International Journal of Computer Applications (0975 - 8887)
Volume 181 - No.15, September 2018

Scheduling Preemptive and Non-Preemptive Tasks of Scientific Workflows using Hybrid Instances in Cloud Environment

Vinay K.
UVCE
Bangalore
Karnataka, India

S. M. Dilip Kumar
UVCE
Bangalore
Karnataka, India

Venugopal K. R.
UVCE
Bangalore
Karnataka, India

ABSTRACT

SWf (Scientific Workflows) are vastly used in scientific domains and typically include non-preemptive and preemptive tasks. Cloud computing facilitates an appropriate ways to access cloud resources as a “pay-as-you-go" model and several resources such as, reserved, on-demand and spot instances are offered by the cloud service providers. The spot instance renting price is less as compared to on-demand instances. But, failures happen due to difference in the instance bid price. Henceforth, it is a challenge to schedule the preemptive and non-preemptive tasks of SWf onto appropriate spot and on-demand spot instances. Therefore, in this paper a SWf scheduling problem using both spot and on-demand instances are considered and the main objective is to reduce the total execution cost under deadline constraints. An efficient rule-based scheduling algorithms are proposed to schedule non-preemptive and preemptive tasks of SWf. The algorithm considers three different rules such as, maximum number of successors, minimum processing time, and minimum slack time to schedule SWf efficiently. Experimental results demonstrate the effectiveness of the proposed rule-based task sequence initialization and virtual machine selection algorithms for different SWf sizes.

Keywords

Cloud Computing, Scientific Workflows, Scheduling, Preemptive, Non-Preemptive, On-demand, Spot, Virtual Machine

1. INTRODUCTION

Cloud computing is a pay-as-you-go oriented resource provisioning paradigm that facilitates cloud users to access resources anywhere and at anytime. Resources such as, server, network, storage, etc. in cloud are virtualized and cloud users do not need to manage large resources, rather pay for the resources consumed from the cloud service provider. SWf applications such as, astronomy application (Montage), bio-informatics project (SIPHT) and astrophysical application (LIGO) [1] typically contains both preemptive and non-preemptive tasks. The tasks that can be interrupted at any time during the execution are termed as P (preemptive) tasks, for example, web-crawler application and the tasks that cannot be interrupted during execution are termed as NP (non-preemptive) tasks, for example, batch processing. In Amazon EC2 [2] there are reserved, on-demand and spot instances in which reserved instances are used for a long period of time. However, the resource utilization is low. The on-demand instance offers users to pay for resources on hourly basis. The price of on-demand is much higher than the reserved instances, but the resource utilization is high as compared with reserved instances. The spot instance resources are subscribed through bidding and the spot instance are allocated to the user only when the user's bid price is higher than the spot price during bidding. Spot instances are available until new higher price bid occurs. But, the price of spot instance is significantly lesser than the on-demand and reserved instances.
The main objective in this paper is to address the scheduling problem of P and NP tasks of SWf in cloud environment and reduce the total execution cost of the SWf. Since the SWf are short term compute-intensive applications and does not require reserved and on-demand instances that make resource cost high. Out-of-bid events when spot instances are considered impose improper scheduling for NP tasks. Hence, in-order to overcome the above mentioned problem, spot-block instances are considered which is also an extension of spot instance. Spot-block instance will not get terminated and run for the defined $1, 2, 3, 4, 5$, or 6 hours. And the price of spot-block instance is 30% to 45% less than on-demand instances. The spot-block instance is ideal for NP tasks only when the execution time of NP tasks are less than the spot-block instance defined hours. Otherwise, the on-demand instance is selected to ensure non-preemptive tasks are not interrupted. P tasks can be scheduled in any of the instances, since these tasks can be interrupted.
This paper presents a scheduling heuristic, SWf architecture for scheduling P and NP tasks of using hybrid instances such as, spot-block and on-demand instances. It minimizes the execution cost and also provides robust schedule to meet the deadline constraint of SWf.

1.1 Contributions

—A new scheduling of P and NP tasks of SWf are considered.

—A scheduling heuristic that uses *spot-block* and *on-demand* resources to schedule SWf in a cost efficient manner.

—A rule-based task sequence initialization and VM selection algorithm to schedule P and NP tasks of SWf.

The rest of the paper is organized as follows. Section 2 establishes the related works. The problem definition and formulation are described in Section 3. The proposed heuristics are presented in Section 4. Computational results are shown in Section 5, followed by conclusions in Section 6.

2. RELATED WORKS

Scheduling scientific applications in cloud is an emerging discussion in the recent years. In the traditional grid computing, resources were geographically dispersed and provided access to the end users. Cost optimization under deadline constraints and execution time optimization under budget constraints [3] are the two main reasons to adopt cloud computing.Time optimization algorithms include scheduling algorithm cluster based algorithm [4], [5], duplication based algorithm [6], ant colony optimization approach [7] and greedy randomized adaptive search [8]. Cost optimization algorithm include, DET (Deadline Early Tree), deadline-MDP [9], CPI (Critical Path-based Iterative) heuristic and PCP (Partial Critical Paths) [10]. Vinay et al. [11] proposed a cost-aware and fault-tolerant aware resource management for SWf using block-spot and on-demand instances. The SWf tasks considered in this work are homogenous. There are limited works on SWf scheduling with spot instances. A fault-tolerant algorithm was proposed by Poola et al. [12] to schedule tasks onto spot and on-demand instances in order to reduce execution cost of the deadline constrained SWf. Further, Poola et al. [13], also proposed scheduling algorithm, named, adaptive just-in-time for SWf. Both spot and on-demand instances were considered to reduce the execution cost and fault-tolerant during execution. Jung et al. [14] proposed a task balanced workflow scheduling scheme to mitigate out-of-bid situation and the total task completion time. Further, Jung et al. [15], also proposed a Genetic Algorithm (GA)-based workflow scheduling scheme to find the optimal task size in a spot instance based cloud environment without increasing users' budgets.
There are limited or no works that considers scheduling P and NP tasks of SWf and these tasks are commonly exists in SWf. The on-demand instances are usually considered for SWf scheduling in the existing literature. A few studies also considered the spot alternative. However, the high cost and out-of-bid failure make these two instances more expensive. Therefore, scheduling SWf with both P and NP tasks on spot-block and on-demand instances are considered in this paper.

3. PROBLEM DEFINITION AND FORMULATION

SWf are vastly used in scientific domains, which typically includes P and NP tasks. The problem is to schedule the P and NP tasks of SWf onto appropriate spot and on-demand spot instances of cloud in order to reduce the execution cost under deadline constraint. Figure 1 depicts the SWf scheduling model with P and NP tasks. SWf users send their requests to cloud service provider to run P and NP tasks and it consists of SWf application layer, SWf scheduling layer and cloud infrastructure. Initially, the different SWf are decomposed into set of P and NP tasks before scheduling to cloud for execution. The scheduler allocate these tasks to each of the available resources. The proposed model uses on-demand and block-spot instances.

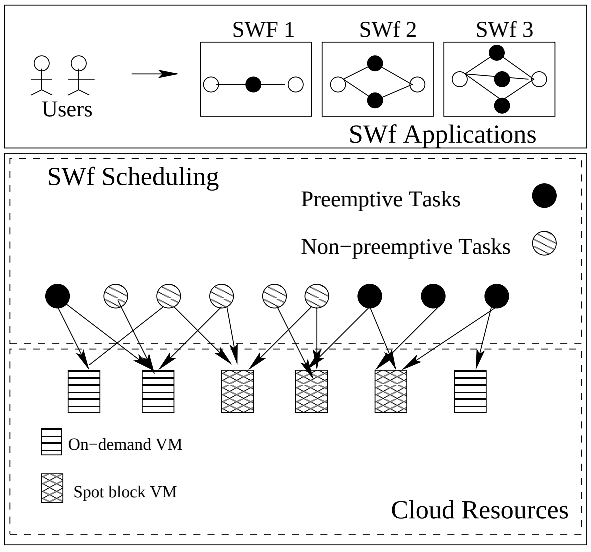

Fig. 1: SWf Architecture

3.1 Mathematical model

The SWf application is denoted by Directed Acyclic Graph (DAG) $G(V, E)$, in which $V = \{v_0, \ldots, v_n\}$ are the tasks and $E = \{(v_i, v_j) | v_i \in V, v_j \in V, i < j\}$ are edges in G.

—Each edge $E = \{(v_i, v_j)\}$ represents the dependency between the task v_i and v_j.

—v_0 and v_n are used as dummy start and end nodes of tasks in SWf.

—P and NP represents preemptive and non-preemptive tasks.

—The task v_i can be executed on one or many homogeneous instances.

—The processing time of v_i on a single resource is represented by PT_i.

—The preemptive task $v_i \in P$ are separated and further allocated on different available resources during scheduling process. But, for non-preemptive tasks $v_i \in NP$, it has to be executed on either single on-demand or block-spot instance only.

—C^o is the unit cost of on-demand instance and C^s is the unit cost of spot block instance at time t.

—The SWf deadline is represented by D.

—s_i and e_i are start and end times of task v_i the total number of resources considered for the execution be R.

Then, the overall execution cost of the SWf TC_{SWf} is represented by the Equation 1.$C^o \times y_r \times T_r$ is the cost of on-demand instances. $C^s \times y_r \times T_r$ estimates the cost of spot-block instances, in which T_r is the task block time of P and NP tasks respectively.

$$TC_{SWf} = \sum_{r=1}^{R} (C^o \times y_r \times T_r + C^s \times y_r \times T_r) \tag{1}$$

Table 1. : The Notations used in the Mathematical Model

Parameters	Definitions
PT_i	Processing time of task v_i
P	Preemptive tasks
NP	Non-preemptive tasks
C^o	Cost of on-demand instance
C^s	Cost of spot block instance
D	SWf Deadline
s_i and e_i	start and end times of task v_i
TC_{SWf}	Total execution cost of SWf
x_i	Binary variable
y_r	Binary variable
T_r	Task block time of preemptive and non-preemptive tasks
q_i	VMs instances required for task v_i

$$x_i = \begin{cases} 1 & v_i \text{ is executed on VM } r \text{ at time } t,\ \forall i \in \{0, n\},\ \forall r \in \{1, R\},\ \forall t \in \{0, D\} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$y_r = \begin{cases} 1 & r \text{ is on-demand VM},\ \forall r \in \{1, R\} \\ 0 & r \text{ is spot-block VM},\ \forall r \in \{1, R\} \end{cases} \tag{3}$$

The binary variable x_i in Equation (2) takes 1 if the task v_i is executed on VM r at time t. The binary variable y_r in Equation (3) takes 1 if the VM r is an on-demand VM, otherwise takes 0 if the VM r is a spot-block VM.

$$T_r = \sum_{i=0}^{n} \sum_{t=0}^{D} x_i,\ \forall r \in \{1, R\} \tag{4}$$

$$s_i = \min_{t \in \{0,D\}} (\sum_{r=1}^{R} t \times x_i),\ \forall i \in \{0, n\} \tag{5}$$

$$e_i = \max_{t \in \{0,D\}} (\sum_{r=1}^{R} t \times x_i),\ \forall i \in \{0, n\} \tag{6}$$

$$q_i = \begin{cases} \sum_{r=1}^{R} \sum_{t=0}^{D} x_i, & \forall v_i \in P \\ 1, & \forall v_i \in NP \end{cases} \tag{7}$$

The cost of on-demand and spot-block instances incurred during execution are calculated by Equation (4). (5) and (6) are used to calculate start and end time of task v_i. Equation (7) calculates the instances required for a task v_i.

4. PROPOSED MODEL

Rule-based SWf scheduling for SWf applications are the common approaches. In this paper an efficient rule-based scheduling algorithms are proposed to schedule non-preemptive and preemptive tasks of SWf. The algorithm considers three different rules such as, minimum processing time, maximum number of successors and minimum slack time to schedule SWf efficiently and are defined as follows:

(1) **Minimum Processing Time:** Minimum processing time of the tasks are identified by the characteristics of the task and the tasks priorities are estimated by the processing time and minimal processing time has the highest priority.
(2) **Maximum number of Successors:** The structures of the SWf are considered and the priorities of the tasks are estimated by the number of successors.
(3) **Minimum Slack Time:** The structure of SWf and the characteristics of the tasks are considered. The total slack time of a task v_i is determined by $TS(v_i) = LS(v_i) - ES(v_i)$. The task which has least slack time are considered as the highest priority.

In rule-based scheduling algorithm, the sequence of tasks are determined by the topological-order of the SWf. The task allocation to different VMs be $T$of the task and sequence initialization of tasks are shown in Algorithm 1. The algorithm begins from dummy start node v_0, and the tasks that has no predecessors are included in the qualified set QS. Further, the highest priority tasks are selected from QS and included to T and the final sequence set FS.

Algorithm 1: Tasks Sequence Initialization (TSI))

```
1  $V = \{v_o, v_1, ...., v_n\}$;
2  $E = \{(v_i, v_j)|v_i \in V, v_j \in V\} and v_i < v_j$;
3  begin
4      $FS \leftarrow \emptyset, QS \leftarrow \emptyset\ T \leftarrow \emptyset$;
5      repeat
6          $FS \leftarrow FS \bigcup \{v_i\}$;
7          $T \leftarrow T \bigcup \{v_i\}$ ;
8          for each $(v_i, v_j) \in E$ do
9              if $\forall (v_i, v_j) \in E, v_i \in FS$ then
10                 $QS \leftarrow FS \bigcup \{v_j\}$;
11         Using rules to select a task $v_i$ from the qualified set $QS$;
12     until $(v_n \in FS)$;
13     return $T$;
```

A VM selection strategy is proposed to schedule preemptive and non-preemptive tasks of SWf. Only homogenous on-demand and spot instances are considered. The preemptive tasks can be interrupted during execution, while non-preemptive tasks cannot be interrupted and the selection of different VMs are shown in Algorithm 2. The selection of VM for scheduling of tasks are mainly based on the topological-order obtained from Algorithm 1. The estimated execution time is calculated for the tasks in the FS. Suppose, the task is a NP, then the estimated task execution time ET_i is equal to the processing time of task v_i on a single VM, i.e., $ET_i = P_i$. If the task is v_i P, then the task is separated and scheduled onto the available VMs, i.e., $ET_i = P_i/|QS|$.

5. EXPERIMENTAL EVALUATION

In this section we discuss the experiments conducted to evaluate the heuristics. Only homogeneous VMs are considered and simulate the VM resources on real clouds, The CloudSim toolkit [16] is extended to support on-demand and spot-block instances. The VM instances considered is $m4.large$ Amazon EC2 [2] and the cost of the instances are tabulated in Table 2). The time required to start a VM is assumed to be zero.
The two SWf such as LIGO and Montage [1] are considered to analyze the performance of proposed algorithms. Montage SWf are

Algorithm 2: VM Selection (VMS) Algorithm

```
1  Input: the task allocation sequence $T$.
2  Output: the estimated execution time of all the tasks in $T$.
3  begin
4      repeat
5          $FS \leftarrow \emptyset, QS \leftarrow \emptyset$;
6          $v_i \leftarrow$ the first task of $T$;
7          $FS \leftarrow FS \bigcup \{v_i\}$;
8          for each $(v_i, v_j) \in E$ do
9              if $\forall (v_i, v_j) \in E, v_i \in FS$ then
10                 $QS \leftarrow QS \bigcup \{v_j\}$;
11         if $v_i \in P$ then
12             $ET_i = P_i / |QS|$;
13         else
14             $ET_i = P_i$;
15         $T = T - \{v_i\}$;
16     until $(T = \emptyset)$;
17     return;
```

Table 2. : Unit cost of spot and on-demand instances

Instance Type	vCPU	On-demand Hourly	Spot Block Hourly	
			1 hour	6 hours
m4.large	2	\$0.120	\$0.069	\$0.088

used by NASA/IPAC [1] and it merges multiple input images together to create custom mosaics of the sky. The Laser Interferometer Gravitational Wave Observatory (LIGO) [1] generates gravitational waveforms from data collected by compact binary star systems. Figure 2 and Figure 3 presents the structure of Montage and LIGO SWf. The nodes mProjectPP, mDiffFit and mJpEG in Figure 2 are the preemptive tasks while other nodes are non-preemptive tasks. The nodes TrigBank in Figure 3 are non-preemptive tasks and other nodes are preemptive tasks.

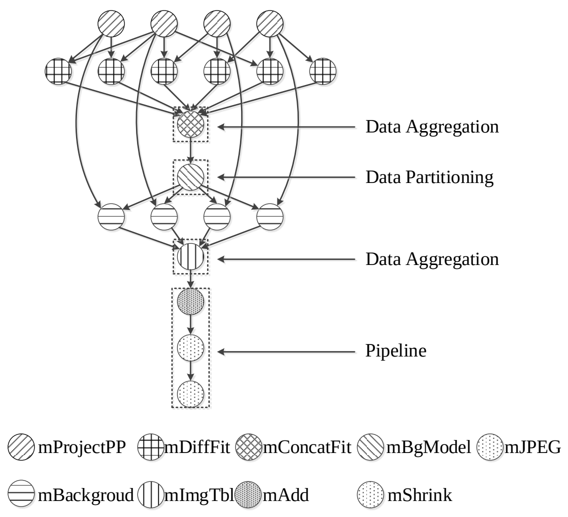

Fig. 2: An example of Montage SWf

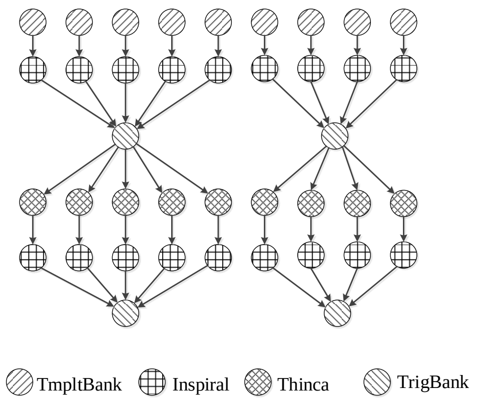

Fig. 3: An example of LIGO SWf

However, experiments were carried out with five different task sizes $(50, 100, 200, 300, 400)$ and the inter-arrival time of tasks are varied to fully explore the performance of the proposed heuristics. The observed output metrics are cost and deadline factor of SWf.
To analyze the efficiency of the algorithms, a a Relative Percentage Deviation (RPD) is used. Suppose, if we consider an instance I, then the optimal schedule and the corresponding cost $C_(\pi_I)$ among compared algorithms is measured by the Equation 8.

$$RPD_I = \frac{C(\pi_I) - C(\pi_I^*)}{C(\pi_I^*)} \times 100\% \tag{8}$$

The proposed approach is compared with the just-in-time (JIT) algorithm proposed by Poola et al. [13]. The VMS and JIT methods are also adopted to consider only on-demand instances. VMS_o and JIT_o are the adopted methods with only on-demand instances. Therefore, VMS, JIT, VMS_o and JIT_o are the different methods compared to evaluate the proposed approaches.

5.1 Cost Evaluation

5.1.1 Montage. The Fig 5 shows VMS performs better than the JIT, VMS_o and JIT_o for different tasks sizes. VMS saves 10% and 20% cost on average while compared with JIT and VMS_o respectively. However, with an increase in tasks, the cost of VMS and JIT increases in similar manner.

5.1.2 LIGO. The Fig 5 shows, the VMS performs better than the VMS_o, JIT and JIT_o for different task sizes. When the SWf size is less, the VMS and JIT behaves same. But, when the size of SWf increases, VMS performs better than JIT. VMS saves 20% cost on average

5.2 Deadline Evaluation

5.2.1 Montage. The deadline evaluation for Montage SWf are depicted in Fig 6. JIT performs better than the proposed algorithm when the deadline of SWf is small. But, when the deadline increases, RPD of JIT also increases sharply while VMS increases slowly. The deadline parameter has less influence on the proposed algorithm, but has bigger influence on JIT method.

5.2.2 LIGO. The comparison between different algorithms with different deadlines for LIGO SWf are shown in Fig 7. It is clearly evident from the result that the VMS performs better than other three algorithms for the all the deadlines with different values. The

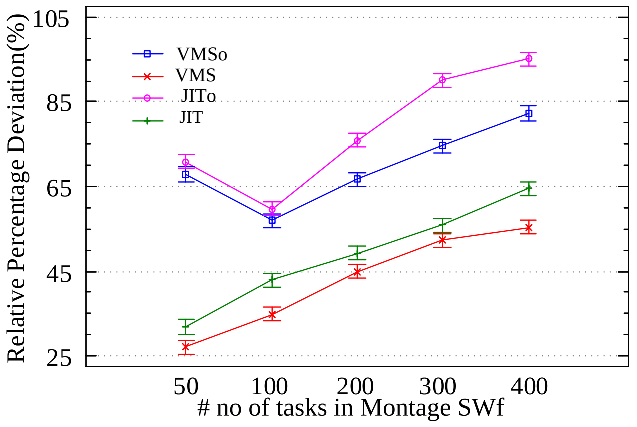

Fig. 4: Comparison of Algorithms for Montage SWf with different number of tasks.

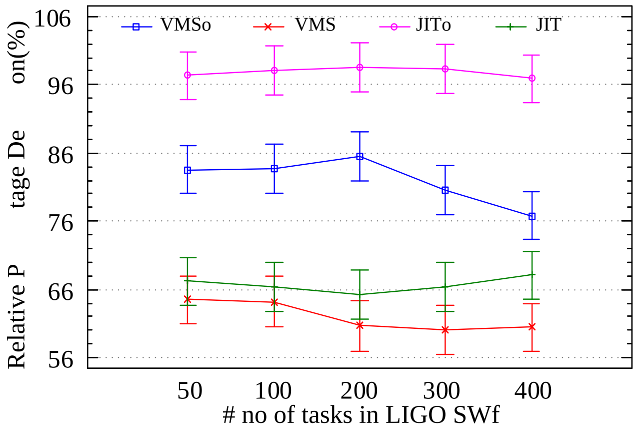

Fig. 5: Comparison of Algorithms for LIGO SWf with different number of tasks.

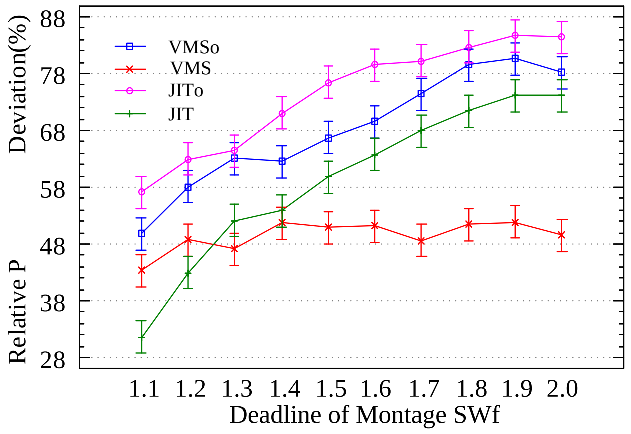

Fig. 6: Comparison of Algorithms for Montage SWf with deadline factors.

proposed VMS saves 10% of the total cost when compared to JIT method and saves 15% of the total cost than the VMS_o method. However, the VMS, JIT and VMS_o remains identical with an increase in the deadline. The deadline parameter has little influence on the LIGO SWf.

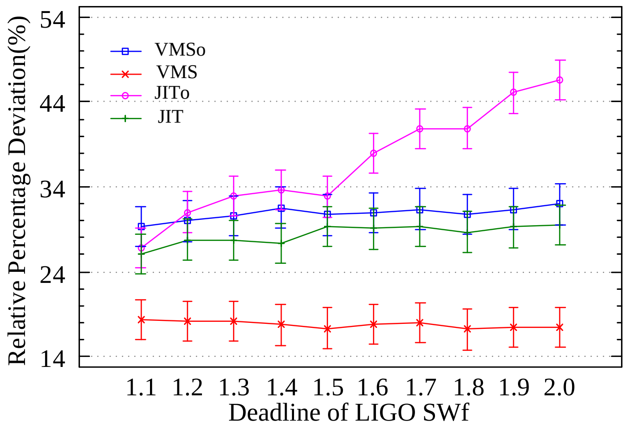

Fig. 7: Comparison of Algorithms for LIGO SWf with different deadline factors.

6. CONCLUSION AND FUTURE WORK

In this paper, scheduling of SWf using spot-block and on-demand instances are proposed. Further, a mathematical model to classify preemptive and non-preemptive tasks to schedule onto spot-block and on-demand instances. Furthermore, a rule-based task sequence initialization and VM selection algorithm heuristics proposed can be easily adapted to other SWf scheduling algorithms. The experimental result shows that the proposed VM selection algorithm has better performance considering spot-block and on-demand instance with different sizes. In future, this work can further be enhanced by considering heterogenous resources and validate in real cloud environments.

7. REFERENCES

[1] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10. IEEE, 2008.

[2] AmazonEC2. Amazon elastic compute cloud (Amazon EC2). *http://aws.amazon.com/ec2/pricing*, 2014.

[3] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3):217–230, 2006.

[4] Apostolos Gerasoulis and Tao Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):686–701, 1993.

[5] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[6] Michael A. Palis, Jing-Chiou Liou, and David S. L. Wei. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):46–55, 1996.

[7] Wei-Neng Chen and Jun Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(1):29–43, 2009.

[8] James Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, volume 2, pages 759–767. IEEE, 2005.

[9] Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *First International Conference on e-Science and Grid Computing (e-Science 2005)*, page 8. IEEE, 2005.

[10] S. Abrishami, M. Naghibzadeh, and DHJ Epema. Cost-driven scheduling of grid workflows using partial critical paths. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1400–1414, 2012.

[11] K Vinay, SM Dilip Kumar, S Raghavendra, and KR Venugopal. Cost and fault-tolerant aware resource management for scientific workflows using hybrid instances on clouds. *Multimedia Tools and Applications*, 77(8):10171–10193, 2018.

[12] Deepak Poola, Kotagiri Ramamohanarao, and Rajkumar Buyya. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Computer Science*, 29:523–533, 2014.

[13] Deepak Poola, Kotagiri Ramamohanarao, and Rajkumar Buyya. Enhancing reliability of workflow execution using task replication and spot instances. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(4):30, 2016.

[14] Daeyong Jung, JongBeom Lim, JoonMin Gil, Eunyoung Lee, and Heonchang Yu. Task balanced workflow scheduling technique considering task processing rate in spot market. *Journal of Applied Mathematics*, 2014, 2014.

[15] Daeyong Jung, Taeweon Suh, Heon-Chang Yu, and Joon-Min Gil. A workflow scheduling technique using genetic algorithm in spot instance-based cloud. *TIIS*, 8(9):3126–3145, 2014.

[16] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.