Deadline-Budget constrained Scheduling Algorithm for Scientific Workflows in a Cloud Environment

Mozhgan Ghasemzadeh[1], Hamid Arabnejad[2], and Jorge G. Barbosa[3]

1 **LIACC, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal**
`mozhgan.ghasemzadeh@fe.up.pt`
2 **IC4, Dublin City University, Dublin, Ireland**
`hamid.arabnejad@dcu.ie`
3 **LIACC, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal**
`jbarbosa@fe.up.pt`

Abstract

Recently cloud computing has gained popularity among e-Science environments as a high performance computing platform. From the viewpoint of the system, applications can be submitted by users at any moment in time and with distinct QoS requirements. To achieve higher rates of successful applications attending to their QoS demands, an effective resource allocation (scheduling) strategy between workflow's tasks and available resources is required. Several algorithms have been proposed for QoS workflow scheduling, but most of them use search-based strategies that generally have a higher time complexity, making them less useful in realistic scenarios. In this paper, we present a heuristic scheduling algorithm with quadratic time complexity that considers two important constraints for QoS-based workflow scheduling, time and cost, named Deadline-Budget Workflow Scheduling (DBWS) for cloud environments. Performance evaluation of some well-known scientific workflows shows that the DBWS algorithm accomplishes both constraints with higher success rate in comparison to the current state-of-the-art heuristic-based approaches.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Resource management, QoS scheduling, scientific workflow applications, deadline-constrained, budget-constrained

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2016.19

1 Introduction

Cloud computing infrastructures are the new platforms for tackling the execution needs of large-scale applications. Cloud computing promises the important benefits such as providing nearly-unlimited computing resources to execute application's task, on-demand scaling and pay-per-use metered service. Computing resources (i.e. virtual machines (VMs)) are dynamically allocated to user tasks based on application requirements, and users just pay for what they use. Each large-scale workflow application contains several tasks. Generally, workflow application can be represented by a Directed Acyclic Graph (DAG) that includes independent tasks, which can be executed simultaneously, or dependent tasks which need to be executed in a given other. In order to meet user's application QoS parameters, we need to find an efficient schedule map to execute the application tasks on multiple resources.

The majority of studies about workflow scheduling focus on single workflow application scheduling. However, these approaches are not adequate for cloud infrastructures due to


© Mozhgan Ghasemzadeh, Hamid Arabnejad, and Jorge Barbosa;
licensed under Creative Commons License CC-BY

two major features: pay-as-you-go pricing model and on-demand resource provisioning. For example, in [1, 20, 28, 31, 30, 5, 13] authors considered fixed number of resources to the whole life time of the workflow application. But in our work, resources can be acquired at any time and released when they are idle, which save the total charged cost. Further, other approaches such as in [1, 20, 28, 31, 30] did not consider the hourly charging billing model in the cost model or the data transfer time in total time reservation of the virtual machine, which affects the effectiveness of the algorithm. In cloud computing infrastructures, such as Amazon EC2[1], the charging policy is based on a hour billing model even if the whole last reservation interval is not used. In this case, time fractions produced by previous tasks can be used by later tasks to save total renting cost. On the other hand, the workflow scheduling problem becomes more challenging when we consider multiple QoS parameters. Many algorithms have been proposed for multi-objective scheduling, but in most of them, meta-heuristic methods or search-based strategies have been used to achieve good solutions. However, these methods based on meta-heuristics or search-based strategies usually need significantly high planning costs in terms of the time consumed to produce good results, which makes them less useful in real platforms that need to obtain map decisions on the fly.

In this paper, a low-time complexity heuristic, named Deadline-Budget Workflow Scheduling (DBWS), is proposed to schedule workflow applications on cloud infrastructures constrained to two QoS parameters, namely, time and cost. The objective of the proposed DBWS algorithm is to find a feasible schedule map that satisfies the user defined deadline and budget constraint values. To fulfill this objective, the proposed approach implements a mechanism to control the time and cost consumption by each task when producing a schedule solution. To the best of our knowledge, the algorithm proposed here is the first *low-time* complexity heuristic, for cloud computing environments, addressing two QoS parameters as constraints.

The contributions of this paper are:

- a review of multiple QoS parameter workflow scheduling on cloud computing environments;
- a new heuristic algorithm with quadratic complexity for workflow application scheduling, constrained to time and cost;
- extensive evaluation with results for real-world applications.

The remainder of the paper is organized as follows. After outlining the related work in Section 2, we introduce the application and infrastructure model in Section 3. Section 4 presents the proposed scheduling algorithm. Section 5 presents results, and Section 6 concludes the paper.

2 Related work

The primary goal of many scheduling algorithms on cloud computing systems has focused on reducing the execution time of workflow applications without considering other factors such as the monetary cost or deadline. In [2, 24] we can find a taxonomy of scheduling algorithms for cloud computing systems. Considering multi-objectives for scheduling, we classify scheduling algorithms into the following two main categories: single workflow and multiple workflow scheduling algorithms. As the scheduling constraints on this paper are time and cost, we only consider these two QoS parameters in our review of previous work. Nevertheless, there are other QoS parameters such as reliability or energy that are not

[1] http://aws.amazon.com/ec2

considered here. Also, once our target platform is a cloud computing system, works that were proposed for grid infrastructures are not considered in this review because of different assumptions in the cost model. In cloud pricing model, i.e. a hour billing model, if the whole of the time interval is not used, it is still charged. Therefore, the formula used for cost consumption in grid platforms cannot be used in the cloud model and we cannot compare grid scheduling approaches with cloud ones in terms of cost consumption.

2.1 Single Workflow Scheduling Algorithms

In this category, the scheduling algorithms aim to find a suitable schedule map between workflow's task and available resources in order to meet application objective function which could be to optimize or to be constrained to a single or to multiple QoS parameters. Our work is related to the strategies which consider time and cost as QoS parameters for workflow scheduling.

2.1.1 Cost-optimization, deadline-constraint

The deadline of a workflow is defined as the maximum finish time of its last task to be executed. Calheiros et al. [5] developed an algorithm that is a cost-minimizer and applies replication of tasks to increase the chance of meeting application deadlines. Sahniet et al. [20] proposed a dynamic cost-effective deadline-constrained heuristic algorithm, namely JIT-C, for scheduling a scientific workflow in a public Cloud. In addition to these heuristic-based scheduling strategies, several works [16, 6] were proposed with the same objectives that by using search-based or meta-heuristic methods aims to find good solutions.

2.1.2 Time-optimization, budget-constraint

Budget is defined as the maximum amount that a user wants to pay for executing a workflow application on computing resources. In [13, 28, 31], authors proposed heuristic-based scheduling algorithms to minimize end-to-end execution under user-specified financial cost constraint. Zeng et al.[30] proposed a security-aware and budget-aware workflow scheduling strategy (SABA) for reducing the total execution time while meet required level of security.

2.1.3 Time-optimization, cost-optimization

Most strategies in this class try to mange the trade-off between running time and cost in order to minimize both QoS parameter time and cost in the provided schedule map. Selvarani et al. [22] proposed a job scheduling algorithm for making efficient mapping of independent tasks to available resources in a cloud. Lee et al. [11] proposed critical-path-first scheduling (CPF) algorithm which uses methods of stretching and compacting the workflow to optimize time and cost. In [4] authors proposed three bi-criteria complementary approaches for scheduling workflows on distributed Cloud resources. The first two algorithms, namely cost-based and time-based approaches, aim to minimize a single objective function (execution cost or time) individually by using Pareto approach, while the third algorithm, namely cost-time-based approach, is based on the obtained solutions by the two first algorithms for selecting only the Pareto solutions. By using the concept of Pareto dominance, authors in [25] proposed an algorithm that minimize total execution time and cost by setting a cost-efficient factor that represents the user's preference for the execution time and the monetary cost. The similar technique was used in [29]. Don et al. [14] proposed a framework which provided the balance between the application schedule performance and mandatory cost on Cloud

resources. However, our problem is different from these approaches in that both time and cost are treated as constraints at the same time, whereas these works try to consider one variable as constraint and optimize the other one, or target to optimize both cost and time.

2.1.4 Deadline-constraint, budget-constraint

Poola et al. [18] proposed a robust heuristic algorithm for scheduling a workflow on cloud computing systems considering deadline or budget constraints. The algorithm uses a search-based strategy to reassign scheduled tasks to new resources in order to satisfy the workflow constraint values for time or cost parameters. In [19], Rahman et al. present an adaptive hybrid heuristic (AHH) for workflow scheduling in hybrid cloud environment.

Two major drawbacks of the previous research work is that: a) usually, in their approaches the pricing model is the pay-as-go model similar to grid infrastructures and did not consider the billing model used in commercial cloud platform, i.e. the hour model; b) a fixed number of resources is considered in the scheduling process; and c) there is no timestamp for release/acquire of each VM resource.

2.2 Multiple Workflow Scheduling Algorithms

In contrast to single workflow scheduling, multiple workflows scheduling has received less attention. Li et al. [12] proposed two level workflow scheduling: the macro multi-workflow scheduling and the micro single workflow scheduling. Workflows are classified into time-sensitive and cost-sensitive based on QoS demands, and different scheduling strategies are adopted in order to meet each QoS time and cost requirements for each workflow type. In [27], authors proposed the Maximize Throughput of Multi-DAG with Deadline (MTMD) algorithm for scheduling concurrent workflow applications in order to improve the ratio of DAGs which can be accomplished within their deadline. Sharif et al. [23] proposed two online multiple workflow scheduling, namely OMPHC-PCPR and OPHC-TR, in Hybrid Cloud Environments. The difference between the two proposed algorithms is the ranking methodology to prioritize tasks during resource allocation.

3 Scheduling background

In this section, we formally describe the QoS workflow scheduling problem on cloud computing infrastructures.

3.1 Application model

Scientific workflow applications are commonly represented by a Directed Acyclic Graph (DAG), a directed graph with no cycles. Formally, a workflow application is a DAG represented by a triple $G =< T, E, data >$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of tasks and n denotes the number of tasks in the workflow application. The set of edges E represent their data dependencies. A dependency ensures that a child node cannot be executed before all its parent tasks finish successfully and transfer the required input data. Let $data$ be a $n \times n$ matrix of communication data, where $data(t_i, t_j)$ is the file size required to be transmitted before task t_j execution from task t_i. The $\overline{C}_{(t_i \to t_j)}$ represents the average transfer time between the tasks t_i and t_j which is calculated based on the average bandwidth and latency among all resources pair. In a given DAG, a task with no predecessors is called an *entry task* and a task with no successors is called an *exit task*. We assume that the DAG has exactly

Table 1 Performance and price of various Amazon EC2 instances.

Instance type	Mean performance [GFLOPS]	Price[$/h]	GFLOPs/$
`m1.small`	2.0	0.1	19.6
`m1.large`	7.1	0.4	17.9
`m1.xlarge`	11.4	0.8	14.2
`c1.medium`	3.9	0.2	19.6

one entry task t_{entry} and one exit task t_{exit}. If a DAG has multiple entry or exit tasks, a dummy entry or exit task with zero weight and zero communication edges is added to the graph.

3.2 Resource model

The target cloud computing platform is composed of a set of m heterogeneous resources $R = \{\cup_{j=1}^{m} r_j \mid r_j \in VM_{type}\}$, that provide services of different capabilities and costs. Each resource includes computation service, e.g. Amazon Elastic Cloud Compute (EC2)[2], and storage service, e.g. Amazon Elastic Block Store (EBS)[3], used as a local storage device for saving the input/output files. In this study, all computation and storage resources are assumed to be in the same data center or region so that average bandwidth between computation resources is considered equal. Notice that the transfer time between two tasks being executed on the same VM is 0. Also, resources are offered in form of different type of virtual machines (VM_{type}). Each VM type has its own configuration for CPU capacity, memory size and an associated cost. Further, it is assumed that there is no limitation of the number of resources (VMs) used by a workflow application, and leasing a VM requires an initial boot time in order to be properly initialized and made available to the user; this time is not negligible and needs to be considered on the scheduling plan [15]. Similarly, on current commercial clouds, the pricing model is based on pay-as-you-go billing model for the number of *time intervals* used by a VM and it is specified by the cloud provider. The user will be charged for each complete *time interval* even if it does not completely use the time interval.

In this study, each resource r_j can be of any type as provided by Amazon EC2 (e.g. `m1.small`, `m1.large`, `m1.xlarge` and `c1.medium`). For a given resource r_j of a certain instance type, the average performance measured in GFLOPs and its price per hour of computation are known. The average performance in GFLOPs of four different Amazon EC2 instance types thorough extensive benchmark experimentation are evaluated in [8]. In our model, we assume that a task executed in any of these resources can benefit from a parallel execution using all the virtual cores exposed by the instance [7]. Also, according to Amazon cloud provider, users are charged based on the time interval of one hour ($interval\ time = 3600\,s$). Table 1 summarises the mean performance, the cost per hour of computation ($Cost_{r_j}$), and the ratio GFLOPs per invested dollar of these resources. Since all resource are located in the same data center or region, the internal data transfer cost is assumed to be zero.

[2] `http://aws.amazon.com/EC2/`
[3] `http://aws.amazon.com/EBS/`

Unlike previous work presented in Section 2, here, we propose an array of release/acquire ($VM_{r/a}$) timestamp for each VM resource which will be updated during the scheduling process. The array of timestamps $VM_{r/a} = \{(S_1, F_1), (S_2, F_2), \ldots\}$ where each pair (S, F) represents the Start and Finish time of consecutively execution of the target VM. These timestamps are calculated based on assigned tasks to the target VM.

During the scheduling process and after making final decision of the appropriate resource (r_{sel}) for execution of the current task (t_{curr}), if the current task could not benefit from last executed task on r_{sel} to reduce its execution cost, i.e. using the remaining last interval from the last previous scheduled task on r_{sel} , the $VM_{r/a}$ of resource r_{sel} is updated in the way that: a) add the release time after execution of the last scheduled task ; b) add the start (acquire) time according to the start time of t_{curr}. Otherwise, the release time of resource r_{sel} will be updated according to the finish time of the current task. Obviously, each resource can be rented for as many times and hours as required for finishing all the tasks scheduled on it. Additionally, we keep the set of scheduled tasks on each resource r_j denoted as $sched_{r_j} = \{t_i \mid AR(t_i) = r_j\}$, where $AR(t_i)$ represents the Assigned Resource on which the task t_i is scheduled to be executed. Each set $sched_{r_j}$ is sorted based on the finish time of its tasks.

3.3 Problem definition

The scheduling problem is defined as finding a map between tasks and resources in order to meet the QoS parameters defined for each job. The problem here consists in finding a schedule map in such a way that the total execution time (makespan) and economical costs are constrained to user's defined values for time and cost. We describe next how the two measures are computed.

3.3.1 Makespan

For computing the total execution time (makespan) of a given workflow, it is necessary to defined the *Time Reservation* (TR) of execution for task t_i on resource r_j as the sum of the execution time of task t_i on resource r_j ($ET(t_i, r_j)$) and the time required for transferring the biggest input data from any parent $t_p \in pred(t_i)$. The information of task execution time (ET) can be gathered via benchmarking or via precise performance models based on existing estimation techniques (e.g. historical data [9] and analytical modelling [17]).

$$TR(t_i, r_j) = \max_{t_p \in pred(t_i)} \left\{ \overline{C}_{t_p \to t_i} \right\} + ET(t_i, r_j)\,. \tag{1}$$

Considering the existence of data transfer time between tasks, for each task t_i to be executed in resource r_j, the resource r_j needs to be deployed before the task t_i starts transferring data from its parent and can only be released after its execution is finished and the data is transferred to its child task. First, we define $avail(r_j)$ as the earliest start Time of task t_i on resource r_j without considering its parents:

$$avail(r_j) = \begin{cases} 0 & , sched_{r_j} = \emptyset \\ FT(t_l, r_j) & , sched_{r_j} \neq \emptyset \end{cases} \tag{2}$$

where t_l is the last task in the sorted tasks scheduled list for resource r_j ($sched_{r_j}$) and $FT(t_l, r_j)$ in the Finish Time of t_l on r_j. Based on $avail(r_j)$, we defined the Release Time of resource r_j ($R_T(r_j)$) as the last rental period of one hour for the last scheduled task on it.

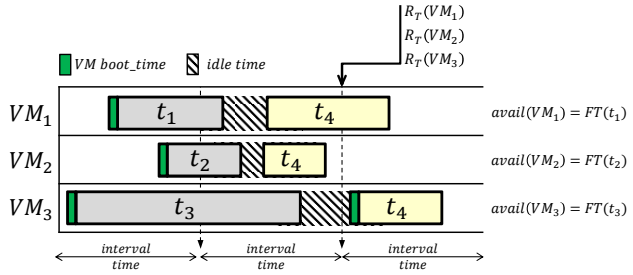

Figure 1 Example of a schedule. Tasks t_1, t_2 and t_3 are scheduled; and task t_4 is evaluated for scheduling.

After that, resource r_j will be released if no other task starts executing on the resource.

$$R_T(r_j) = \left\lceil \frac{avail(r_j)}{interval\ time} \right\rceil \times interval\ time\,. \tag{3}$$

Figure 1 shows a sample schedule generated for scheduled tasks t_1, t_2 and t_3 and current task t_4 which is selected to be scheduled. The Release Time of three resources, calculated by Eq. 3, are indicated as the last interval used by their last scheduled task. Note that, task t_4 is not scheduled and assigned to its target resource yet.

Next, the Start Time (ST) and Finish time (FT) of task t_i on each resource r_j are calculated as:

$$ST(t_i, r_j) = \max\left\{ \max_{t_p \in pred(t_i)} \{FT(t_p)\}, avail(r_j) \right\}, \tag{4}$$

$$FT(t_i, r_j) = \lambda_{(t_i,r_j)} + ST(t_i, r_j) + TR(t_i, r_j) \tag{5}$$

where $\lambda_{(t_i,r_j)}$ is defined as required boot time for acquiring resource instance r_j. If task t_i can be started at last interval time for resource r_j, no boot time required to be considered for task's completion, otherwise, the target resource r_j need to be lunched and its boot time should be considered as a delay in task finish time. For example, in Figure 1, only if task t_4 is scheduled on resource VM_3, a boot time is required to be consider on its finish time because it starts after current release time of VM_3. The $\lambda_{(t_i,r_j)}$ is calculated by:

$$\lambda_{(t_i,r_j)} = \begin{cases} 0 & ,ST(t_i, r_j) < R_T(r_j) \quad \text{OR} \quad sched_{r_j} = \emptyset \\ boot\_time_{r_j} & ,\text{otherwise} \end{cases} \tag{6}$$

where $boot\_time_{r_j}$ is the VM startup/boot time. In this study, we consider $boot\_time_{r_j} = 97\,s$ based on the measurements reported in [15] for the Amazon EC2 cloud. Please note that, during the resource selection phase for each task t_i, the $\lambda_{(t_i,r_j)}$ value is calculated according to the current situation of the target resources r_j, i.e. previous scheduled task on it.

The *makespan* or Schedule length is finally defined as the finish time of the last task of the workflow:

$$\text{DAG}_{makespan} = FT(t_{exit})\,. \tag{7}$$

3.3.2 Financial cost

The financial cost of task t_i on resource r_j is calculated based on the total usage time for complete task execution, considering data transfer time and execution time, and resource

usage price. In this research, we consider Amazon EC2 instance as the platform which makes hour price billing for each instance. In this model, partial hours are rounded up. As a consequence, if other tasks can be executed during that paid interval, they will not be charged for it. We define total usage time of task t_i as the payable period to be charged.

$$pay_{time}(t_i, r_j) = \begin{cases} FT(t_i, r_j) - ST(t_i, r_j) & , R_T(r_j) < ST(t_i, r_j) \\ 0 & , FT(t_i, r_j) < R_T(r_j) \\ FT(t_i, r_j) - R_T(r_j) & , \text{otherwise} \end{cases} \tag{8}$$

The *otherwise* condition will be applied if the task t_i starts before the current release time of resource r_j ($R_T(r_j)$) and finishes after it. So, in this case, the time slice before $R_T(r_j)$ is paid by previous tasks and should not be considered in the current usage time of task t_i.

The pay_{time} equal to zero means that the task can be executed on a previously paid interval (but not fully used) without any additional charge. For example, in Figure 1, if current task t_4 is scheduled on resource VM_2, the $pay_{time}(t_4, VM_2) = 0$. By considering Eq. 8, the pay time for resource VM_1 is equal to $pay_{time}(t_4, VM_1) = FT(t_4, VM_1) - R_T(VM_1)$ and for resource VM_3 we have $pay_{time}(t_4, VM_3) = FT(t_4, VM_3) - ST(t_4, VM_3)$. Please not that, the *boot_time* of resource VM_3 is already considered in $FT(t_4, VM_3)$ by Eq. 5.

The execution cost of task t_i for $pay_{time} > 0$ on resource r_j is computed by:

$$Cost(t_i, r_j) = \left\lceil \frac{pay_{time}(t_i, r_j)}{interval\ time} \right\rceil \times P_{r_j} \tag{9}$$

where P_{r_j} is the associated cost of resource r_j for each usage interval (Table 1, price column). Thus, the overall cost for executing a workflow application is:

$$\text{DAG}_{cost} = \sum_{t_i \in T} \left\{ Cost(t_i, r') \mid t_i \in sched_{r'} \right\} \tag{10}$$

4 Proposed Deadline–Budget workflow Scheduling (DBWS) algorithm

In this section, we present the Deadline-Budget Workflow Scheduling (DBWS) for cloud environments, which aims to find a feasible schedule within a budget and deadline constraints. The DBWS algorithm is a heuristic strategy that in a single step obtains a schedule that always accomplishes the deadline constraint and that may accomplish or not the budget constraint. If the cost constraint is met, we have a successful schedule, otherwise we have a failure and no schedule is produced. The algorithm is evaluated based on the success rate.

Before the description of the DBWS algorithm, next we present the attributes used in the algorithm.

- t_{curr} denotes the current task to be schedule, selected on the task selection phase among all ready tasks;
- r_{sel} denotes the target resource to execute t_{curr} on it;
- $FT_{min}(t_{curr})$ and $FT_{max}(t_{curr})$ denote the minimum and maximum finish time of current task among all tested resources;
- $\ell(t_i)$ denotes the level of task t_i; it is an integer value representing the maximum number of edges of the paths from the entry node to t_i. For the entry node, the level is $\ell(t_{entry}) = 1$, and for other tasks, it is given by:

$$\ell(t_i) = 1 + \max_{t_p \in pred(t_i)} \left\{ \ell(t_p) \right\}. \tag{11}$$

- $Cost_{min}(t_{curr})$ and $Cost_{max}(t_{curr})$ denote the minimum and maximum execution cost of the current task among all tested resources;
- $Cost_{high}(DAG)$ and $Cost_{low}(DAG)$ represent the total execution cost for scheduling target application workflow on the set of homogeneous VMs with highest and lowest cost among all possible VM types in our platform. Here, we use PEFT [3] algorithm to schedule a workflow application.

The DBWS algorithm consists of two phases, namely a *task selection* phase and a *resource selection* phase as described next.

4.1 Task selection

Tasks are selected according to their priorities. To assign a priority to a task in the DAG, the upward rank ($rank_u$) [26] is computed. This rank represents, for a task t_i, the length of the longest path from task t_i to the exit node(t_{exit}), including the computational time of t_i, and it is given by Eq. 12:

$$rank_u(t_i) = \overline{ET}(t_i) + \max_{t_{child} \in succ(t_i)} \left\{ \overline{C}_{t_i \to t_{child}} + rank_u(t_{child}) \right\} \tag{12}$$

where $\overline{ET}(t_i)$ is the average execution time of task t_i over all resources, $\overline{C}_{t_i \to t_{child}}$ is the average communication time between two tasks t_i and t_{child}, and $succ(t_i)$ are the set of immediate successor tasks of task t_i. To prioritize tasks it is common to consider average values because they have to be prioritize before knowing the location where they will run. For the exit node, $rank_u(t_{exit}) = \overline{ET}(t_{exit})$.

4.2 Resource Selection

The target VM to be selected to execute the current task is guided by the following quantities related to cost and time. To select the best suitable resource, a trade-off between these two variables, time and cost, is evaluated. We define a variable, S_{DL} as limit for time. S_{DL} is defined as the sub-deadline that is assigned to each task based on total application deadline. First, all tasks are divided in different levels based on their depth in the graph. We defined level execution ($Level_{exe}$) as the maximum execution length of all tasks in corresponding level and is given by:

$$Level^j_{exe} = \max_{\substack{t_i \in T \\ \ell(t_i) == j}} \left\{ ET_{max}(t_i) + \max_{t_p \in pred(t_i)} \{\overline{C}_{t_p \to t_i})\} \right\} \tag{13}$$

where $ET_{max}(t_i)$ represents the maximum execution time for task t_i among all VM_{type}. In the next step, we distribute the user deadline (D_{user}) among all levels. The sub-deadline value for level j ($Level^j_{DL}$) is computed recursively by traversing the task graph downwards, starting from the first level, as shown below:

$$Level^j_{DL} = Level^{j-1}_{DL} + D_{user} \times \frac{Level^j_{exe}}{\sum_{1 \le j' \le \ell(t_{exit})} Level^{j'}_{exe}}. \tag{14}$$

For the first level ($Level^1_{DL}$), the first part of Eq. (14) is considered zero. Finally, all tasks belonging to the same level have the same sub-deadline.

$$S_{DL}(t_{curr}) = \left\{ Level^j_{DL} \mid \ell(t_i) == j \right\}. \tag{15}$$

Note that, the task's sub-deadline is a soft limit as in most deadline distribution strategies; if the scheduler cannot find a resource (VM) that satisfies the sub-deadline for the current task, the resource that can finish the current task at the earliest time may is selected.

The resource selection phase is based on the combination of the two QoS factors, time and cost, in order to obtain the best balance between time and cost minimum values. We define two relative quantities, namely, Time Quality ($Time_Q$) and Cost Quality ($Cost_Q$), for current task t_{curr} on each resource $r_j \in R \cup R'$, where R represents the set of resources (VM instances) used in previous steps of scheduling, and R' is defined as the set of one temporary resource from each available VM_{type}. At each step after selecting the suitable resource r_{sel} for task t_{curr}, R is updated by $R = \{R \cup r_{sel} \mid r_{sel} \notin R\}$.

Both time and cost quantities are shown in (16) and (17), respectively. Both quantities are normalized by their maximum values.

$$Time_Q(t_{curr}, r_j) = \frac{\xi \times S_{DL}(t_{curr}) - FT(t_{curr}, r_j)}{FT_{max}(t_{curr}) - FT_{min}(t_{curr})} \tag{16}$$

$$Cost_Q(t_{curr}, r_j) = \frac{Cost_{max}(t_{curr}) - Cost(t_{curr}, r_j)}{Cost_{max}(t_{curr}) - Cost_{min}(t_{curr})} \times \xi \tag{17}$$

where

$$\xi = \begin{cases} 1 & \text{if} \quad FT(t_{curr}, r_j) < S_{DL}(t_{curr}) \\ 0 & otherwise \end{cases} \tag{18}$$

$Time_Q$ measures how much closer to the task sub-deadline (S_{DL}) the finish time of current task on resource r_j is. The sub-deadline defines the maximum allowance of task completion time. Consequently, resources with higher $Time_Q$ values, i.e. larger distance between finish time and sub-deadline, have higher possibility to be selected. If the current task has higher finish time on resource r_j than its sub-deadline, $Time_Q$ assumes a negative value for r_j, reducing the possibility for this resource to be selected. Similarly, $Cost_Q$ measures how much less the actual cost on resource r_j is than the maximum execution cost.

In the case that none of the resources can guarantee the current task sub-deadline ($S_{DL}(t_{curr})$), $Cost_Q$ is zero for all of them, and $Time_Q$ for each resource r_j is a negative value that represents the relative finish time obtained with r_j, i.e. a lower finish time causes a lower negative value. And, the resource with higher $Time_Q$, i.e. close to zero, would be selected.

Finally, to select the most suitable resource for current task, the Quality measure (Q) for each resource r_j is computed as shown in Eq. (19):

$$Q(t_{curr}, r_j) \;=\; Time_Q(t_{curr}, r_j) \times (1 - C_F) \;+\; Cost_Q(t_{curr}, r_j) \times C_F \tag{19}$$

where Cost-efficient Factor (C_F) is the tradeoff factor and defined as:

$$C_F = \frac{Cost_{low}(DAG)}{B_{user}}\,. \tag{20}$$

Both $Time_Q$ and $Cost_Q$ parameters are weighted by the ratio of the cheapest cost execution of the whole workflow application ($Cost_{low}(DAG)$) over the user defined available budget (B_{user}), so that the effectiveness of both time and cost factors can be controlled. A lower value of C_F means that the user prefers to pay more to execute the application faster, so that the time quality ($Time_Q$) is more predominant in the resource Quality measure (Q). In the same way, a higher value of C_F means that the user available budget is close

Algorithm 1 DBWS algorithm.

Require: a DAG and user's QoS Parameters values for Deadline (D_{user}) and Budget (B_{user})
1: Compute and sort all tasks based on their upward rank ($rank_u$) value
2: Compute PEFT schedule cost on the resources with cheapest ($Cost_{low}$) and most expensive ($Cost_{high}$) cost from VM_{type}
3: **if** $B_{user} < Cost_{low}(DAG)$ **then**
4: **return** no possible schedule map
5: **else if** $B_{user} > Cost_{high}(DAG)$ **then**
6: **return** PEFT scheduled map on most expensive VM_{type}
7: **end if**
8: Compute the Sub-DeadLine value (S_{DL}) for each task
9: **while** there is an unscheduled task **do**
10: t_{curr} = the next ready task with highest $rank_u$ value
11: **for all** $r_j \in R \cup R'$ **do**
12: Calculate Quality measure $Q(t_{curr}, r_j)$ using Eq. 19
13: **end for**
14: r_{sel} = resource r_j with highest Quality measure (Q)
15: Assign current task t_{curr} to resource r_{sel}
16: Update $R = \{R \cup r_{sel} \mid r_{sel} \notin R\}$
17: Update $VM_{r/a}(r_{sel})$
18: **end while**
19: **return** Schedule Map

to cheapest possible execution cost of the workflow, so that the time quality ($Time_Q$) is inconspicuous while the cost quality ($Cost_Q$) becomes more influential, allowing the selection of more cheap resources that guarantee a lower execution cost for t_{curr}.

The DBWS algorithm is shown in Algorithm 1. After some initializations in lines 1–2, first, the possibility of finding a schedule map under a user defined budget is checked in line 3. Then, the sub-deadline value for each task is computed according Eq. 15 in line 8. The DBWS algorithm starts to map all tasks of the application (while looping in lines 9–18). At each step, on line 10, among all ready tasks, the task with highest priority ($rank_u$) is selected as the current task (t_{curr}). Then, in lines 11–13, the Quality measure for assigning t_{curr} to the resource r_j ($Q(t_{curr}, r_j)$) is calculated. Note that, first, the finish time (FT) and execution cost of the current task is calculated and then the quality measure for all resources is calculated. Next, the resource (r_{sel}) with the highest quality measure among all resources is selected (line 14). Finally, after assigning the current task to the resource, the release/acquire timestamp for the resource r_{sel} is updated as explained in subsection *resource model* in Section 3.

In terms of time complexity, DBWS requires the computation of the upward rank ($rank_u$) and Sub-DeadLines (S_{DL}) for each task that have complexity $O(n.p)$, where p is the number of available resources and n is the number of tasks in the workflow application. In the resource selection phase, to find and assign a suitable resource for the current task, the complexity is $O(n.p)$ for calculating ST and FT for the current task among all resources, plus $O(p)$ for calculating the Quality measure. The total time is $O(n.p + n(n.p + p))$, where the total algorithm complexity is of the order $O(n^2.p)$.

5 Experimental results

This section presents performance comparisons of the DBWS algorithm with four most recently published algorithms, Hybrid [25], MTCT (Min-min based time and cost tradeoff)

[29], CwFT (Cost with Finish Time-based) [14] scheduling algorithms and SABA (Security-Aware and Budget-Aware) [30]. We choose MTCT[29] for comparison because it outperforms LOSS algorithms [21] and IC-PCP[1].

5.1 Budget and deadline parameters

To evaluate the DBWS algorithm, the user budget (D_{user}) and deadline (B_{user}) parameters assume values in a range so that the constraints are feasible. To specify these parameters, boundary values are defined for each of them, by using PEFT scheduling algorithm. We calculate the total execution time (makespan) of the workflow scheduled on the set of homogeneous VM instances with highest and lowest associated cost as the minimum (min_D) and the maximum (max_D) deadline boundary value. In the same way, the corresponding execution costs are the maximum (max_B) and minimum (min_B) execution cost of the workflow application. With these highest and lowest bound values, we define for the current application a unique Deadline and Budget constraint, as described by Eqs. (21) and (22):

$$D_{user} = min_D + \alpha_D \times (max_D - min_D) \tag{21}$$

$$B_{user} = min_B + \alpha_B \times (max_B - min_B) \tag{22}$$

where the deadline parameter α_D and budget parameter α_B can be selected in the range of $[0 \dots 1]$. In this paper, to observe the ability of finding valid schedule maps, we selected a low set of values, $\{0.1, 0.3, 0.5\}$, for time and cost parameters (α_D and α_B) to test the performance of each algorithm on harder conditions. Undoubtedly, increasing values for α_D and α_B, we would be able to achieve higher successful percentage rates.

5.2 Performance metric

To evaluate and compare our algorithm with other approaches, we consider the Planning Successful Rate (PSR), as expressed by Eq. (23). This metric provides the percentage of valid schedules obtained in a given experiment.

$$\text{PSR} = 100 \times \frac{\textit{Number of Successful Planning workflows}}{\textit{Total Number of workflows in experiment}}. \tag{23}$$

In addition, to investigate the quality of results, we compute the ratio of deadline defined and makespan achieved (NM) for each workflow, as well as the ratio of budget and execution cost (NB) of the schedule produced, as described in Eqs. (24) and (25):

$$\text{NM} = \frac{D_{user}}{\text{DAG}_{makespan}}, \tag{24}$$

$$\text{NB} = \frac{B_{user}}{\text{DAG}_{cost}}. \tag{25}$$

Note that, both metrics NM and NB are calculated for each schedule map, even for not successful ones, achieved by the algorithm. Basically, a lower value than 1 for NM and NB metrics means that the schedule map could not meet the constraint values for time and cost, respectively.

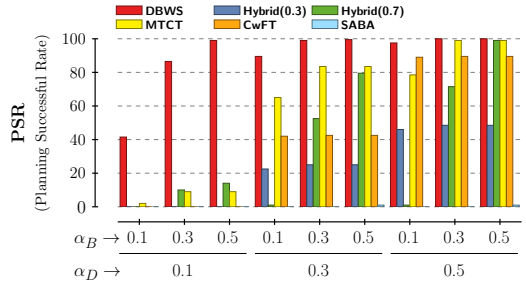

Figure 2 PSR value for CyberShake.

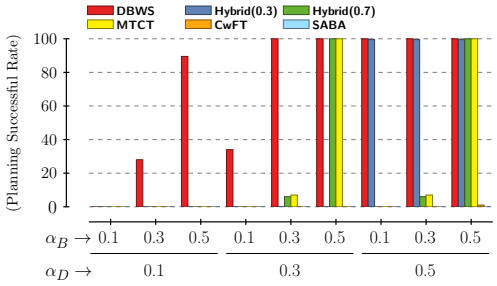

Figure 3 PSR value for Epigenomic.

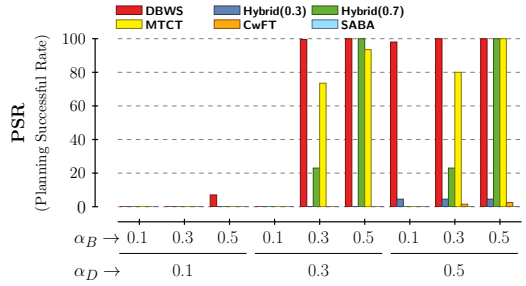

Figure 4 PSR value for Ligo.

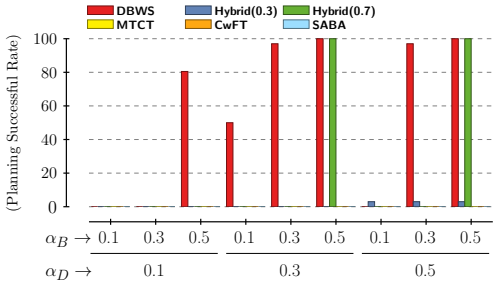

Figure 5 PSR value for Montage.

5.3 Results and discussion

To evaluate the algorithms on a standard and real set of workflow applications, a set of workflows were generated using the code developed in Pegasus toolkit[4]. Four well known structures were chosen [10], namely: CyberShake, Epigenomic, Ligo and Montage. The workflows are characterized as CPU-bound (Epigenomic), I/O-bound (Montage), data-intensive (CyberShake); workflows with large memory requirements (CyberShake, Ligo); and workflows with large resource requirements (CyberShake, Ligo). For each type of these real world workflows, we generated 1000 DAGs with a number of tasks equal to 50, 300 and 1000 tasks.

The original implementation of the compared scheduling algorithms assumed a fixed number of resources during the schedule map. In our implementation of those algorithms, we assigned an initialized fixed number of resources equal to the maximum number of concurrent tasks among all levels in the workflow application. Also, to apply the cost consumption during the scheduling process, we consider the same approach to calculate the cost execution of each task (Eq. 9) for all algorithms. For the Hybrid [25] scheduling algorithm, we consider two versions, namely Hybrid ($\alpha = 0.3$) and Hybrid($\alpha = 0.7$), where the α parameter represents the user's preference for minimizing the execution time or the monetary cost, i.e lower α corresponds to less monetary cost.

Figures 2, 3, 4 and 5 show the average Planning Successful Rate (PSR) obtained for the real workflow application considered here. The main result is that the algorithm DBWS obtains good performance in comparison to other state-of-the-art heuristic-based algorithms, for the range of budget and deadline values considered here. By increasing the budget factor (α_B), more budget is available to run the workflow resulting in an increase of the PSR value

[4] `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`

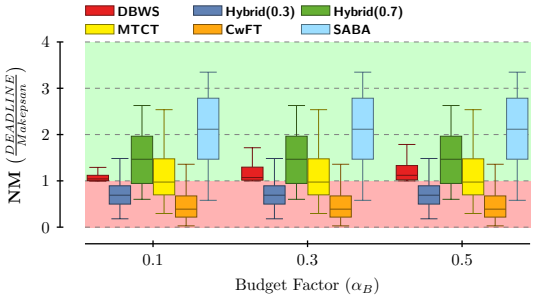

Figure 6 Deadline-makespan ratio.

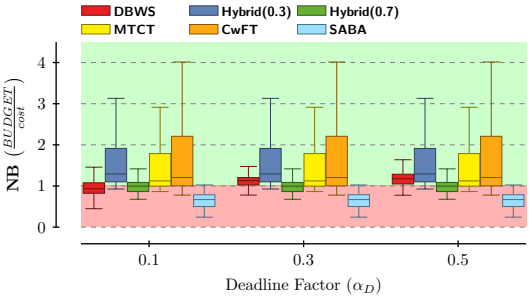

Figure 7 Budget-schedule cost ratio.

for DBWS. Note that for shorter deadlines ($\alpha_D = 0.1$) in the most cases only DBWS is able to complete workflows (Figures 3, 4 and 5).

Figures 6 and 7 represent the ratios related to time and cost, obtained by each algorithm. In other to have a better presentation, NM and NB values are divided into two main categories, where *safe* is represented by the green color and *unsafe* is represented by the red color. A value greater than 1 for the NM metric (Eq. 24) means that the algorithm could obtain a scheduled makespan lower than the user defined deadline. But a value $NM < 1$ means that the algorithm failed to find the schedule map with a makepsan lower than the user defined deadline. The same explanation can be considered for NB metric (Eq. 25). As it is shown in Figure 6, the makespan of the schedule map obtained by proposed DBWS algorithm always meet the user defined deadline constraint value for all range of budget factor α_B. However, for the total cost execution in Figure 7, by decreasing the deadline factor α_D, the execution cost of the schedule map obtained by DBWS becomes higher than the user defined budget. Note that the PSR values represented in Figures 2–5 represent the percentage of workflows for which the schedule meets both time and cost constraint values. For example, despite of the best reduction in execution cost by CwFT scheduling algorithm in Figure 7, due to failing in meeting the deadline value (Figure 6), the CwFT approach shows the worst performance in terms of PSR value (Figure 2–5). Also, for SABA scheduling algorithm, it fails in most cases as shown by the PSR metric. The reason can be explained due to the strategy used for VM assignment, namely the Comparative Factor (CF). The CF approach used the trade-off between time and cost factors and did not control the cost consumption during the scheduling process. As seen from Figures 6 and 7, SABA shows the best performance in total execution time reduction and worst one for the total cost.

6 Conclusions and future work

In this paper, we present the Deadline-Budget Workflow Scheduling (DBWS) algorithm for cloud environments, which maps a workflow application to cloud resources constrained to user-defined deadline and budget values. The algorithm was compared with other state-of-the-art heuristic-based scheduling algorithms. In terms of time complexity, which is a critical factor for effective usage on real platforms, our algorithm has quadratic time complexity. In terms of the quality of results, DBWS achieves better rates of successful schedules compared to other heuristic-based approaches for the real world applications considered. For the range values of deadline and budget constraints considered in this paper, DBWS shows a significant improvement of the planning successful rate for the workflows and cloud platform considered.

In conclusion, we have presented the DBWS algorithm for budget and deadline constrained scheduling, which has proved to achieve better performance than other heuristic-based approaches, namely Hybrid[25] MTCT[29] and CwFT[14].

In future work, we intend to extend the algorithm to consider dynamic concurrent workflow applications which can be submitted by any user at any time.

References

1 Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H. J. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

2 Ehab Nabiel Alkhanak, Sai Peck Lee, and Saif Ur Rehman Khan. Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities. *Future Generation Computer Systems*, 50(0):3–21, 2015.

3 Hamid Arabnejad and Jorge G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):682–694, 2014.

4 Kahina Bessai, Samir Youcef, Ammar Oulamara, Claude Godart, and Selmin Nurcan. Bi-criteria workflow tasks allocation and scheduling in cloud computing environments. In *5th Int. Conf. on Cloud Computing*, pages 638–645. IEEE, 2012.

5 Rodrigo N. Calheiros and Rajkumar Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *Parallel and Distributed Systems, IEEE Transactions on*, 25(7):1787–1796, 2014.

6 Wei-Neng Chen and Jun Zhang. A set-based discrete PSO for cloud workflow scheduling with user-defined QoS constraints. In *Int. Conf. on Systems, Man, and Cybernetics (SMC)*, pages 773–778. IEEE, 2012.

7 Juan J. Durillo and Radu Prodan. Multi-objective workflow scheduling in amazon ec2. *Cluster Computing*, 17(2):169–189, 2014.

8 Alexandru Iosup, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick H. J. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, 2011.

9 S. Jang, Xingfu Wu, Valerie Taylor, Gaurang Mehta, Karan Vahi, and Ewa Deelman. Using performance prediction to allocate grid resources. Technical report, Technical Report 2004-25, GriPhyN Project, USA, 2004.

10 Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

11 Young Choon Lee and Albert Y. Zomaya. Stretch out and compact: Workflow scheduling with resource abundance. In *13th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, pages 219–226. IEEE, 2013.

12 Wenjuan Li, Qifei Zhang, Jiyi Wu, Jing Li, and Haili Zhao. Trust-based and qos demand clustering analysis customizable cloud workflow scheduling strategies. In *Cluster Computing Workshops*, pages 111–119. IEEE, 2012.

13 Xiangyu Lin and Chase Qishi Wu. On scientific workflow scheduling in clouds under budget constraint. In *42nd International Conference on Parallel Processing (ICPP)*, pages 90–99. IEEE, 2013.

14 Nguyen Doan Man and Eui-Nam Huh. Cost and efficiency-based scheduling on a general framework combining between cloud computing and local thick clients. In *Int. Conf. on Computing, Management and Telecommunications*, pages 258–263. IEEE, 2013.

15 Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *5th International Conference on Cloud Computing*, pages 423–430. IEEE, 2012.

16 Sahar Mirzayi and Vahid Rafe. A hybrid heuristic workflow scheduling algorithm for cloud computing environments. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(6):721–735, 2015. doi:10.1080/0952813X.2015.1020524.

17 Graham R. Nudd, Darren J. Kerbyson, Efstathios Papaefstathiou, Stewart C. Perry, John S. Harper, and Daniel V. Wilcox. Pace – a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

18 D. Poola, S. K. Garg, R. Buyya, Yun Yang, and K. Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *28th Int. Conf. on Advanced Information Networking and Applications*, pages 858–865. IEEE, 2014.

19 Mustafizur Rahman, Xiaorong Li, and Henry Palit. Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment. In *Int. Symp. on Parallel and Distributed Processing (IPDPSW)*, pages 966–974. IEEE, 2011.

20 Jyoti Sahni and Deo Vidyarthi. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

21 Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D. Dikaiakos. Scheduling workflows with budget constraints. In *Integrated research in GRID computing*, pages 189–202. Springer, 2007.

22 S. Selvarani and G. Sudha Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *Int. Conf. on Computational intelligence and computing research*, pages 1–5. IEEE, 2010.

23 Shaghayegh Sharif, Javid Taheri, Albert Y. Zomaya, and Surya Nepal. Online multiple workflow scheduling under privacy and deadline in hybrid cloud environment. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 455–462. IEEE, 2014.

24 Sucha Smanchat and Kanchana Viriyapant. Taxonomies of workflow scheduling problem and techniques in the cloud. *Future Generation Computer Systems*, 52:1–12, 2015.

25 Sen Su, Jian Li, Qingjia Huang, Xiao Huang, Kai Shuang, and Jie Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4):177–188, 2013.

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

27 Wei Wang, Qingbo Wu, Yusong Tan, and Fuhui Wu. Maximize throughput scheduling and cost-fairness optimization for multiple dags with deadline constraint. In *Algorithms and Architectures for Parallel Processing*, pages 621–634. Springer, 2015.

28 Chase Qishi Wu, Xiangyu Lin, Dantong Yu, Wei Xu, and Li Li. End-to-end delay minimization for scientific workflows in clouds under budget constraint. *Cloud Computing, IEEE Transactions on*, 3(2):169–181, 2015.

29 Heyang Xu, Bo Yang, Weiwei Qi, and Emmanuel Ahene. A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery. *KSII Transactions on Internet and Information Systems*, 10(3):976–995, 2016.

30 Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. Saba: A security-aware and budget-aware workflow scheduling strategy in clouds. *Journal of Parallel and Distributed Computing*, 75:141–151, 2015.

31 Lingfang Zeng, Bharadwaj Veeravalli, and Albert Y. Zomaya. An integrated task computation and data management scheduling strategy for workflow applications in cloud environments. *Journal of Network and Computer Applications*, 50:39–48, 2015.