Parallel Computing 39 (2013) 177–188

Contents lists available at SciVerse ScienceDirect

Parallel Computing

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

Cost-efficient task scheduling for executing large programs in the cloud

Sen Su [a,*], Jian Li [a], Qingjia Huang [a], Xiao Huang [a], Kai Shuang [a], Jie Wang [b]

[a] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, 10 Xi Tu Cheng Road, Beijing, China
[b] Department of Computer Science, University of Massachusetts, Lowell, MA 01854, USA

ARTICLE INFO

Article history:
Received 14 February 2012
Received in revised form 25 February 2013
Accepted 5 March 2013
Available online 15 March 2013

Keywords:
Cloud computing
Cost-efficient scheduling
Parallel task scheduling
Directed acyclic graph

ABSTRACT

Executing a large program using clouds is a promising approach, as this class of programs may be decomposed into multiple sequences of tasks that can be executed on multiple virtual machines (VMs) in a cloud. Such sequences of tasks can be represented as a directed acyclic graph (DAG), where nodes are tasks and edges are precedence constraints between tasks. Cloud users pay for what their programs actually use according to the pricing models of the cloud providers. Early task scheduling algorithms are focused on minimizing makespan, without mechanisms to reduce the monetary cost incurred in the setting of clouds. We present a cost-efficient task-scheduling algorithm using two heuristic strategies.The first strategy dynamically maps tasks to the most cost-efficient VMs based on the concept of Pareto dominance. The second strategy, a complement to the first strategy, reduces the monetary costs of non-critical tasks. We carry out extensive numerical experiments on large DAGs generated at random as well as on real applications. The simulation results show that our algorithm can substantially reduce monetary costs while producing makespan as good as the best known task-scheduling algorithm can provide.

1. Introduction

Cloud computing offers a promising new platform to execute large programs (see, e.g., the Pregel project [1] and the Nephele project [2]). In addition to providing multiple VMs to execute tasks contained in a program, cloud computing also offers on-demand scaling and pay-per-use metered service. That is, computing resources are dynamically allocated to user programs (or taken away) based on needs, and users just pay for the resources their programs actually consume [3,4], similar to the conventional pay-per-use metered service for utility consumptions of water, electricity, and natural gas.

Execution of a program can be viewed as executing multiple tasks contained in it, represented by a DAG [5]. Independent tasks in a DAG can be executed by multiple VMs simultaneously, while related tasks will need to be executed in the correct sequential orders. It is NP-Complete to schedule tasks to obtain the minimum makespan [6], and so one can only hope for near-optimal task scheduling without performing exhaustive search. Consequently, much of the prior work (see, e.g., [7–13]) was dedicated to finding efficient task-scheduling mechanisms for producing near-minimum makespan, without worrying about monetary charges of using computing resources. In the setting of cloud computing, however, we would need to consider the pay-per-use metered service model to balance efficiency and monetary costs.

* Corresponding author.
E-mail addresses: susen@bupt.edu.cn (S. Su), upton@bupt.edu.cn (J. Li), QJHuang@bupt.edu.cn (Q. Huang), HuangXiao@bupt.edu.cn (X. Huang), shuangk@bupt.edu.cn (K. Shuang), wang@cs.uml.edu (J. Wang).

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.03.002

Different cloud providers may offer different types of pay-per-use metered service. For example, Amazon EC2 [14] charges users based on the number and the type of VMs their programs use and how long they use them. Google AppEngine [15], on the other hand, charges users based on the number of CPU cycles actually consumed by their programs. Cloud providers also offer multiple types of VMs with different capacities and pricing. For example, Amazon EC2 offers 11 VM instances with different computing capacities, OS types, and pricing [14]. Thus, different task-scheduling plans of the same DAG using different VMs in different clouds may result in different makespan and different monetary costs. We would therefore want a cost-efficient task-scheduling algorithm for dispatching tasks to provide good performance while reducing monetary costs.

We investigated in a preliminary study [16] how to schedule tasks in a large DAG to reduce monetary costs. We devised a cost-conscious scheduling algorithm in a cloud setting with only one VM type and one pricing model. We showed that our algorithm provides a good balance between efficiency and monetary costs. However, this algorithm is inapplicable to multiple types of VMs with different pricing models.

Minimizing makespan and minimizing monetary costs are competing objectives. Thus, we formulate an optimization problem of task scheduling with multiple VMs and different pricing models as a convex combination of minimizing makespan and monetary costs. Built on previous results of DAGMan/Pegasus [17], Dryad [5], Nephele [2], and others, we devise a scheduling algorithm to better match the pricing model and the opaque nature of the cloud. The key component of our algorithm is two task-scheduling heuristics. The first heuristic uses the concept of Pareto dominance [18] to generate a cost-efficient schedule based on the execution time of the tasks and the monetary charges of the VMs. The second heuristic complements the first heuristic–it attempts to minimize the monetary costs of non-critical tasks by extending their execution time. We carry out numerical simulations on randomly generated large DAGs as well as on real applications. Simulation results show that our algorithm can substantially reduce monetary costs while producing makespan as good as the best known task-scheduling algorithm can provide.

The rest of the paper is organized as follows. We briefly survey related research in Section 2. We formulate the task-scheduling problem in Section 3, and describe our scheduling algorithm in Section 4. In Section 5 we present numerical results obtained from simulations and we conclude the paper in Section 6.

2. Related work

Existing research on task-scheduling for DAGs can be classified into the following three categories:

1. System development. A variety of systems have been developed in recent years to improve performance. For example, DAGMan/Peagus is a system that maps workflows onto grids [17]. Dryad is a parallel and distributed application developed by Microsoft for DAG-based task processing [5], where the scheduler of Dryad is designed to optimize the throughput of the overall distributed computing clusters with locality constraints. Nephele is a system designed to minimize the number of allocated instances in the cloud for optimizing data locality [2].
2. Scheduling heuristics. Because task scheduling for DAGs is NP-hard, a number of heuristics were devised in the literature. Evaluations and comparisons of different task-scheduling algorithms were surveyed in [19]. These heuristics can be further classified into four groups: (1) *List Scheduling Algorithm* (e.g., DLS [20] and HEFT [9]); (2) *Task Duplication-based Scheduling Algorithm* (e.g., TANH [10] and DBUS [21]); (3) *Clustering Algorithm* (e.g., DSC [22]); and (4) *Guided Random Search Algorithm* [7] (e.g., genetic algorithms). It was shown [23] that for task scheduling, HEFT outperforms the other scheduling heuristics. All these algorithms intend to minimize makespan, and none of them deals with the problem of dispatching large tasks in a cloud setting, or considers monetary costs.
3. Multi-objective workflow scheduling in a heterogeneous resource environment. Wieczorek et al. provided a multi-criteria taxonomy of the workflow scheduling algorithms and selected the best schedule according to the user requirements [19]. Kurowski et al. investigated prediction mechanisms and resource reservation for multi-objective task scheduling in Grids [24,25]. Yu and Buyya partitioned the workflows and used genetic algorithms to meet time and budgetary constraints [26,27]. Sakellariou and Zhao devised two scheduling algorithms that dispatch parallel tasks based on the backtracking technology to meet budgetary constraints of the users [28]. Singh et al. presented a slot-based provisioning model on grids to provide scheduling according to the availability and cost of resources [29].

None of these early results, however, is suitable for handling multiple tasks in the cloud environment, where resources may be unbounded and there is no wait time to get the needed resources. We note that Zhu et al. studied automated and dynamic scheduling of adaptive applications with a fixed time bound and monetary constraints [30]. In these applications, there exists application-specific flexibility with respect to the scheduling performed. For example, the data arrival rate and workload structure can be changed dynamically during runtime. Garg et al. scheduled parallel applications on Utility Grids to manage the tradeoff between running time and monetary constraints [31]. We differ from the above work in that we give a clear formulation of two pricing models for the cloud resources and devise a cost-efficient scheduling heuristic to balance between makespan and monetary costs. Furthermore, we also use the idea of minimizing the monetary costs by reclaiming slack times for non-critical tasks.

3. Problem formulation

3.1. Application model

We follow the terminologies used in Microsoft Dryad [5]. Let $G = (V, E)$ be a DAG, where V is the set of v tasks to be executed and E is the set of e edges representing the precedence constraints between tasks. Without loss of generality, we assume that G has a task without any predecessors. We call it an *entry task* and denote it by v_{entry}. We also assume that G has a task without any successors. We call it an *exit task* and denote it by v_{exit}. If there is more than one exit task, we will connect them to a pseudo exit task with zero execution time and zero cost on the incoming edges connected to it. Likewise, if there is more than one entry task, we will connect a zero-cost (both node and edges) pseudo entry task to all of them. Adding the pseudo exist task or the pseudo entry task does not impact scheduling. Among the predecessors of a task v_i, the predecessor that completes the processing at the latest time is called the *most influential parent* (MIP) of the task, denoted by MIP_i. We call the longest path of a DAG a *critical path* (CP).

The weight of a node v_i, denoted by dt_{v_i}, represents the computation load for task v_i. We assume that the cloud computing environment consists of a set of m fully connected heterogeneous VMs, denoted by M. Let ca_{m_j} denote the CPU cycles allocated to VM m_j. Similar to the capped mode of the XEN scheduler, the CPU cycle for a particular VM cannot be changed at runtime. Each task can be executed on a different VM, and so we denote by $t(v_i, m_j)$ the execution time of the task v_i on VM m_j. We have

$$t(v_i, m_j) = \frac{dt_{v_i}}{ca_{m_j}}. \tag{1}$$

In our model, task executions are assumed to be non-preemptive. In addition, the average execution time of a task v_i is given by

$$\bar{T}_{v_i} = \sum_{j=1}^{m} \frac{t(v_i, m_j)}{m}. \tag{2}$$

The weight of edge (v_i, v_k), denoted by ct_{v_i,v_k}, represents the communication time between task v_i and task v_k. If both v_i and v_k reside in the same VM, then ct_{v_i,v_k} is zero. The priority of task v_i is computed by traversing the DAG upward, starting from the exit task, and is defined by

$$p_{v_i} = \bar{T}_{v_i} + \max_{v_k \in succ(v_i)} (ct_{v_i,v_k} + p_{v_k}), \tag{3}$$

where $succ(v_i)$ is the set of successors of task v_i. The value of p_{v_k} is the priority of immediate successors of task v_i. Since the priority is computed by traversing the task graph upward, the priority of exit task is equal to

$$p_{v_{exit}} = \bar{T}_{v_{exit}}. \tag{4}$$

Fig. 1 shows a simple task DAG, with details listed in Table 1. The values presented in the table are the values of priority and the different execution time of the tasks on three VMs.

Let $EST(v_i, m_j)$, $EFT(v_i, m_j)$, and $LFT(v_i, m_j)$ denote, respectively, the earliest execution start time, the earliest finish time, and the latest finish time of a task v_i on a VM m_j. For the entry task v_{entry}, we have

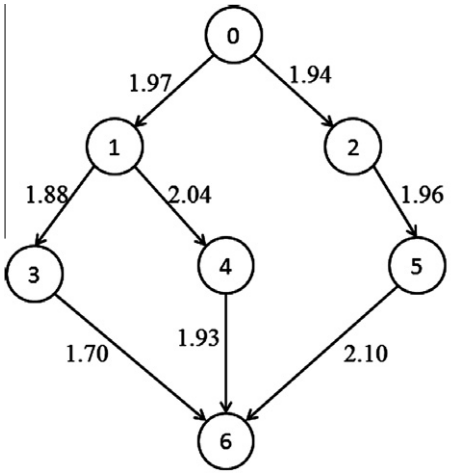


Fig. 1. A simple task graph.

Table 1
Execution time and priority of tasks.

Task	VM1	VM2	VM3	Priority
0	9.26	11.01	13.79	51.95
1	9.2	10.5	13.9	38.47
2	10.05	12.12	13.15	38.66
3	9.54	10.41	15.4	24.81
4	10.33	11.51	14.08	25.23
5	9.36	11.1	14.03	24.93
6	9.39	10.73	13.88	11.33

$$EST(v_{entry}, m_j) = 0. \tag{5}$$

For the other tasks in the graph, we compute EST, EFT, and LFT recursively as follows:

$$EST(v_i, m_j) = \max\left\{time(j), \max_{v_l \in pred(v_i)}(AFT(v_l) + ct_{v_l, v_i})\right\}, \tag{6}$$

$$EFT(v_i, m_j) = t(v_i, m_j) + EST(v_i, m_j), \tag{7}$$

$$LFT(v_i, m_j) = \min_{v_k \in succ(v_i)}(AST(v_k) - ct_{v_i, v_k}), \tag{8}$$

where m_j is the VM on which the task v_i is scheduled to run, and $pred(v_i)$ is the set of immediate predecessor tasks of task v_i, and $time(j)$ is the time when the VM m_j is ready for task execution, and $succ(v_i)$ is the set of immediate successor tasks v_i. The inner max block in the EST equation returns the ready time. The actual start and finish time of task v_i on VM m_j, denoted by $AST(v_i, m_j)$ and $AFT(v_i, m_j)$, respectively, may be different from its earliest start time $EST(v_i, m_j)$ and finish time $EFT(v_i, m_j)$. The makespan is equal to the actual finish time of the exit task v_{exit} and is defined by

$$M = AFT(v_{exit}). \tag{9}$$

The makespan is also referred to as the running time for the entire DAG.

3.2. Cloud resources model

Cloud providers offer different types of VMs for different types of workload. These VMs have different processing capacities and pricing models.

We use the Google pricing model in this study. Similar to [30], we assume a fine-grained pricing model based on the actual allocation of CPU cycles. In other words, a VM with more processing power is associated with a higher monetary cost, and vice versa. In particular, we use a linear pricing model and an exponential pricing model. In the linear pricing model, the cost of using a VM is linearly correlated with the number of CPU cycles. Let ca_{slow} denote the CPU cycle for the slowest VM m_{slow}. If Vc_{base} is the base price charged to m_{slow}, then the cost incurred to execute task v_i on VM m_j can be calculated as follows:

$$c(v_i, m_j) = \sigma \times t(v_i, m_j) \times Vc_{base} \times \left(\frac{ca_{m_j}}{ca_{m_{slow}}}\right), \tag{10}$$

where σ is a random variable used to generate different combinations of VM pricing and capacity. In the exponential pricing model, the cost of VM allocation is calculated as follows:

$$c(v_i, m_j) = \sigma \times t(v_i, m_j) \times Vc_{base} \times \exp^{\frac{ca_{m_j}}{ca_{m_{slow}}}}. \tag{11}$$

The total monetary cost is computed by

$$C = \sum_{j \in select} c(v_i, m_j). \tag{12}$$

Other resources such as memory space and network bandwidth are not considered in the current model, which will be a topic for future investigations.

3.3. Scheduling objectives

Given a DAG of tasks, we would like to find a schedule of these tasks so that the monetary costs and the makespan are both minimized. These two objectives, however, present a competition, and so it is reasonable to consider minimization of a convex combination of these objectives.

4. Cost-efficient scheduling algorithm

We use the concept of Pareto dominance [18] to devise a cost-efficient scheduling algorithm to process multiple tasks in a DAG in the cloud setting.

4.1. Pareto optimality

Incorporating monetary cost into task scheduling adds a layer of complexity. Since we cannot directly compare different scheduling plans with competing objectives, we will use the concept of Pareto dominance to select VMs and carry out comparisons.

Definition 1. *Pareto dominance:* A vector $\vec{x} = (x_1, \ldots, x_n)$ is said to dominate a $\vec{y} = (y_1, \ldots, y_n)$ (denoted by $\vec{x} \prec \vec{y}$) if and only if $\vec{x}$ is partially less than $\vec{y}$; namely, if and only if $x_i \leqslant y_i$ for all $i \in \{1, \ldots k\}$ and $x_i < y_i$ for some $i \in \{1, \ldots k\}$.

Given two task scheduling plans s_1 and s_2, we say that s_1 dominates s_2 if the following two conditions are both true:

1. Condition (1): s_1 is not worse than s_2 in terms of running time and monetary costs; namely, either for all nodes v_i in the given DAG, we have $t(v_i, m_1) \leqslant t(v_i, m_2)$ and $c(v_i, m_1) \leqslant c(v_i, m_2)$; or for the makespan and total monetary costs, we have $M(s_1) \leqslant M(s_2)$ and $C(s_1) \leqslant C(s_2)$
2. Condition (2): s_1 is strictly better than s_2 either on the running time or on the monetary costs; namely, either for all nodes v_i in the given DAG, we have $c(v_i, m_1) < c(v_i, m_2)$ or $t(v_i, m_1) < t(v_i, m_2)$; or for the makespan and total monetary costs, we have $M(s_1) < M(s_2)$ and $C(s_1) < C(s_2)$

We use Fig. 2 as an example to illustrate this concept. Each point in Fig. 2 represents different combinations of monetary cost and execution time for the same task. It is easy to see that none of the resource provisioning solutions is optimal with respect to both objectives. The task scheduling solutions s_1 and s_2 do not dominate each other, and they are not dominated by any other solution either. The scheduling plan s_3 is dominated by s_1 and the scheduling plan s_4 is dominated by both s_1 and s_2. Motivated by the strategies used for solving the multi-criteria workflow scheduling problem [24], we formulate the following minimization problem with an objective function for each node $v_i \in V$ as a convex combination of running time and monetary cost:

$$Minimize: \alpha \times T(i,j) + (1-\alpha) \times C(i,j) \text{ for all } m_j \in M, \tag{13}$$

$$Subject to: T(i,j) = \frac{t(v_i, m_j) - t_{\min}}{t_{\max} - t_{\min}}, \tag{14}$$

$$C(i,j) = \frac{c(v_i, m_j) - c_{\min}}{c_{\max} - c_{\min}}, \tag{15}$$

$$\alpha \in [0,1], \tag{16}$$

where α is a cost-efficient factor that represents the user's preference for the execution time and the monetary cost; $T(i,j)$ and $C(i,j)$ represent cost-efficiency ratios of time and costs, respectively; $t_{min(max)}$ and $c_{min(max)}$ are, respectively, the minimum (maximum) execution time and the minimum (maximum) monetary cost of any task scheduling plan.

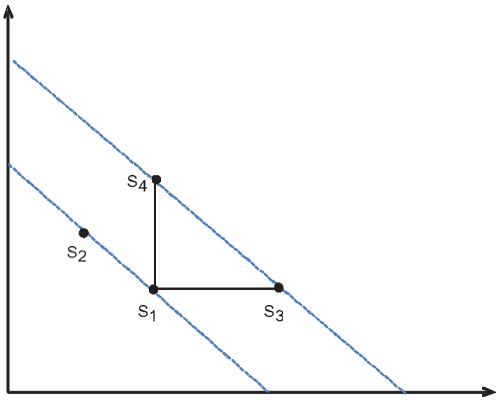


Fig. 2. Pareto dominance.

4.2. Pareto optimal scheduling heuristic (POSH)

We describe a heuristic to dispatch tasks in a DAG to the cost-conscious VMs based on Pareto dominance, and we call it Pareto Optimal Scheduling Heuristic (POSH). POSH is an extension of the Heterogeneous Earliest Finish Time (HEFT) heuristic. Developed for scheduling tasks in a DAG on heterogeneous dedicated multiprocessing systems, HEFT is better than other scheduling heuristics (see, e.g., [32,33]). HEFT assigns a priority to each task in the DAG and then maps the task with the highest priority to the VM that minimizes the earliest finish time. POSH uses both the running time and the monetary cost to modify the last step to map the task with the highest priority to the most cost-efficient VM based on Pareto dominance. POSH involves the following three phases:

(a) Weighting Phase: Assign the weights to the nodes and edges in the workflow. The weights assigned to nodes are calculated based on the predicted execution time of the tasks and the weights assigned to edges are calculated based on predicted time of the data transferred between the VMs.
(b) Prioritizing Phase: Create a sorted list of tasks organized in the order how they should be executed. The priority of each task is to be set with the upward priority value, which is equal to the weight of the node plus the execution time of the successors. The task list is generated by sorting the tasks by the descending order of priority.
(c) Mapping Phase: Assign the tasks to the resources based on Pareto dominance. Consecutive tasks are mapped to the resources based on the priority queue. For each task, choose the VM that favors scheduling tasks with low monetary cost to run it. This is done by the pre-defined objective function (see Section 3).

Algorithm 1: Pareto Optimal Scheduling Heuristic (POSH)

Input: $G(N, E)$: the DAG task graph
M: the set of VMs

```
1  Compute priority for all nodes v_i ∈ V by traversing graph upward;
2  Sort v_i in the descending order by its priority value;
3  for each v_i ∈ V do
4  |   for each m_i ∈ M do
5  |   |   Compute α × T(i, j) + (1 − α) × C(i, j);
6  |   end
7  end
8  for each task v_i in the ready queue do
9  |   Assign task v_i to the VM m_j that minimizes the objective function (Equation
   |   13) of task v_i;
10 end
```

The time complexity of POSH is the same as that of HEFT, which is equal to $O(em)$ for e edges and m VMs.

4.3. Slack time scheduling heuristic

POSH produces a scheduling plan with time slots for task executions and data communications on VMs. Fig. 3 shows the scheduling obtained by POSH on the input shown in Fig. 1.

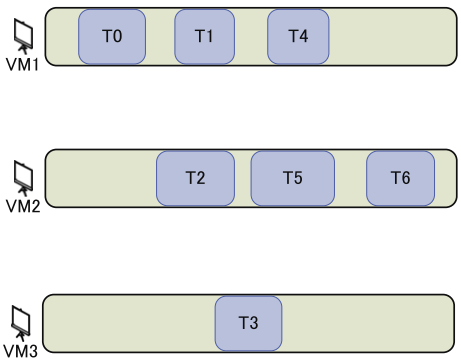


Fig. 3. Scheduling of tasks with POSH.

The Critical Path of this scheduling is $T0 \rightarrow T2 \rightarrow T5 \rightarrow T6$. We can see that non-critical tasks $T3$ and $T4$ have room to extend their execution time and select more cost-conscious VMs. To reschedule non-critical tasks for reducing monetary costs, we need to compute the slack time for the non-critical tasks. The slack time for a non-critical task T_i can be calculated by

$$T_{slack}(v_i) = \min_{v_j \in pred(v_i)} (LFT(v_j)) - EST(v_i), \tag{17}$$

where $pred(v_i)$ is the set of immediate predecessor tasks of task v_i. We can extend the running time of task v_i to $t(i,j) + T_{slack}(v_i)$ without changing the finish time of its precursors and the start time of its successors. We devise Slack Time Scheduling Heuristic (STSH) algorithm to minimize the monetary cost for each non-critical task. At each stage, STSH calculates the slack time for each non-critical task and then reschedules it to the idle VM with the minimum monetary cost. The details of the STSH are as follows. The time complexity of STSH is $O(mn)$.

Algorithm 2: Slack Time Scheduling Heuristic (STSH)

Input: $G(N, E)$: the DAG task graph
SP: A schedule plan of the large graph

```
1 Let m_i = the virtual machine on which v_i is currently scheduled;
2 Compute AST(v_i), LFT(v_i), the slack time for each non-critical task v_i;
3 for each task v_i that has non-zero slack time do
4 |   for each idle VM m_j do
5 |   |   if no increase in makespan and c(v_i, m_i) > c(v_i, m_j) then
6 |   |   |   replace m_i with m_j;
7 |   |   end
8 |   end
9 end
```

5. Experiments and results

We use Hybrid to denote the algorithm that first uses POSH and then uses STSH to produce a scheduling plan. We carry out extensive numerical experiments to evaluate and compare the performance of Hybrid, and HEFT on large DAGs generated at random and on real-life applications. We use HEFT as the baseline algorithm for comparisons even though it does not consider monetary cost, for it is the most frequently used heuristic for the task scheduling problem [23]. We used simulations to perform a statistically significant number of experiments for a wide range of configurations without paying high price of leasing VMs for an extensive period of time. Ongoing research also indicates that the progression of clouds is moving into a direction that computing resources in a fine-grained model will be provided [30]. Thus, we carried out our experiments in a heterogeneous setting similar to those used in early works (see, e.g., [12,32]). These experiments would have been difficult to do in a real cloud.

5.1. Comparison metrics and goals

Similar to [9], we normalize the makespan and monetary cost, and call them the schedule length ratio (SLR) and monetary cost ratio (MCR), as follows:

$$SLR = \frac{makespan}{\sum_{v_i \in CP} \min_{m_j \in M}\{t(i,j)\}}. \tag{18}$$

$$MCR = \frac{C}{\sum_{v_i \in CP} \min_{m_j \in M}\{c(i,j)\}}. \tag{19}$$

The denominator is the summation of the minimum execution time and monetary cost of tasks on the critical path (CP) without communication cost. Both SLR and MCR are less than or equal to 1. For a given task graph, an algorithm that produces a scheduling plan with smaller SLR and MCR is a more cost-efficient algorithm.

5.2. Experimental setup

The cloud used in our simulations is generated with 100 heterogeneous VMs in a 100 Mb/s fully connected communication network. Similar to [32], the CPU clock rates of VM are generated uniformly and independently at random, from 1.33 GHz to 4.0 GHz. The base price for the slowest VM is 0.5 dollars per computing unit.

The DAGs are generated at random with real-life applications. In particular, we used the program [34] to generate a large number of large DAGs. This DAG generator was also used in other related works, such as [9,32]. The program generates weighted application DAGs with various characteristics depending on 5 parameters: $n, flat, density, regular$, and $jump$, where

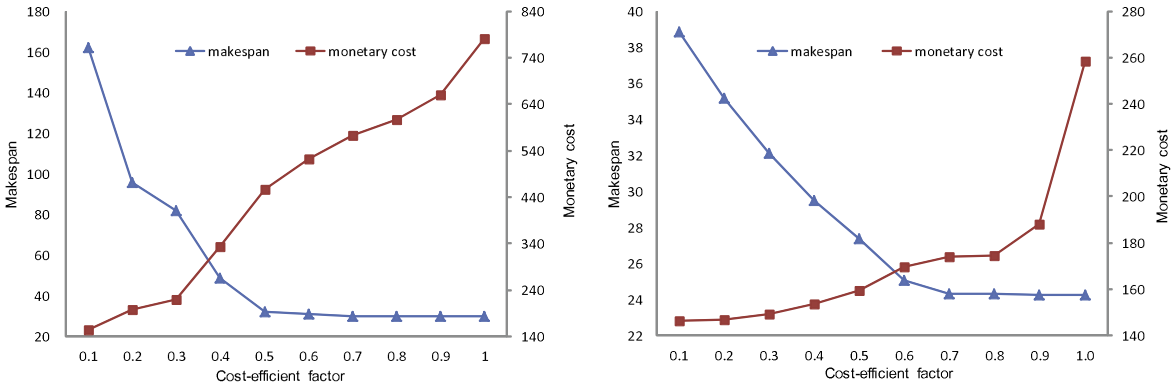
n is the number of tasks in the DAG; *flat* is the maximum number of tasks that can be executed concurrently (Note: a small value of *flat* will result in a narrow DAG with low parallelism of tasks); *density* determines the numbers of dependencies between two levels of the DAG; *regular* is the uniformity between tasks at two consecutive levels); and *jump* is the maximum number of levels spanned by inter-task communications. Changing each parameter will generate diverse DAGs with various characteristics. The real-life applications used for our experiments are the Gaussian elimination application and SIPHT bioinformatics project [35]. The computational load of the nodes were generated using historical data from real applications [36,35].

5.3. Measured cost-efficient factor

We dynamically change the input cost-efficient factor α to measure the makespan and the monetary cost. In Figs. 4 and 5, the value of the cost-efficient factor is shown in the x-axis and the average results of makespan and monetary cost are plotted in blue and red, respectively. Fig. 4 shows the makespan and the monetary cost of a random graph with 100 tasks using different cost-efficient factors and two pricing models. Fig. 5 shows the results of Gauss elimination application with 209 tasks. The results clearly show that there is a tradeoff between the makespan and monetary cost. For a smaller value of cost-efficient factor, the algorithm favors more cost-efficient VMs at the likely expense of the extended makespan. From the average results reported for different cost-efficient factors we can see that using $\alpha > 0.3$, further change in α has much effect in the overall monetary cost, and $\alpha < 0.7$, further change in α has less effect in the makespan. Hence, we next examine our algorithm for each type of large using $\alpha = 0.3, 0.5, 0.7$ and communication to computation ratios (CCR) = 0.3.

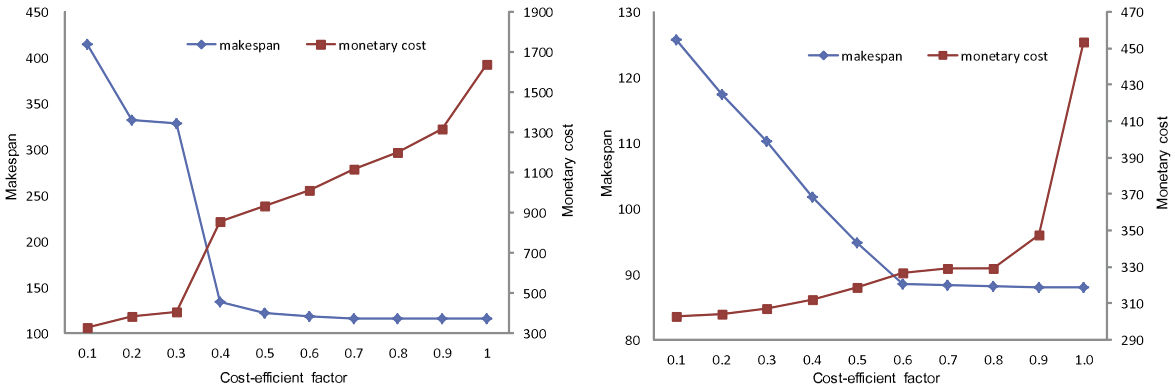
5.4. Results

In this subsection, we evaluate the performance of Hybrid against HEFT with respect to the makespan and monetary cost. We present results obtained with the use of the same machine configurations, different graph size, and both of the the linear



(a) Linear pricing model (b) Expo. pricing model

Fig. 4. Comparing different α for randomly generated DAGs.

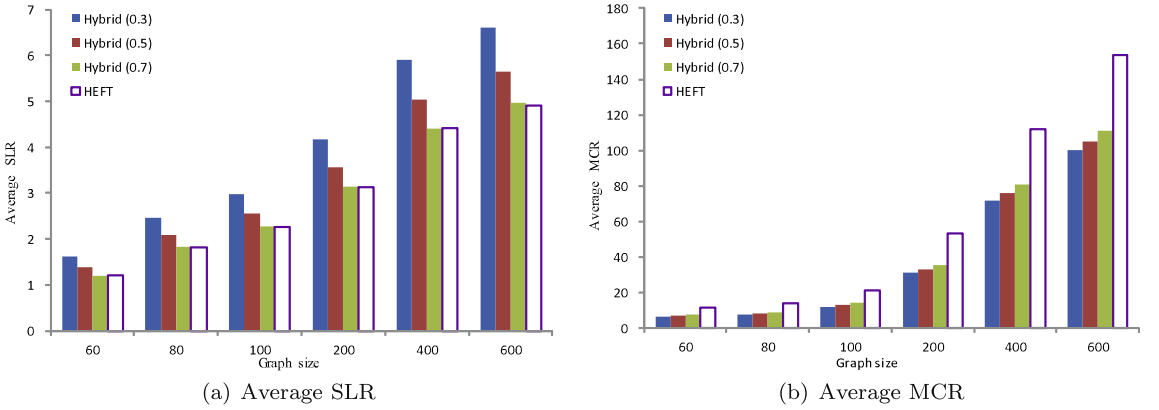


(a) Linear pricing model (b) Expo. pricing model

Fig. 5. Comparing different α for Gauss graphs.

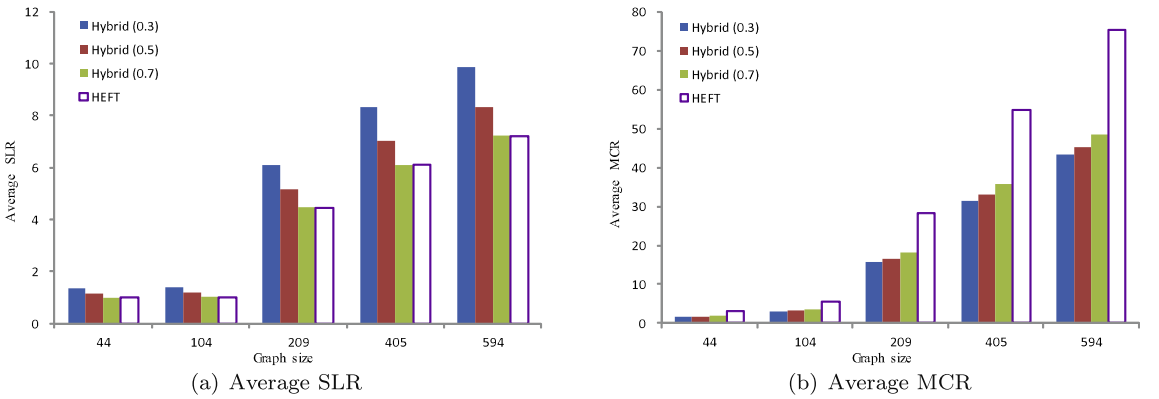
pricing model and the exponential pricing model. For each event in our experiment, we executed 25 runs and reported the average values. Because we use three cost-efficient factors and two pricing models to run each large graph under the same workload, there are $3 \times 2 \times 2 \times 6 \times 6 \times 25 = 10800$ runs.

First, we show how large applications could benefit from our task scheduling algorithms using the exponential pricing model. Fig. 6 shows the average results of SLR and MCR for each size of random graphs. Fig. 7 shows the average results of SLR and MCR for each Gauss elimination application graph. Fig. 8 shows the average results of SLR and MCR for the SIPHT bioinformatics project graphs. Table 2 shows the overall comparison results. We can see that when the cost-efficient factor



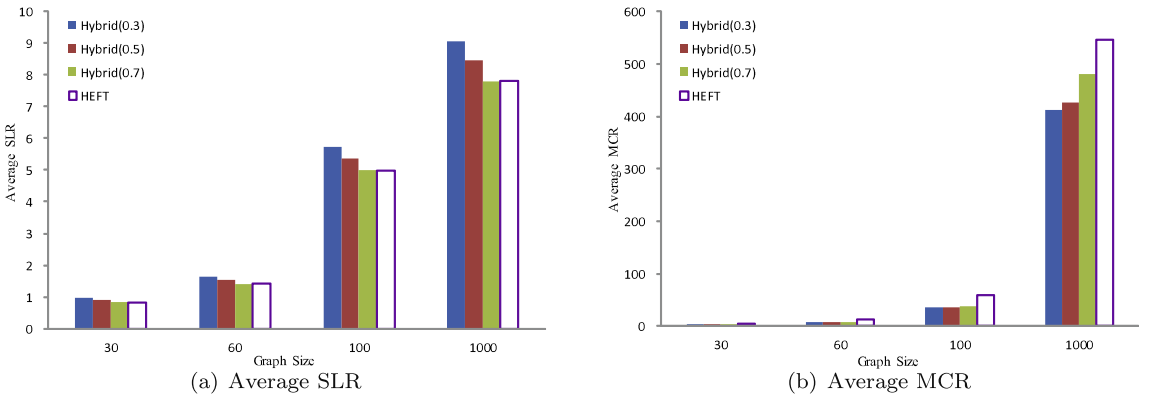
(a) Average SLR (b) Average MCR

Fig. 6. Average results for random generated large graph using exponential pricing model.



(a) Average SLR (b) Average MCR

Fig. 7. Average results for gauss elimination application graph using exponential pricing model.

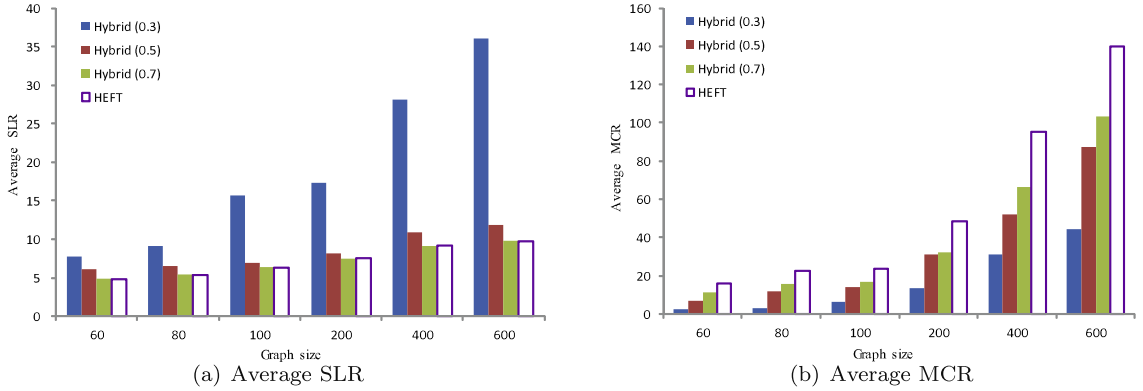


(a) Average SLR (b) Average MCR

Fig. 8. Average results for SIPHT bioinformatics project graph using exponential pricing model.

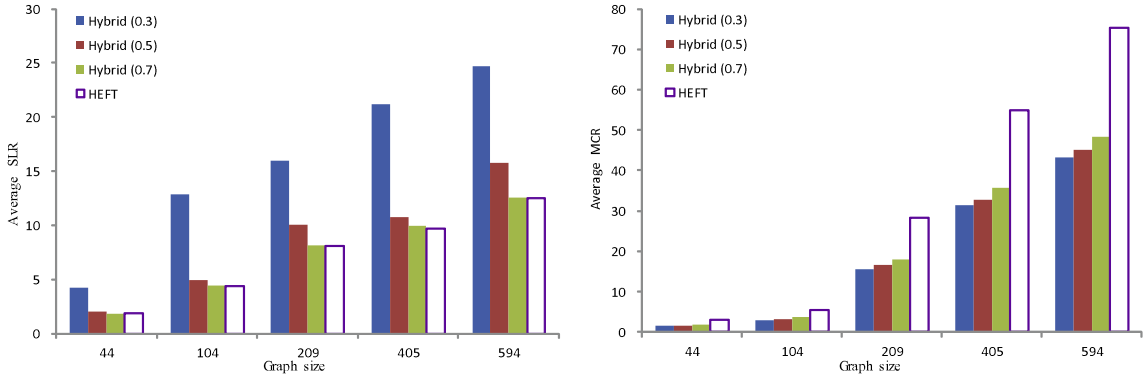
Table 2
Comparative results using exponential pricing model.

Graph	Algo					
	HEFT over H (0.7)		HEFT over H (0.5)		HEFT over H (0.3)	
	Makespan (%)	Money (%)	Makespan (%)	Money (%)	Makespan (%)	Money (%)
random (60)	0	31.2	−15.8	39.1	−35.4	43.1
random (80)	0	34.2	−14.2	40.5	−34.2	43.4
random (100)	−0.8	33.8	−13.5	40	−32.8	43.6
random (200)	0	33.3	−13.7	37.9	−32.9	40.9
random (400)	0	27.6	−14.1	32	−33.9	35.7
random (600)	−0.9	28	−14.7	31.6	−34.7	34.8
Gauss (44)	0	40.9	−15.3	47.1	−36.7	50.5
Gauss (104)	−0.5	34.2	−14.9	40.6	−35.4	44.6
Gauss (209)	−0.8	36.5	−15.9	41.8	−37.5	44.9
Gauss (405)	0	35.1	−15.1	40.1	−36.3	42.8
Gauss (594)	0	35.6	−15.4	40	−36.8	42.3
SIPHT (30)	−0.8	36.7	−10.2	40.2	−25.4	26.9
SIPHT (60)	0	38.4	−8.9	38.1	−23.8	39.9
SIPHT (100)	0	36.1	−7.8	37.4	−25.9	41.8
SIPHT (1000)	−0.4	43.1	−8.2	27.4	−27.8	43.81



(a) Average SLR (b) Average MCR

Fig. 9. Average results for random generated large graph using linear pricing model.



(a) Average SLR (b) Average MCR

Fig. 10. Average results for gauss elimination application graph using linear pricing model.

$\alpha = 0.7$, Hybrid incurs 34.98 percent less monetary cost than HEFT on average, with almost the same makespan. Recall that HEFT is a scheme that has been proven to produce generally good schedules with minimum makespan. This implies that Hybrid performs well in a cloud setting. When $\alpha = 0.5$, Hybrid incurs 38.25 percent less monetary cost than HEFT on average, with about 13.18 percent makespan extension. When $\alpha = 0.3$, Hybrid incurs 41.26 percent less monetary cost than HEFT on average, but with 32.65 percentage more makespan.

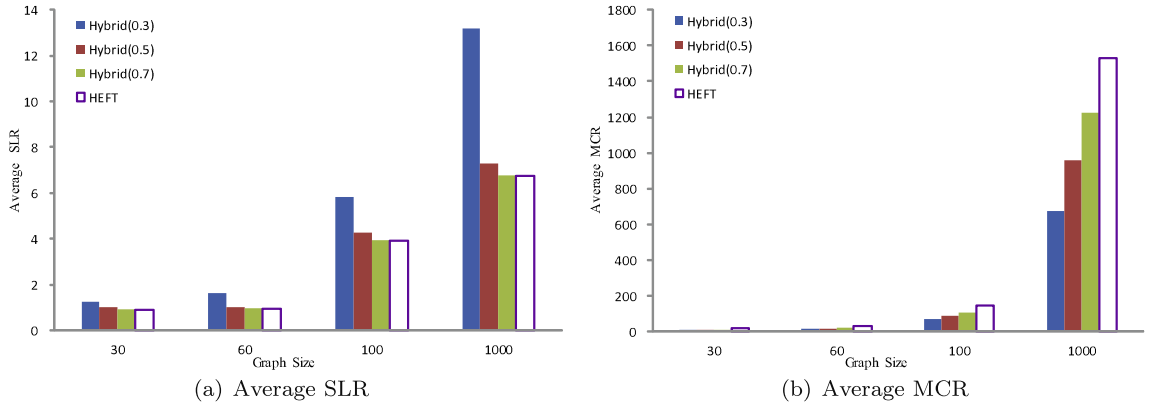


Fig. 11. Average results for SIPHT bioinformatics project graph using linear pricing model.

Table 3
Comparative results using linear pricing model.

Graph	Algo					
	HEFT over H (0.7)		HEFT over H (0.5)		HEFT over H (0.3)	
	Makespan (%)	Money (%)	Makespan (%)	Money (%)	Makespan (%)	Money (%)
random (60)	0	30.4	−16.2	54.9	−42.4	67.7
random (80)	0	30.1	−9.7	51.7	−52.5	64.8
random (100)	0	29.2	−8.6	42.4	−35	68.2
random (200)	0	30	−6.3	34.9	−69.9	38.3
random (400)	0	30.1	−15.2	47.3	−67.5	55.2
random (600)	−0.7	26.1	−21.1	38.3	−75.5	51.2
Gauss (44)	0	30.7	−19.7	41	−65	67.9
Gauss (104)	0	37.3	−19.3	44.5	−43.5	69.7
Gauss (209)	−1.11	34.1	−13.1	42.8	−64.7	72.5
Gauss (405)	−2.3	33.9	−9	62.2	−58.5	72.2
Gauss (594)	0	37.4	−3.9	45.9	−52.6	54.7
SIPHT (30)	0	19.8	−5.1	37.2	−37.5	69.3
SIPHT (60)	0	26.7	−9.9	38	−48.4	51.6
SIPHT (100)	0	31	−8.5	40.7	−58.5	67.9
SIPHT (1000)	0	30.6	−7.6	39.4	−63.8	73.8

We also calculate the SLR and MCR for large application graphs using the linear pricing model. The results for randomly generated large graphs and Gauss elimination application models follow the same trend as for exponential pricing model. Fig. 9 shows the average results of SLR and MCR for each size of random graphs. Fig. 10 shows the average results of SLR and MCR for each size of Gauss elimination application graphs. Fig. 11 shows the average results of SLR and MCR for the SIPHT bioinformatics project graphs. Table 3 shows The overall comparison results. We can see that when $\alpha = 0.7$, Hybrid incurs 30.49 percent less monetary cost than HEFT on average, with almost the same makespan. When $\alpha = 0.5$, Hybrid uses 44.1 percent less monetary cost than HEFT on average, with about 11.55 percent more makespan. When $\alpha = 0.3$, Hybrid incurs 63 percent less monetary cost than HEFT on average, but with 55.69 percent more makespan.

6. Conclusion

We devised a new task-scheduling algorithm for running large programs in the cloud. Most conventional task scheduling algorithms do not consider monetary costs, and so they cannot be directly applied in a cloud setting. In this paper, Our algorithm computes scheduling plans that produce makespan as good as the best known algorithm of while significantly reducing monetary costs. We evaluated our algorithm with extensive simulations on both randomly generated large DAGs and real-world applications.

We note that further improvements can be made using new optimization techniques (e.g., metaheuristic) and incorporating penalties for violating consumer-provider contracts. To manage monetary costs using cloud services, we would need to manage computing, storage, and network resources with a holistic approach.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61170274), by the National Key Basic Research Program of China (973 Program, 2011CB302704), by the Innovative Research Groups of the National Natural Science

Foundation of China (61121061), by the Important national science & technology specific projects (2011ZX03002-001-01), by the Beijing Municipal foundation of the Joint Research Program of Central Universities in Beijing. This work is also supported in part by the National Science Foundation (USA) under Grant CNS-1018422. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author (s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, in: Proceedings of the 2010 International Conference on Management of Data, ACM, 2010, pp. 135–146.
[2] D. Warneke, O. Kao, Nephele: efficient parallel data processing in the cloud, in: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, ACM, 2009, p. 8.
[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., Above the clouds: a berkeley view of cloud computing, Tech. rep., Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.
[4] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision hype and reality for delivering computing as the 5th utility, Future Generation computer systems 25 (6) (2009) 599–616.
[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, Dryad: distributed data-parallel programs from sequential building blocks, ACM SIGOPS Operating Systems Review 41 (3) (2007) 59–72.
[6] J. Ullman, Np-complete scheduling problems, Journal of Computer and System Sciences 10 (3) (1975) 384–393.
[7] E. Hou, N. Ansari, H. Ren, A genetic algorithm for multiprocessor scheduling, IEEE Transactions on Parallel and Distributed Systems 5 (2) (1994) 113–120.
[8] Y. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, IEEE Transactions on Parallel and Distributed Systems 7 (5) (1996) 506–521.
[9] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.
[10] R. Bajaj, D. Agrawal, Improving scheduling of tasks in a heterogeneous environment, IEEE Transactions on Parallel and Distributed Systems 15 (2) (2004) 107–118.
[11] D. Bozdag, F. Ozguner, U. Catalyurek, Compaction of schedules and a two-stage approach for duplication-based dag scheduling, IEEE Transactions on Parallel and Distributed Systems 20 (6) (2009) 857–871.
[12] Y. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys (CSUR) 31 (4) (1999) 406–471.
[13] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, et al, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed computing 61 (6) (2001) 810–837.
[14] A. WebServices. <http://www.aws.amazon.com/ec2>.
[15] G. AppEngine. <http://code.google.com/appengine>.
[16] J. Li, S. Su, X. Cheng, Q. Huang, Z. Zhang, Cost-conscious scheduling for large graph processing in the cloud, in: 2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC), IEEE, 2011, pp. 808–813.
[17] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, et al, Pegasus: a framework for mapping complex scientific workflows onto distributed systems, Scientific Programming 13 (3) (2005) 219–237.
[18] E. Ulungu, J. Teghem, Multi-objective combinatorial optimization problems: a survey, Journal of Multi-Criteria Decision Analysis 3 (2) (1994) 83–104.
[19] M. Wieczorek, A. Hoheisel, R. Prodan, Towards a general model of the multi-criteria workflow scheduling on the grid, Future Generation Computer Systems 25 (3) (2009) 237–256.
[20] G. Sih, E. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, IEEE Transactions on Parallel and Distributed Systems 4 (2) (1993) 175–187.
[21] D. Bozdag, U. Catalyurek, F. Ozguner, A task duplication based bottom-up scheduling algorithm for heterogeneous environments, in: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, IEEE, 2006, pp. 12–24.
[22] T. Yang, A. Gerasoulis, Dsc: Scheduling parallel tasks on an unbounded number of processors, IEEE Transactions on Parallel and Distributed Systems 5 (9) (1994) 951–967.
[23] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of scientific workflows in the askalon grid environment, ACM SIGMOD Record 34 (3) (2005) 56–62.
[24] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Grid multicriteria job scheduling with resource reservation and prediction mechanisms, Perspectives in Modern Project Scheduling, 2006. pp. 345–373.
[25] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Scheduling jobs on the grid-multicriteria approach, Computational methods in science and technology 12 (2) (2006) 123–138.
[26] J. Yu, R. Buyya, C. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: First International Conference on e-Science and Grid Computing, 2005, IEEE, 2005, pp. 8–pp.
[27] J. Yu, R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in: Workshop on Workflows in Support of Large-Scale Science, 2006, WORKS'06, IEEE, 2006, pp. 1–10.
[28] R. Sakellariou, H. Zhao, E. Tsiakkouri, M. Dikaiakos, Scheduling workflows with budget constraints, Integrated Research in Grid Computing (2007) 189–202.
[29] G. Singh, C. Kesselman, E. Deelman, A provisioning model and its comparison with best-effort for performance-cost optimization in grids, in: Proceedings of the 16th international symposium on High performance distributed computing, ACM, 2007, pp. 117–126.
[30] Q. Zhu, G. Agrawal, Resource provisioning with budget constraints for adaptive applications in cloud environments, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, 2010, pp. 304–307.
[31] S. Garg, R. Buyya, H. Siegel, Time and cost trade-off management for scheduling parallel applications on utility grids, Future Generation Computer Systems 26 (8) (2010) 1344–1355.
[32] J. Barbosa, B. Moreira, Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters, Parallel Computing 37 (8) (2011) 428–438.
[33] L. Bittencourt, R. Sakellariou, E. Madeira, Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm, in: 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2010 , IEEE, 2010, pp. 27–34.
[34] D. generation program. <http://www.loria.fr/suter/dags/html>.
[35] P. Project. <https://confluence.pegasus.isi.edu/display/pegasus/workflowgenerator>.
[36] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, K. Vahi, Characterization of scientific workflows, in: Third Workshop on Workflows in Support of Large-Scale Science, 2008, WORKS 2008, IEEE, 2008, pp. 1–10.