. [21] focused on a multi-step heuristic workflow scheduling algorithm, which maximizes energy conservation and resource utilization under the premise of guaranteeing service quality. Considering the priority execution order of tasks in workflow, Jard et al. [22] proposed a scheduling algorithm based on pre-calculus for instance-intensive workflow scheduling problem. The algorithm provides idle resource for execution according to the order of task priority from high to low before scheduling. For improving the resource utilization to optimize the system performance, and taking the constraints of QoS into account, Wang et al. [23] built a system scheduling model with two scheduling stages. Then an improved particle swarm optimization algorithm and a load-aware scheduling strategy were proposed based on the model to achieve the balance between resource utilization and QoS. Rodriguez et al. [24] researched scientific workflow scheduling in cloud, with QoS and resource utilization are seen as main factors, in addition, they also took factors such as performance, reliability, capacity and cost into consideration.

3 Time overhead and resource utilization analysis in cloud environment

After workflow applications are submitted to the network and distributed to certain datacenters, workflow scheduling strategies are applied to break workflows into tasks with dependencies, and configure VMs to respond tasks according to some restrictions and principles [25]. Afterwards, the datacenter dispatches resources to these VMs with corresponding resource scheduling policies.

The purpose of this paper is to reduce the time overhead of workflow, denoted as t_{total}, while maximizing the average resource utilization of the datacenter, denoted as *ARUR*, under the premise of satisfying the resource demand of tasks.

To facilitate the following discussion, the key symbols used in the time overhead model and the resource utilization model are listed in Table 1.

As a result, the objective function of this paper is summarized as:

$$\min\ t_{total}, \quad \max ARUR \tag{1}$$

Table 1 Description of symbols in the time overhead model and the resource utilization model

Symbol	Description
num	The total number of tasks in the workflow set
$t_{m,n}$	The n-th task in the m-th workflow of the workflow set
$st_{m,n}$	The start time of task $t_{m,n}$,
$ft_{m,n}$	The finish time of task $t_{m,n}$,
tw_m	The makespan of executing the workflow wf_m
$tc_{average}$	The average makespan of workflows set WF
$avn_{h,t}$	The number of active VMs on PM s_h at time t
$ru_{h,t}$	The instantaneous resource utilization of PM s_h at time t,
rur_h	The average resource utilization of PM s_h

In Sects. 3.1 and 3.2, computational procedure of t_{total} and *ARUR* are introduced in detail, respectively.

3.1 Time overhead model

Assume there are H PMs, denoted as $S = \{s_h | 1 \le h \le H\}$, available for promoting task execution in the datacenter. In practical applications, the physical resources are composed of CPU, memory, storage capacity, bandwidth and etc. [26]. For the following discussion, here lists a restriction: the physical resources on a PM are encapsulated into several independent resource blocks of same configuration, i.e., a VM [27]. There is no discrepancy in performance of the VMs on different PMs, except the calculation speed. Same configured PMs have exactly identic VMs. Therefore, for a PM $s_h (1 \le h \le H)$, the number of resources it possessed is denoted as v_h, and its processing speed is denoted as ps_h., which is equally divided by all VMs deployed on it.

Besides, the workflow set is represented as $WF = \{wf_m | 1 \le m \le M\}$, where M represents the size of WF. For a workflow $wf_m (1 \le m \le M)$, it is denoted as $wf_m = \{T_m, MA_m, wst_m\}$, where T_m, MA_m, wst_m indicate the task set, the dependencies between tasks and the expected start time of wf_m, respectively. The users submit the service request in a random way, so the request start time of workflow applications are subject to random distribution. In addition, wst_m is predetermined while the workflow is submitted to datacenter. As $t_{m,n}$ represents the n-th task of the workflow wf_m, the task set T_m is denoted as $T_m = \{t_{m,n} | 1 \le n \le N_m\}$, where N_m is the number of tasks in wf_m. Therefore, the total number of tasks in WF, indicated as num, can be calculated as:

$$num = \sum_{m=1}^{M} \sum_{n=1}^{N_m} t_{m,n} \tag{2}$$

According to dependency set, which is denoted as $MA_m = \{t_{m,i}, t_{m,j} | 1 \le i, j \le N_m\}$, the direct predecessor task set and the direct successor task set of a task $t_{m,n} (1 \le n \le N_m)$, represented as $Pre_{m,n}$ and $Suc_{m,n}$ respectively, are obtained. A node without a predecessor task is called a source task, while a node without a successor task is called a sink task [28]. In addition to the source and the sink, each task has at least one predecessor task and a successor task.

To facilitate the following research, a task $t_{m,n}$ $(1 \le n \le N_m)$ is defined as: $t_{m,n} = (st_{m,n}, l_{m,n}, ft_{m,n}, vn_{m.n}, p_{m,n})$, where $st_{m,n}$, $l_{m,n}$, $ft_{m,n}$, $vn_{m,n}$ and $p_{m,n}$ represent the start time, the amount of computation, the finish time, the number of VMs needed for execution and the number of direct predecessors of $t_{m,n}$, respectively, and $p_{m,n} = |Pre_{m,n}|$. As the characteristic of workflows mentioned above, $st_{m,n}$ depends on the implementation of all tasks in $Pre_{m,n}$. For task $t_{m,n}$, if $p_{m,n} = 0$, i.e., $t_{m,n}$ is the source task of wf_m, $st_{m,n} = wst_m$, otherwise $st_{m,n}$ equals to the latest finish time of tasks in $Pre_{m,n}$. Assume that there are k tasks in $Pre_{m,n}$, the start time of $t_{m,n}$, i.e., $st_{m,n}$ can be calculated as:

$$st_{m,n} = \max_{0 \le i \le k-1} ft_i, t_i \in Pre_{m,n} \tag{3}$$

Consequently, the finish time of $t_{m,n}$, i.e., $ft_{m,n}$ can be calculated as:

$$ft_{m,n} = st_{m,n} + \frac{l_{m,n} / v_{m,n}}{ps_h / v_h} \tag{4}$$

which is translated as:

$$ft_{m,n} = st_{m,n} + \frac{l_{m,n} \cdot v_h}{v_{m,n} \cdot ps_h} \tag{5}$$

where ps_h / v_h is the processing speed of a VM in the selected PM s_h, and $(l_{m,n} \cdot v_h)/(vn_{m,n} \cdot ps_h)$ indicates the execution time that s_h serves $t_{m,n}$ under the circumstance that $t_{m,n}$ is executed on s_h all the time. However, $ft_{m,n}$ varies if the task is rescheduled, so that, $ft_{m,n}$ should be obtained through calculating the time re-scheduling occurs, remaining workload, and the performance of the PM reselected [29].

The makespan of $wf_m (1 \le m \le M)$, defined as tw_m, is determined by the tasks first performed and the task last finished. Therefore, tw_m could be calculated as:

$$\mathrm{tw}_m = \max_{1 \le n \le N_\mathrm{m}} ft_{m,n} - \min_{1 \le n \le N_m} st_{m,n} \tag{6}$$

since the earliest start time of tasks in the workflow is limited by the default start time of workflow, i.e., wst_m, thus, the formula above is equivalent to:

$$tw_m = \max_{1 \le n \le N_\mathrm{m}} ft_{m,n} - wst_m \tag{7}$$

The average makespan of workflows in WF, $tc_{average}$, can be expressed as:

$$tc_{average} = \sum_{m=t}^{M} tw_m / M \tag{8}$$

According to the content above, the total time overhead of the workflow setWF, expressed as t_{total}, can be calculated using the following formula:

$$t_{total} = \max_{1 \le n \le N_\mathrm{m}, 1 \le m \le M} ft_{m,n} - \min_{1 \le m \le M} wst_m \tag{9}$$

3.2 Resource utilization model

Every moment during execution, there may be new workflows scheduled or existing workflows completing, therefore, the instantaneous resource utilization is time-varying, so is the average resource utilization of a PM [30]. In order to illustrate the instantaneous resource utilization, the number of active VMs on s_h at t is denoted as $avn_{h,t}$ and calculated as:

$$\mathrm{avn}_{h,t} = \sum_{1 \le m \le M} \sum_{1 \le n \le N_\mathrm{m}} \left(f_{m,n,h,t} \cdot vn_{m,n} \right) \tag{10}$$

where $f_{m,n,h,t} = 1$ while the task $t_{m,n}$ is employing resources on s_h at time t, otherwise, $f_{m,n,h,t} = 0$. As a result, the definition of instantaneous resource utilization is given here: for any PM s_h $(1 \le h \le H)$, the instantaneous resource utilization at time t, denoted as $ru_{h,t}$, equates to the percentage of occupied VMs in the number of total resources in the PM. Since $avn_{h,t}$ is known, $ru_{h,t}$ can be calculated as:

$$ru_{h,t} = \mathrm{avn}_{n,t} / v_h \tag{11}$$

As $ru_{h,t}$ is figured out in formula (11), the average resource utilization of PM s_h in the whole process of serving the users can be obtained by $ru_{h,t}$, and the active time of PM that serving for the workflows, presented as T_h, namely, for a PM $s_h (1 \le h \le H)$, the average resource utilization is defined as rur_h and can be calculated as:

$$\mathrm{rur}_h = \int_0^{T_k} ru_{h,t} \Big/ T_h \tag{12}$$

The average utilization of a single PM is inadequate to evaluate the performance of entire datacenter while serving users, so another new concept of average utilization of datacenter is given here: the average resource utilization of PMs occupied by the workflow set WF during the whole execution is defined as $ARUR$, which can be calculated with the utilization of all active PMs and the time span for implementation of WF as follow:

Step1: Critical Path Lookup. In this part, we choose dynamic programming algorithm as the way to analyze the critical paths according to workflow structures, and then the priorities are assigned as significant basis for tasks selecting resource.
Step2: Time and Utilization Aware Task Scheduling. When dealing with the waiting queue, the method first considers the priorities of tasks to reduce makespan of each workflow, dispatches tasks to PMs according to the real-time utilization of each PM.
Step3: Priority Driven Resource Preemption. New tasks with higher priority have the right to seize the VMs on high performance PMs from lower-priority ones while looking for resources, if the resources lower-priority tasks occupied is sufficient.

Fig. 1 Specifications of *DSM* method

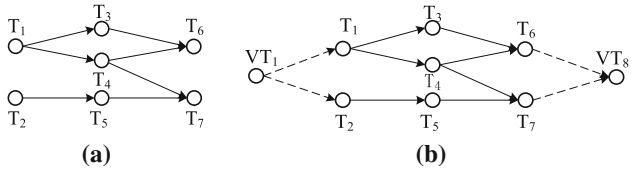


Fig. 2 An example of workflow DAG conversion. **a** Workflow with multi sources and multi sinks. **b** Workflow transformed into single source and single sink

$$ARUR = \sum_{h=1}^{PN} \int_0^{T_k} ru_{h,t}/PN \cdot t_{total} \tag{13}$$

where PN is the number of PMs that the tasks in WF has been scheduled on for execution.

4 A dynamic scheduling method for concurrent workflows

Base on the analysis in Sect. 3, a relevant dynamic scheduling method for concurrent workflows, named as *DSM*, in cloud environment is proposed. The method is comprised of following parts, i.e., critical path lookup, task scheduling based on time and utilization and resource preemption for critical tasks, as shown in Fig. 1. And the global workflow scheduling is proposed at the end of this section which consists of the following parts.

4.1 Critical path lookup

Logically, a workflow is composed of thousands of tasks and dependencies between tasks. There needs a lot of resources and a reasonable and effective scheduling approach based on the dependency between tasks to co-ordinate execution of the tasks [31]. A task is assigned to execute after all its predecessor tasks have been accomplished. In contrast, the tasks, which belong to concurrent workflows, have no dependencies with each other, can be assigned simultaneously.

According to the feature mentioned above, a workflow is generally analyzed as a directed acyclic graph (DAG) and scheduled in the form of tasks which have complex dependencies with each other [32]. As an effective tool for describing the process of an engineering or system, a DAG consists of vertices and edges, and vertices are usually bound by certain conditions, which is represented by the edges. According to the constraints between workflow tasks, tasks and dependencies in the workflow are transformed into vertices and edges respectively. Generally speaking, workflows in actual application may have multiple start nodes and multiple end nodes, like the example shown in Fig. 2a. It is difficult to analyze this structure in details, so we add virtual nodes to turn the situation into cases like single source and single sink which have no processing requirements in principle, as shown in Fig. 2b. In this paper, due to workflow properties, there is a default that the first task of each workflow is the source task.

Under the above assumptions, this paper proposes a workflow scheduling algorithm for workflows with single source and single sink. Thus, as workflow $wf_m = \{T_m, MA_m, wst_m\}(1 \le m \le M)$ is expressed by DAG, T_m is the node set and MA_m is the set of edges between nodes.

The source node in a DAG is first assigned for implementation, and the scheduling sequence of each task is strictly in accordance with the structure, i.e., each task can be assigned to the PM upon all of predecessor tasks are completed. If all the tasks are processed with same resources, the path with the largest amount of computation in all paths from source to sink is called the critical path of the DAG, whose processing time can be usually regarded as the execution time of the workflow. It is impossible for the tasks on the critical path to be executed concurrently, so the execution time of the critical path cannot be compressed. Therefore, the critical path of each workflow has a great influence on the makespan of a workflow. It is important to emphasize that tasks on non-critical paths are also performed, while the task on the critical path has the right to choose better resources.

In general, workflow can be regarded as a mixed structure where serial structure and parallel structure coexist. The completion time of a serial workflow is equal to the sum of the execution times of all tasks, while that of a parallel workflow depends on the maximum execution time of all tasks [33]. For the hybrid workflow, with scope expansion of task activities, time cost can be effectively optimized by determining mapping relation between critical tasks and high-performance resources [34].

In order to simplify the algorithm, this paper uses an attribute of task, start time, to limit the execution order of the workflow. It should be noted that in actual scheduling, the start time of next task is influenced by many terms, especially the size of data transferred and bandwidth between PMs executing dependent tasks. Applications are classified as data-intensive, computationally intensive and IO-intensive by type. This research mainly focuses on the processing time of computing intensive applications in datacenters so that *DSM* ignores the transmission time, considers that the start

time of subsequent tasks equals to the end time of precursor task.

Concurrent workflows refer to workflows that are mutual independence. Massive applications can be submitted to the cloud platform by diverse users at any moment which are neither ordered nor relevant. Universally, existing workflow scheduling algorithms are mostly aimed at single workflow applications rather than concurrent workflows, so research on concurrent workflows is essential.

Once a workflow arrives at the datacenter, the source node and the sink node should be founded according to the dependencies between tasks. Due to the hypothesis above, the dynamic programing algorithm is used to find the critical paths of the workflows. Combining the breadth-first idea, the algorithm selects the key path of every node in turn according to dependencies, constructing the critical path rapidly, and finally assigns highest priority to the task on the critical path. As a result, this algorithm skips topology sorting, reduces the difficulty of analysis, and cuts execution time. After execution, the task $t_{m,n}$ sends a signal to all the tasks in $Suc_{m,n}$ and changes their value of unprocessed predecessor $rp_{m,n+1}$. For every task in $Suc_{m,n}$, the following dynamic programming transfer equation is used to verify whether the current path is the critical path from the source to the current node $t_{m,n}$.

$$maxload_{m,n} = \max_{i \in Pre_{m,n}} \{maxload_{m,n}, maxload_{m,i} + l_{m,n}\} \tag{14}$$

$$maxload_{m,st} = l_{m,st} \tag{15}$$

where $maxload_{m,n}$ is the maximum workload from the source node to $t_{m,n}$.

The critical path lookup algorithm based on dynamic programming is shown as Algorithm 1. Firstly, all the tasks are added to the queue Q (line 1). If Q is not empty, then there exists a workflow to be processed. When processing a task t, its successor set is traversed to fine the critical path (line 8–16).

4.2 Time and utilization aware task scheduling

After analyzing the critical paths, there is another question should not be overlooked, that is, the makespan of the workflow is not necessarily reduced if the tasks on the critical path get better resource.

For example, assume that there are only two kinds of PMs, with processing speed are 2 and 1 MIPS respectively. While dealing with the workflow shown as Fig. 3, the critical path $T_0 \rightarrow T_3 \rightarrow T_4 \rightarrow T_7$ can be easily obtained by dynamic programming presented in Algorithm 1. Table 2 shows the calculation amount of each node in Fig. 3. Since the tasks on critical path executed on the PMs with higher

Algorithm 1: Critical Path Lookup (wf_m)

Input: The workflow wf_m
Output: The set of critical tasks kn_m of wf_m
1:Add all tasks in wf_m to the queue Q
2:Get $Suc_{m,n}$ of $t_{m,n}$ according to MA_m // $t_{m,n}$ is the source node of wf_m
3:while $Q \neq \varnothing$ **do**
4: **for** i=0 to $tsnum$ **do**
5: **if** t is ready **then** //t is the i-th task
6: pl=$t.PathLength$
7: Get the successor set of t as suc
8: **for** j=0 to suc.size() **do**
9: Get the j-th task in suc as t_1
10: l= l_{t1}+pl, cp= CP_t
11: **if** $t_1.PathLength$<l **then**
12: Add t_1 and cp as CP_{t1}
13: **else if** $t_1.PathLength$==l **then**
14: Add t_1, cp and CP_{t1} as CP_{t1}
15: **end if**
16: **end for**
17: Remove t from Q
18: **end if**
19: **end for**
20:end while
21: Get the critical path of sink node as cps
22: Set the priority of all nodes

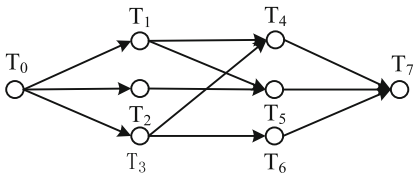


Fig. 3 An example of workflow with 8 tasks ($T_0 \sim T_7$)

efficiency due to their higher priority, execution time of the critical path is 8. However, the time cost of the second longest path,$T_0 \rightarrow T_1 \rightarrow T_5 \rightarrow T_7$, is 11.5. Thus, it is not enough to only consider the optimization of critical path. Unfortunately, existing algorithms cannot efficiently calculate the second longest path, and if the second longest path is obtained, then it is bound to consider the third long path, then the algorithm has no difference from exhaustion.

In order to avoid the above problem, a parameter a is designed to limit the percentage of tasks with higher priority in the workflow, that is, the parameter is used to optimize the implementation of tasks on non-critical paths. Tasks selected by a are given the second highest priority and have the right to take efficient resources for execution without affecting the implementation of critical tasks.

It is unquestionable that the order in which the tasks are scheduled lies on the dependencies between tasks and can be converted to the attribute $st_{m,n}$ of the task $t_{m,n}$ [35]. So it should be elaborated that when matching resources for the tasks, the datacenter only consider the task in the wait queue and the running state of PMs, and workflow scheduling is carried out according to the PM resource state. If a VM handles one task at a time, resource blocks cannot be re-split for multiple tasks.

Table 2 Attributes of tasks

Task	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Length	1	3	2	4	7	6	4	4

Since the completion of critical path lookup and priority assignment, it is necessary to schedule the task to PMs for implementation. Assigning tasks to active hosts which are serving users is an important way to increase resource utilization, for which it is necessary to monitor the resources utilization of each PM. In order to achieve the goal of monitoring PM resource utilization, each PM is configured with an allocation records table which consisted of multiple records of the process of resource allocation from the PM point of view. For any PM s_h, a record in the allocation table is defined as $Allor_{h,i} = (wid, t, pri_t, st_t, ot_t, vn_t)$, where wid, t, pri_t, st_t, ot_t, vn_t represents the workflow that the tasks assigned belongs, the task allocated to the PM, the start time of execution and the duration of possession, the amount of resources the task occupied. Besides, the PM allocation record table should be updated upon new tasks' arrival and existing tasks' migration or accomplishment. With the allocation table designed from the PM perspective, a distribution history table is designed for each task to record the scheduling process, i.e., which PMs are occupied at what time and how long the task executed on the PM. For task t, the scheduling record is defined as $Schr_{h,i} = (s, st, ot)$, where s, st, ot represents the PM task t are assigned, the start time of execution and the duration of possession, similar to the definition above.

In particular, a heuristic algorithm called Best Fit Decreasing (BFD) is selected as the tool to complete the research in this paper, it first sorts items in descending order of size, and then find out the smallest free partition which meets the requirements of items successively, making the resource fragmentation as small as possible after allocation. In task scheduling problem, tasks and PMs are described as objects of different size and boxes, so the task scheduling problem can be transformed into a bin-packing problem and solved with BFD.

Datacenter select the active PM whose amount of resources is closest to the task and able to accommodate the task while tasks with high priority have the right to occupy more efficient resources. Whenever a new task collection waits to be processed, the resource usage of PMs in the PM list is analyzed before allocating. What is more, the distribution records of PMs and distribution history records of tasks are updated upon every operation.

Algorithm 2 specifies the process of time and utilization aware task scheduling. First, all PMs in the list S are sorted in ascending order of the remaining resources (line 2), and then the task t is dispatched to the PM with the least amount of remaining resources that can hold the task (line 3–18). Once the PM is determined, the task's distribution records and the status information of all tasks in the successor set of t are modified (line 10–15).

Algorithm 2: Time and Utilization Aware Task Scheduling (t, S)

Input: The task t, PM list S
Output: The distribution history record of tasks in TP
1:Count idle resources of S at $Idlenum$
2:Sort S in ascending order according to the amount of idle resources
3:for p=0 to $Ssize$ **do**
4: **if** s_p. $Idlenum$>= vn_i **then** // There is some PM that can handle the task t_i
5: Select s_p as the selected PM for t_i
6: Calculate ft of task t, and alter the attributes of t
7: New an allocation record of t
8: Modify the allocation table of s_p
9: $flag$=1
10: **for** i=0 to Suc_i **do**
11: Get i-th task in Suc_i as t_i
12: **if** all the predecessors of t_i are ready **then**
13: **add** t_i into UNP
14: **end if**
15: **end for**
16: **Remove** t from UNP and TP
17: **end if**
18:end for

4.3 Priority driven resource preemption

The above section gives the task scheduling algorithm based on BFD. However, Algorithm 2 does not take the fact that datacenter PMs cannot be identical. Due to the superior performance of efficient resources, tasks prefer to occupy the efficient resources to escape from redundant waiting time. Furthermore, tasks of high priority have the power to preempt resources from low priority tasks. An example is given as below. There is a workflow set with only two workflows and they are identical shown as Fig. 4 and their attributes are lists in Table 3. A high-performance PM p_1 and several general PMs (p_2, p_3, ...) are provided for execution of the set. Each VM is known to host 4 VMs, meanwhile the processing speed of the high-performance PM is 2 MIPS, and the speed of others are 1 MIPS. Assume that the default start time of wf_1 is 0.0 and that of wf_2 is 0.6, while all the tasks in the set require 2 VMs to support them.

After submitted, the source task T_0 and V_8 are added into the queue UNP, and T_0 is scheduled on p_1 immediately, upon its accomplishment, T_1, T_2 and T_3 are added into UNP with their start time is 0.5. Since T_3 is on the critical path, it is scheduled on p_1, while T_1 is dispatched on p_1 on account of its priority and T_2 is dispatched on p_2. When the time is 0.6, T_8 is dispatched to p_1 owing to its priority, it snatches the resource from T_2. If requirement of T_8 is not met yet, it will grab T_1 of its resources and reschedule T_1 according to BFD. The other tasks are disposed according to the same theory.

When a task with highest priority is being scheduled, the datacenter manager first checks the resource utilization of better PMs, if the idle resource can meet the requirement of

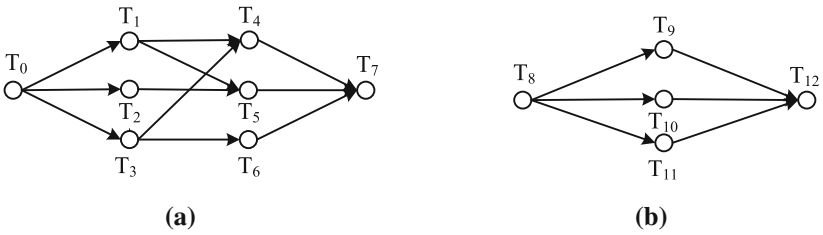


Fig. 4 An example of two concurrent workflows. **a** wf_1. **b** wf_2

Table 3 Attributes of tasks in wf_1 and wf_2

Task	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
Length	1	3	2	4	7	6	4	4	1	3	2	5	2

Algorithm 3: Task Migration on Specific PM(t_i, *st*, *pf*)
Input: Task t_i, time *st*, selected PM *s*, Migration type identifier *pf*
Output: *UNP*
1: Get the amount of idle resource on *s* as *idlenum*
2: **if** *pf*=1 **then**
3: Get all tasks whose priority is 0 as set ts_0 sort ts_0 by *vn*
4: *sum*=*idlenum*, *n*=0
5: **while** *sum*< vn_i **do**
6: Get the *n*-th task in ts_0 as t_0
7: Remove t_0 from *s*, update *Allor* and *Schr*, and add t_0 into *UNP*
8: *n*++, *sum*=*sum*+ vn_{t0}
9: **end while**
10: **end if**
11: **if** *pf*=2 **then**
12: *sum*=*idlenum*, *n*=0
13: Remove all tasks whose priority is 0 at *st* and add them into *UNP*
14: Update *Allor* of *s* and *Schr* of tasks, and modify *sum*
15: Get tasks whose priority are 1 as set ts_1, and sort $ts_1$by *vn*
16: **while** *sum*<*goal* **do**
17: Get the *n*-th task in ts_1as t_1
18: Remove t_1 from *s* at *st* and update *Allor* and *Schr*
29: *UNP*.add(t_1)
20: *n*++, *sum*=*sum*+ vn_{t1}
21: **end while**
22: **end if**

Algorithm 4: Preemption of high-performance resources (t_i, *SH*)
Input: Task t_i, sorted PM list *SH*
Output: The distribution history record of t_i
1: **if** *pid*= Null **then**
2: Sort *SH* in ascending order by *se*
// *se* is the sum of idle resources and resources occupied by tasks whose priority is 0
3: **for** *k*=0 **to** *SHsize* **do**
4: Update *se*
5: **if** p_i=2 or 1 and *se*>= vn_i **then**
6: Select the s_p as the selected PM for t_i
7: *pid*=*k*, *pf*=1
8: **Algorithm 3**(t_i, *st*, *pf*)
9: **end if**
10: **end for**
11: **end if**
12: **if** *pid*=Null **then**
13: Sort *SH* in ascending order by *sei*
//*sei* is the sum of idle resources and resources occupied by tasks whose priority is 0
14: **for** *k*=0 **to** *SHsize* **do**
15: Update *sei*
16: **if** p_i=2 and *sei*>= vn_i **then**
17: Select the s_p as the selected PM for t_i
18: *pid*=*k*, *pf*=2
19: **Algorithm 3**(t_i, *st*, *pf*)
20: **end if**
21: **end for**
22: **end if**

the task, it is handled without snatch. Otherwise, the manager considers seizing the resource other tasks occupied. During the process of resource preemption, lowest priority task which is taking the least amount of resources are always first migrated until the sum of resources is enough for the new task. If it still fails to meet the new task's requirements when all the tasks with lowest priority are migrated, it is time for middle-priority tasks to be migrated. Unless the sum of free resources and resources possessed by lowest and middle priority tasks is insufficient, the new task is finally put on a new general PM.

Algorithm 3 specifies the process of task migration on specific PM according to the identifier which indicates how preemption occurred and the form and scale of migration. If the total amount of resources occupied by the tasks with lowest priority meets the requirement of t_i, the tasks are migrated in ascending order of occupied VMs (line 3–9). Otherwise, the resource requirement of t_i is met by the migration of all non-critical tasks (line 13–21).

Algorithm 4 illustrates the process of determining whether to snatch resources and resources of which tasks are preempted in workflow scheduling. If the task t_i is on the critical path, it tries to preempt the resources of other tasks on non-critical paths (line 5–9). If t_i is not a critical task with a large computation, it tries to preempt the resources of the tasks with lowest priority (line 16–20). Algorithm 3 is used to perform migration of tasks on selected PM based on the determination (line 8 and line 19).

4.4 Global workflow scheduling

This section describes the global scheduling policy based on critical path priority, BFD and resource preemption. When workflow set arrives, critical paths of workflows are analyzed and priorities are set. Then, PMs with better match degree with tasks are obtained according to the PM utilization. Meanwhile, better PMs with higher speed will prioritize higher-priority tasks.

After the workflow set *WF* is submitted to datacenter, the subset *TP* with earliest start time in the waiting queue is obtained by above algorithm and taken as the object of a single processing, while the tasks in the waiting queue are represented by *UNP*. Cloud datacenter manager first sorts the tasks in *UNP* in ascending order of start time, and secondly chooses all tasks with earliest start time into a partition *TP*.

Algorithm 5: Global workflow scheduling (*WF*)

Input: The workflow set *WF*, PM list *S*

Output: The scheduling records of tasks in *WF*

1: Get high-performance PMs as *SH*, and general PMs as *SL*
2: Find the critical paths of every workflow in *WF* and set the priorities by **Algorithm 1**
3: Add all the source points into *UNP*
4: While *UNP*≠ ∅ **do**
5: Get all tasks with earliest start time in *UNP* as *TP*, and start time as *st*
6: Sort *TP* according to priority and requirment
7: *pf* =0
8: **For** *i*=0 to *TPsize* **do**
9: *pid*=Null
10: **Algorithm 2**(t_i, *SH*)
11: **if** *pid*= Null **then**
12: **Algorithm 4**(t_i, *SH*)
13: **end if**
14. **if** *pid*= *Null* **then**
15: **Algorithm 2**(t_i, *SL*)
16: **end if**
17: **if** *pid*= Null **then**
18: Wake up new PM for t_i and add it to *SL*
19: **end if**
20: Remove t_i from *UNP*
21: Update *Allor* of s_p and Schr of t_i
22: Set the start time of all successors and add them into *UNP* if they are ready
23: **end for**
24: end while

Then *TP* is split into three sub sets in descending order by priority, $TP' = \{TP_1,\ TP_2,\ TP_3\}$, i.e. *TP* is divided into three subsets of key path tasks, tasks picked by a, and other tasks. According to BFD, the tasks in each slice are first sort in descending order of required resources, so that the algorithm can schedule tasks in *TP* on the basis combination of precedence and start time. Tasks with earlier start time are always handled before later ones, while tasks with higher priority are scheduled before others while they are occupied same start time under the premise of same start time.

Algorithm 1 first find the critical path between the source point and sink point of each workflow and identifies all the tasks on the path by configuring highest priority, while tasks with larger computation selected by the parameter a are assigned the middle priority. In addition, all tasks ready for execution are added into the queue *UNP* which is transformed into a union of multiple task sets split by their start time in chronological order. Then Algorithm 2 is applied to process task scheduling of subset *TP* with the earliest start time according to the priority given by Algorithm 1, i.e. the PM where the remaining VM is closest to the demand and greater than or equal to the demand is selected when assigning each task, and tasks with high priority are first processed. Besides, while choosing PMs, tasks have the right to seize resources in the light of Algorithm 4 if the resources satisfy the requirements for the task.

Algorithm 5 specifies the global scheduling method, where Algorithm 1 is used to find the critical paths of workflows and define the priority (line 2), and Algorithm 2 is firstly used to select resources in the high-performance PM set and handle tasks on the basis of BFD strategy (line 10, line 15). If the resources of high performance not quite meet the needs, attempts are made to preempt the resource using algorithm 4 (line 12).

5 Experiments and analysis

In this section, extensive experimental evaluations are promoted to demonstrate the performance of the proposed dynamic scheduling method for concurrent workflows. The experimental environment and parameter settings are elaborated in detail in Sects. 5.1, and 5.2 gives the performance evaluation and analysis based on experimental results.

5.1 Experiment environment and parameter setting

To estimate the merits of a workflow scheduling method, measure of its performance should be built on some sample workflows, which are generated using a random workflow generator as the tool and has diverse properties [36], e.g. workflow start time, the number of tasks, etc. There is no doubt that an experiment on realistic workflows is more convincing, whereas there is no such a broad library of resources open to us [37]. For this reason, various workflows are created as the sample to support the experiment. Each workflow is initialized with start time, number of tasks and dependencies between tasks while each task is initialized with required amount of resources, amount of calculation and other attributes.

In this paper, time overhead model and resource utilization are used to estimate the performance of *DSM*, which appraise the makespan of the workflow set and the resource usage of occupied PMs during the execution of entire workflow set.

In the simulation experiments, we establish a datacenter in cloud environment which comprises of 2 kinds of PM with different operation speed of CPU shown in Table 4. Driven by Amazon EC2 instances of [38], a miniature Elastic Compute Unit can be regarded as a virtual machine. Therefore, the capacity of a PM is equivalent to the VMs that the resources can be encapsulated [39], so the capacities of the 2 kind of PM in the simulation platform are set to 8, i.e. there are 8 VMs in a PM that divides the CPU speed of the PM equally. The processing speeds of the 2 kinds of PMs are 40 MIPS and 24 MIPS respectively.

High-performance resources are always in short supply in the cloud. As a matter of fact, the performance of the entire algorithm varies with the number of predominant PMs available in the processing of applications [40]. So evaluations in four different environments are carried out, which owns 50, 100, 150 and 200 PMs of higher processing speed, respec-

Table 4 Experimental context

Hardware	CPU Speed (MIPS)
HP ProLiant ML110 G5	24
HP ProLiant BL460c G6	40

Table 5 Parameter settings

Parameter	Range
Size of workflow set	{50,100,150,200,250,300,350,400}
Size of each workflow	[5,20]
Numbers of VMs required by each Task	[1,5]
Calculation amount of each task (million instructions)	[0.1,5.0]

tively. Moreover, the number of general PMs employed is variable according to the actual handling of applications.

Accordingly, the parameters and their ranges of the dataset used in this paper are specified as Table 5. There are 8 different scales of dataset, and a workflow is 1 3-tuple while a task is a 5-tuple as defined in Sect. 3. A task generated can only get its start time upon the finish of all its precursors and its finish time depends on its calculation amount and the speed of VMs it selected.

5.2 Performance evaluation and analysis

In this section, evaluations on workflow time overhead and resource utilization of datacenter are implemented to evaluate the performance of *DSM* in cloud environment according to the results of simulation experimental.

Implemented on 8 datasets, *DSM* and other two algorithms are compared to estimate their performance. One of the two algorithms is a traditional algorithm–*Greedy* Algorithm which uses BFD to select PMs for tasks under the restriction conditions of optimum resource utilization. The other algorithm refers to the scheduling method based on hybrid improved Cuckoo search (*ICS*) algorithm and decision tree [14]. *ICS* minimizes data dependency by splitting workflows and then uses the decision tree to choose resources to meet QoS constraints of tasks. The three algorithms are described briefly as follows:

5.2.1 Evaluation on number of employed PMs

Before resource utilization, the number of employed PMs directly reflects the performance discrepancy between the three algorithms while dealing with applications, where the time overhead and resource utilization are first given expression. As a result, we investigate circumstance of occupying PMs with the above three algorithms on datasets of different scales. Figure 5 shows the observation results, where it can be found that the proposed *DSM* occupies less PMs compared with *Greedy* algorithm and *ICS* algorithm. The reason for this result is that *DSM* selects PM for tasks in a utilization aware way, and adjusts the scheduling results in real time to close down the low load PMs during the scheduling process. Therefore, *DSM* employed less PMs than the other two algorithms.

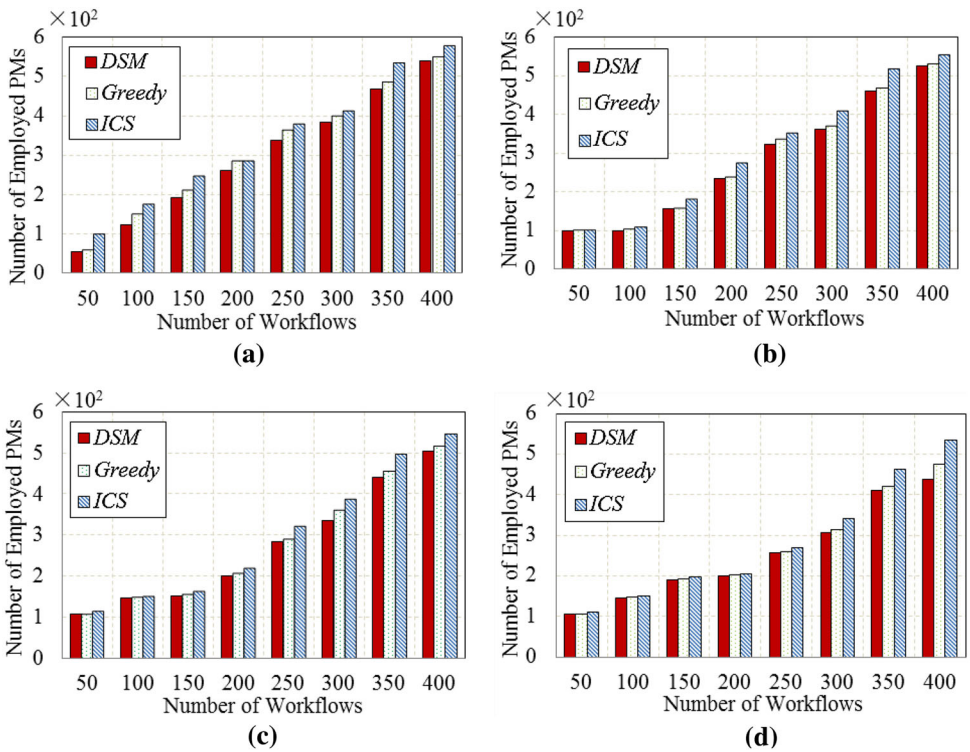


Fig. 5 Comparison of Number of PMs Employed by Different Algorithms on 8 Datasets. **a** Number of high-performance PMs =50, **b** number of high-performance PMs =100, **c** number of high-performance PMs =150, **d** number of high-performance PMs=200

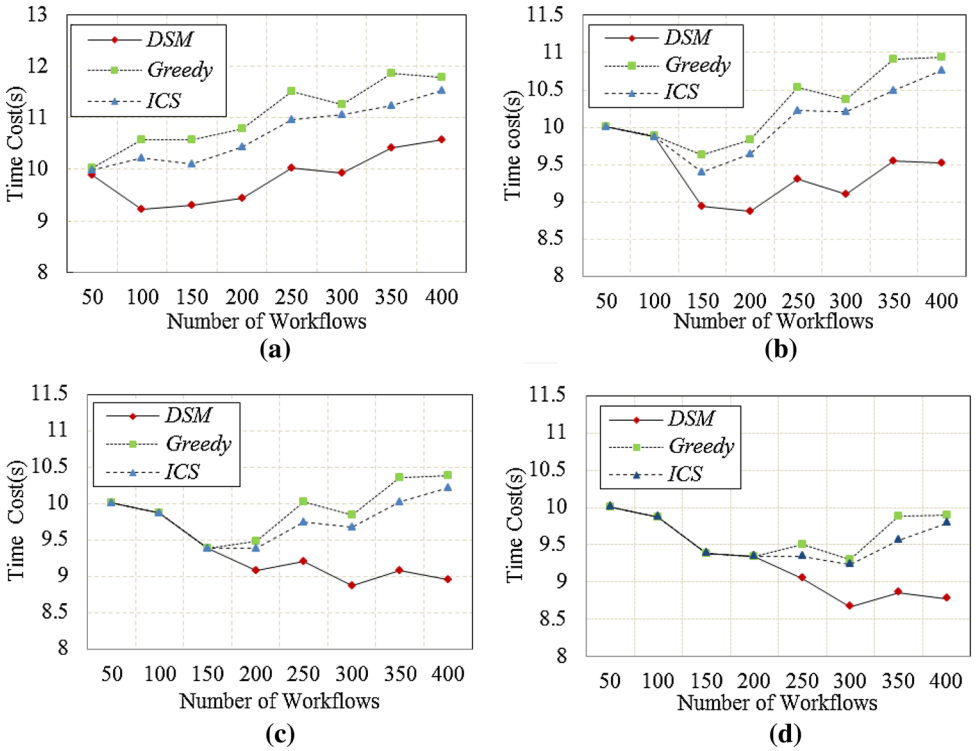


Fig. 6 Comparison of Time Overhead with Different Algorithms on 8 Datasets. **a** Number of high-performance PMs =50, **b** number of high-performance PMs =100, **c** number of high-performance PMs =150, **d** number of high-performance PMs =200

The number of employed PMs can only show the prediction results of performance cursorily, for concrete experiments are needed to get the utilization and time cost.

5.2.2 Evaluation on time cost

To evaluate performance more accurately, the time cost of three algorithms processing different datasets under three different conditions is achieved through experiments.

The 4 figures in Fig. 6 illuminate the results of simulation applying the three algorithms to deal with 8 different datasets in the case where the number of high speed PMs in the datacenter is 50,100,150, 200 respectively.

By comparing the three graphs, we find that the performance differences between the three algorithms can be affected as the number of servers with high computational speeds increases. For example, in Fig. 6d, there are 200 high speed PMs available for execution in the datacenter, the time costs of the three algorithms are almost the same while handling the datasets owns 50,100, 150 and 200 workflows. This is because better PMs are sufficient in dealing with a small amount of workflow, and all the tasks in the dataset get efficiently execution, so the performance differences of the algorithms cannot be reflected. But when the scale of the dataset is larger than high-speed PMs, *DSM* reduces the time cost greatly, compared with the other two algorithms under the same configuration. In combination with the 3 figures in Fig. 6, a conclusion can be drawn that *DSM* has some advantages in reducing execution time. The time advantage of DSM benefits from the scheduling strategy that tasks on the critical path are endowed with higher priority and always occupies resources with high-speed resources. This strategy greatly reduces the execution time of critical paths, as a result, the makespan of the workflows are substantially decreased.

5.2.3 Evaluation on resource utilization

In addition to time cost, resource utilization is another important indicator presented to evaluate the performance of algorithm in this paper. The average resource utilization of PMs employed in the three algorithms while processing different datasets under four different conditions is shown as Fig. 7.

The four figures in Fig. 7 illuminate the results of simulation applying the four algorithms to deal with 8 different datasets. From the results of performance evaluation on resource utilization, it can be clearly seen from the figure that *DSM* for workflow scheduling has the leading advantage in terms of resource utilization relative to the other two algorithms under same configuration.

However, since high-speed resources become more numerous, the difference between *DSM* and *Greedy* algorithm and *ICS* algorithm getting obviously smaller. This is because the high-speed VMs are sufficient, and the resources datacenter selected for applications are mostly the same in the

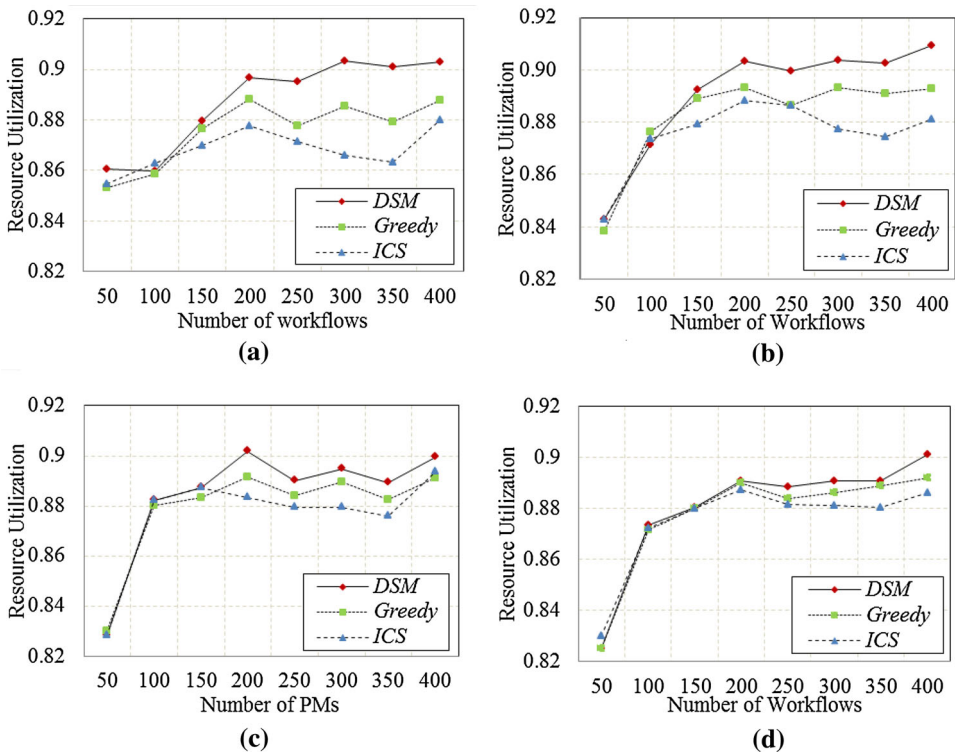


Fig. 7 Comparison of Resource utilization with Different Algorithms on 8 Datasets. **a** Number of high-performance PMs =50, **b** number of high-performance PMs =100, **c** number of high-performance PMs =150, **d** number of high-performance PMs =200

task scheduling process, and the superiority of the algorithm gradually disappears.

By comparing the results of the simulation experiment thoroughly, it can be found that, *DSM* proposed in this paper has certain superiority and practical significance compared with two other algorithms—*Greedy* algorithm and *ICS* algorithm. Because *DSM* chooses PMs with the most suitable capacity for the tasks every time and the tasks are concentrated on active PMs as much as possible while the task set is processed, accordingly, the PM selected by the algorithm is used efficiently. The experimental results show that the *DSM* algorithm proposed in this paper has some practical significance in practice.

6 Conclusion and future work

In this paper, a dynamic scheduling method for concurrent workflows, named as *DSM*, has been proposed based on critical path lookup, time and resource utilization aware task scheduling and priority driven resource preemption. To elaborate, a time overhead model has been put forward for workflow applications while a resource utilization model for cloud datacenters has been raised. Furthermore, a dynamic scheduling method for concurrent workflows has been proposed, which minimizes the makespan of workflow through critical path lookup, and maximizes the utilization through time and utilization aware task scheduling and priority driven resource preemption. At last, extensive simulation experiments have been implemented and the results have demonstrated the efficiency and effectiveness of our proposed method.

Based on the research advanced in this paper, we intend to extend *DSM* and adapt it to actual cloud environment in the future. The transmission time overhead of workflow is a necessary factor in real cloud environment. In addition, according to the different amounts of different resources configured, PMs can be divided into different types. Hence the algorithm needs adjustment to select the appropriate type of PM for tasks according to their characteristics. Transmission time and the PM types will be considered in future research to optimize use of resources and improve users' experience. And we will test obtained theoretical results in the actual cloud platform to verify the effectiveness and superiority of the algorithm.

Acknowledgements This research is supported by the National Science Foundation of China under Grant Nos. 61702277, 61672276, 61402167 and 61672290. Besides, this work is also supported by The Startup Foundation for Introducing Talent of NUIST, the open project from State Key Laboratory for Novel Software Technology, Nanjing University under grant no. KFKT2017B04, the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) fund, Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology (CICAEET), the project "Six Talent Peaks Project in Jiangsu Province" under grant no. XYDXXJS-040, and Innovation Platform Open Foundation of Hunan Provincial Education Department (No. 17K033).

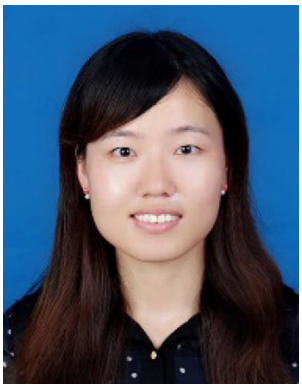
References

1. Netjinda, N., Sirinaovakul, B., Achalakul, T.: Cost optimal scheduling in IaaS for dependent workload with particle swarm optimization. J. Supercomput. **68**(3), 1579–1603 (2014)
2. Rao, J., Zhou, X.: Towards fair and efficient SMP virtual machine scheduling. In: ACM Sigplan Symposium on Principles and Practice of Parallel Programming. vol. 49(8), pp. 273–286 (2014)
3. Wang, P., Huang, Y., Li, K., Guo, Y.: Load balancing degree first algorithm on phase space for cloud computing cluster. J. Comput. Res. Dev. **51**(5), 1095–1107 (2014)
4. Armbrust, M., Fox, A., Griffith, R., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Eecs Department University of California Berkeley, vol. 53(4), pp. 50–58 (2009)
5. Zhang, S., Qian, Z., Wu, J., Lu, S., Epstein, L.: Virtual network embedding with opportunistic resource sharing. IEEE Trans. Parallel Distrib. Syst. **25**(3), 816–827 (2014)
6. Li, W., Zhang, Q., Wu, J., Li, J., Hao, H.: Trust-driven and QoS demand clustering analysis based cloud workflow scheduling strategies. Clust. Comput. **17**(3), 1–18 (2014)
7. Dou, W., Xu, X., Meng, S., Zhang, X., Hu, C., Yu, S., Yang, J.: An energy-aware virtual machine scheduling method for service QoS enhancement in clouds over big data. Concurr. Comput. Pract. Exp. **29**(14), 1–20 (2017)
8. Shen, H., Li, X.: Algorithm for the cloud service workflow scheduling with setup time and deadline constraints. J. Commun. **36**(6), 183–192 (2015)
9. Guo, H., Chen, Z., Yu, Y., Wang, Y., Chen, X.: A communication aware DAG workflow cost optimization model and algorithm. J. Comput. Res. Dev. **52**(6), 1400–1408 (2015)
10. Chen, W., Lee, Y.C., Fekete, A., Zomaya, A.Y.: Adaptive multiple-workflow scheduling with task rearrangement. J. Supercomput. **71**(4), 1297–1317 (2015)
11. Dong, J., WANG, H., Cheng, S.: Energy-performance tradeoffs in laaS cloud with virtual machine scheduling. China Commun. **12**(2), 155–166 (2015)
12. Durao, F., Carvalho, J.F.S., Fonseka, A., Garcia, V.C.: A systematic review on cloud computing. J. Supercomput. **68**(3), 1321–1346 (2014)
13. Ahmad, S.G., Liew, C.S., Rafique, M.M., Munir, E.U., Khan, S.U.: Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems. In: IEEE Fourth International Conference on Big Data and Cloud Computing pp. 129–136 (2015)
14. Chen, C.: Workflow task scheduling in cloud computing based on hybrid improved CS algorithm and decision tree. J. Univ. Electron. Sci. Technol. China **45**(6), 974–980 (2016)
15. Liu, J.X., Yang, X.F., X. Y.: Cloud workflow scheduling method based on batch processing strategy. Comput. Integr. Manufac. Syst. **21**(2), 336–343 (2015)
16. Luo, Z., Wang, P., You, B., Jie, S.U.: Optimization scheduling of workflow's accuracy based on reverse reduction under constraint time. J. Beijing Univ. Posts Telecommun. **40**(1), 99–104 (2017)
17. Doulamis, N.D., Kokkino, P., Varvarigos, E.: Resource selection for tasks with time requirements using spectral clustering. IEEE Trans. Comput. **63**(2), 461–474 (2014)
18. Kong, Y., Zhang, M., Ye, D.: A belief propagation-based method for task allocation in open and dynamic cloud environments. Knowl. Based Syst. **115**, 123–132 (2016)
19. Xing, G., Xu, X., Xiang, H., Xue, S., Ji, S., Yang, J.: Fair energy-efficient virtual machine scheduling for internet of things applications in cloud environment. Int. J. Distrib. Sensor Netw. **13**(2), 1–11 (2017)
20. Hao, L., Cui, G., Qu, M., Ke, W.: Resource scheduling optimization algorithm of energy consumption for cloud computing based on task tolerance. J. Softw. **9**(4), 895–901 (2014)
21. Cao, F., Zhu, M.M., Wu, C.Q.: Energy-efficient resource management for scientific workflows in clouds. In: 2014 IEEE World Congress on Services (SERVICES), pp. 402–409 (2014)
22. Jrad, F., Tao, J., Brandic, I., Streit, A.: SLA enactment for large-scale healthcare workflows on multi-Cloud. Future Gener. Comput. Syst. **43**(4), 135–148 (2015)
23. Wang, Y., Wang, J., Han, Y.: A two-stage resource scheduling method for workflow cloud computing system. J. Sourth China Univ. Technol. **45**(1), 80–87 (2017)
24. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**(2), 222–235 (2014)
25. Luo, Z.Y., Wang, P., You, B., Zhu, X.S.: Serial reduction optimization research of complex product workflow's accuracy under the tine constraint. Adv. Mech. Eng. **8**(10), 1–9 (2016)
26. Ahmed, W., Wu, Y.: Estimation of cloud node acquisition. Tsinghua Sci. Technol. **19**(1), 1–12 (2014)
27. Cao, B., Wang, X., Xiong, L., Fan, J.: Searching method for partical swarm optimization of cloud workflow scheduling with time constraint. Comput. Integr. Manufac. Syst. **22**(2), 372–380 (2016)
28. Xu, Y., Li, K., Hu, J., Li, K.: A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf. Sci. **270**(6), 255–287 (2014)
29. Chen, H., Zhu, J., Manho, M.A., Zhu, X.: Scheduling for stochastic tasks and resources in virtualized clouds. Syst. Eng. Electron. **39**(2), 348–354 (2017)
30. Wu, C.M., Chang, R.S., Chan, H.Y.: A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters. Future Gener. Comput. Syst. **37**(7), 141–147 (2014)
31. Calheiros, R.N., Buyya, R.: Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans. Parallel Distrib. Syst. **25**(7), 1787–1796 (2014)
32. Guo, Y.Q., Song, J.X.: Optimal task-level scheduling based on multimedia applications in cloud. Comput. Sci. **42**(11), 413–416 (2015)
33. Mandal, Vasundhara, D., Kar, R., Ghoshal, S.P.: Digital FIR filter design using fitness based hybrid adaptive differential evolution with particle swarm optimization. Nat. Comput. **13**(1), 55–64 (2014)
34. Prasad, A.S., Rao, S.: A mechanism design approach to resource procurement in cloud computing. IEEE Trans. Comput. **63**(1), 17–30 (2014)
35. Chen, H.K., Zhu, J.H., Zhu, X.M., Ma, M.H., Zhang, Z.S.: Resource-delay-aware scheduling for real-time tasks in clouds. J. Comput. Res. Dev. **54**(2), 446–456 (2017)
36. Xu, X., Dou, W., Zhang, X., Chen, J.: EnReal: an energy-aware resource allocation method for scientific workflow executions in cloud environment. IEEE Trans. Cloud Comput. **4**, 166–179 (2016)
37. Abrishami, S., Naghibzadeh, M., Epema, D.H.J.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
38. Yang, G., Stolyar, A.L., Walid, A.: Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud. IEEE Infocom **12**(11), 620–628 (2013)
39. Li, X., Wu, J., Tang, S., Lu, S.: Let's stay together: towards traffic aware virtual machine placement in data centers. In: 2014 IEEE Conference on Computer Communications (INFOCOM), pp. 1842–1850 (2014)
40. Xiaohu, W., Loiseau, P.: Algorithms for scheduling deadline-sensitive malleable tasks. In: 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 530–537 (2015)

Shengjun Xue received his Ph.D. degree in Wuhan University of Technology, China, in 2002. From 2007 to 2008, he did his postdoctoral research in the Department of Computer Science and Technology, Purdue University, USA. Now, he is a full professor of Jiangsu Engineering Center of Network Monitoring of Nanjing University of Information Science & Technology, China. Up to now, he has chaired four national projects and published more than 90 research papers in international journals and international conferences. His research interests include Computer networks, computer supported cooperative work (CSCW) and high performance computing research.

Yue Peng is currently studying as a Master Candidate in Computer Science, Nanjing University of Information Science & Technology, China. She received the Bachelor's degree in Network Engineering in 2015 from Nanjing University of Information Science & Technology. Her research interests include cloud computing, green computing and big data.

Xiaolong Xu received the doctor's degree in computer science and technology in 2016 from Nanjing University. He is currently an assistant professor in the School of computer and software, Nanjing University of Information Science & Technology, China. He has published more than 20 research papers in international journals and conferences. His research interests include cloud computing, green computing, and big data.



Jie Zhang is working towards her Ph. D. degree in the department of computer science, Nanjing University. She also serves as an engineer of Shanghai Meteorological Bureau. She receives her Master degree and Bachelor degree both from Nanjing University of Information Science & Technology. Her research interests include Cloud Computing, Big Data and Workflow Scheduling.

Chao Shen is a lecturer in the School of computer and software, Nanjing University of Information Science & Technology, China. She has published several research papers in international journals and conferences. Her research interests include meteorological information technology and big data.

Feng Ruan is a lecturer at Nanjing University of Information Science & Technology. He received the B.S. and M.S. degree in the Nanjing University of Information Science & Technology. Now he is PHD student in Computer Science Department of Nanjing University of Information Science & Technology. His research interests mainly include complex system, computer network, routing protocol and algorithm design, data fusion, and cloud computing.