Task-based Budget Distribution Strategies for Scientific Workflows with Coarse-grained Billing Periods in IaaS Clouds

Muhammad Hafizhuddin Hilman, Maria A. Rodriguez and Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
Email: hilmanm@student.unimelb.edu.au, {marodriguez, rbuyya}@unimelb.edu.au

***Abstract*—The use of cloud computing, particularly of Infrastructure as a Service clouds, for the execution of large-scale scientific workflows has been a topic of interest in recent years. These environments offer on-demand access to all of the infrastructure required for the deployment of workflows, allowing users to pay only for what they use. This leads to schedulers having to find a trade-off between two conflicting quality of service requirements: time and cost. The majority of research in this area has focused on developing scheduling algorithms that have as objective minimizing the infrastructure cost while meeting a deadline constraint. Few algorithms, however, have addressed the problem of minimizing the execution time of the workflow while meeting a budget constraint. This paper focuses on the latter case. We propose a budget-distribution algorithm that assigns a portion of the overall workflow budget to the individual tasks. This task-level budget then guides the dynamic scheduling process and is continuously refined to reflect any unexpected costs. When compared to the state-of-the-art algorithm, the performance evaluation results demonstrate that in 88% of the cases, our proposal achieves equal or better performance in terms of meeting the budget constraint and achieves lower execution times in 84% of the cases.**

***Keywords*-budget distribution; task-based; coarse-grained billing period; scientific workflow**

I. Introduction

Modern scientific instruments can collect vast amounts of data that enable scientists to conduct more meaningful and precise analyses and simulations. These scientific experiments are commonly expressed as workflows, that is, applications that are composed of multiple computational tasks with dependencies between them. Such scientific workflows are large-scale applications and require extensive computational resources to process their input data in a reasonable amount of time. Thus, they are commonly deployed on distributed systems, in particular, cloud computing has become a popular platform for this purpose in recent years.

Infrastructure as a Service (IaaS) clouds offer a convenient way for workflow management systems to access computational resources. Specifically, they provide access to Virtual Machines (VMs) of different types. These VMs can be accessed on-demand, they can be leased when they are needed and released when they are not. Users are charged only for what they use, usually in increments of a billing period defined by the provider. This elasticity makes IaaS environments ideal platforms for the execution of scientific workflows; the number of VMs and their types can be easily adjusted to match the characteristics and number of workflow tasks that need to be executed at any given point in time.

However, this flexibility and ability to easily scale the number of resources leads to a trade-off between two conflicting Quality of Service (QoS) requirements: time and cost. The reason is straightforward, more powerful VMs capable of processing a task faster will be more expensive than slower, less powerful ones. Thus, provisioning algorithms that can decide the type and number of VMs required by a workflow execution and scheduling algorithms capable of efficiently mapping the tasks to the resources while considering time and cost are essential. There has been extensive research [1] on this topic, with most works proposing algorithms that aim to minimize the total execution cost while finishing the workflow execution before a user-defined deadline. In this work, we focus on optimizing the usage of resources so that the total execution time of the workflow (i.e., makespan) is minimized while meeting a budget constraint.

Various strategies can be used to achieve these scheduling objectives when deploying workflows in IaaS clouds. A popular one is using meta-heuristics to produce a static mapping of tasks to resources in advance. In this way, an estimate of the total cost and makespan of the workflow execution is known as well as the required resources and their leasing period. This technique, however, is computationally intensive and does not scale well with the number of tasks in the workflow. Also, because the schedule is produced before runtime and remains unchanged throughout the execution, these algorithms are unable to adapt to the inherent dynamicity and uncertainty of IaaS clouds environment. Other algorithms use lighter-weight heuristics to produce static schedules to address the scalability issue. However, they still fail to respond to environmental changes. Some strategies choose to dynamically schedule the tasks as they become ready for execution to overcome the responsiveness issue. This dynamic approach enables the algorithm to scale easily and to adapt and make decisions based on the state of the system. Since the scheduling is task-based, the overall workflow budget must be distributed to each task. This budget allocation guides the scheduling process as it determines the type of resources that can be allocated to each task as well as the time when they should be deployed. Budget distribution is a challenging problem mainly due to the pricing model offered by IaaS clouds providers.

It is not uncommon for the average execution time of workflow tasks to be considerably smaller than the coarse-grained billing periods (e.g., one hour) offered by most IaaS vendors. Thus, scheduling algorithms aim to efficiently utilize idle time slots on leased VMs as a cost-controlling mechanism. Coarse-

grained billing periods make it difficult to estimate the portion of the budget that should be allocated to each task; the main reason being that the cost of a single task must be overestimated by either rounding up its execution time to one (or more) billing periods or (potentially) underestimated by determining their cost in time units. Deciding how to factor VM provisioning delays when estimating the costs of tasks is another significant challenge. Some algorithms choose to consolidate tasks per workflow level and assign a corporate budget to them to be spent greedily to avoid time slots wastage. Depending on how the budget is split, this may result in the insufficient budget allocated to some levels, violating budget constraints due to tasks in a level taking longer to execute, and inefficient use of the budget. In this paper, we explore distributing the budget to each task by rounding their cost to billing periods. We argue that this enables the algorithm to spend the budget more efficiently as it has a better awareness of the remaining budget and hence can better utilize it. Furthermore, such an algorithm can respond faster to unexpected delays. Also, to avoid underutilizing resources, this strategy is combined with policies that encourage the reuse of time slots in already-leased VMs.

Thus, we focus on distributing a portion of the budget to individual tasks and spending it only when necessary, that is when free idle time slots on existing VMs cannot be reused. Our approach considers inherent features of IaaS clouds such as the abundance of heterogeneous computing resources, VM provisioning delays, and the dynamic and uncertain behavior of VMs performance. Our solution consists of two components, a budget distribution strategy and a scheduling one. For the budget distribution, we propose an algorithm with two variants, namely Fastest-First Task-based Distribution (FFTD) and Slowest-First Task-based Distribution (SFTD). For the scheduling, we adapt EPSM [2], an existing approach designed to schedule multiple workflows with deadline constraints. We modify it so that it considers a single workflow and aims to complete its execution as fast as possible with the given budget. Our simulation results demonstrate that our algorithm can adapt to unexpected delays and meet the budget constraint while achieving lower makespans compared to the state-of-the-art budget distribution algorithm.

The rest of this paper is organized as follows. Section II reviews works that are related to our paper. Section III describes the considered resource and application models. The proposed algorithms are explained in section IV followed by their performance evaluation and a discussion of the results in section V. Finally, the conclusions and future work are depicted in Section VI.

II. Related Work

The scheduling of scientific workflows in IaaS clouds has been extensively researched. The majority of existing algorithms have as objective to meet a deadline constraint and minimize the cost of renting the infrastructure. Examples include the solutions presented by Mao and Humphrey [3], Abrishami et al. [4], Malawski et al. [5], Arabnejad et al. [6], Cai et al. [7] and Chen et al. [8].

Table. I: Summary of Related Work

Strategies	[9]	[10]	[11]	[12]	[13]	[14]	[15]	Ours
Static Heuristic	✓	✓			✓			
Static Metaheuristic			✓	✓				
Dynamic Level-based						✓	✓	
Dynamic Task-based								✓

Only a few of the existing algorithms focus on meeting a budget constraint while minimizing the workflow makespan. An example is the Partial Critical Paths Budget Balanced (PCP-B^2) [9] algorithm. It partitions a workflow into pipelines of partial critical paths and finds the optimal resource type that maximizes the budget utilization. Contrary to our work, PCP-B^2 assumes a time unit pricing model as opposed to the more common model of billing periods. The Critical-Greedy [10] algorithm finds a workflow's schedule by iteratively refining an initial schedule plan that encourages the use of more powerful VM types if there is budget remaining. Other works with the same objectives use Particle Swarm Optimization [11] and Genetic Algorithms [12] to develop a static plan that minimizes the makespan before runtime. These algorithms rely on calculating a near-optimal schedule by using computationally intensive meta-heuristic techniques. The strategy differs from our solution in that we use a lightweight, adaptive, heuristic-based dynamic approach that makes scheduling and resource provisioning decisions at runtime based on the state of the system. The DBD-CTO [13] algorithm also considers budget as a constraint, however, contrary to our solution, deadline is also a part of its scheduling objectives.

BAGS [14] is another example of existing budget-constrained algorithm. It partitions the workflow into bags of tasks (BoTs) that are on the same workflow level. BAGS is based on an online budget distribution strategy that guides the resource provisioning and scheduling plans of BoTs dynamically, as tasks become ready for execution. However, contrary to us, BAGS assumes fine-grained billing periods (e.g., one minute) that are not much longer than the average execution time of tasks. Finally, the Budget Distribution Trickling (BDT) [15] algorithm uses a similar strategy by consolidating tasks on the same workflow level. The budget is distributed to each level, and the algorithm trickles down any remaining budget to the next level. It assumes an hourly billing period but ignores the performance variation of VMs. The authors of BDT explore several level-based budget distribution strategies based on characteristics such as the number of tasks in the level and the number of levels in the workflow. The algorithms differ from ours in that our solution schedules tasks independently when they are ready for execution, that is, whenever the task's parents have finished executing, and the input data is available. We present the summary of discussed works in Table I.

III. Application and Resource Model

Our work is designed to schedule scientific workflows that are modelled as Directed Acyclic Graphs (DAGs). A workflow W consists of a set of tasks $T = (t_1, t_2, \ldots, t_n)$ and a set of directed edges $E = (e_{12}, e_{13}, ..., e_{mn})$ in which an edge e_{ij}

represents data dependency between task t_i (parent task) and task t_j (child task). Hence, t_j will only be ready for execution after t_i has completed. In addition, we assume the size of a task S_t, is available to the scheduler and is measured in Millions of Instructions (MI).

VMs are leased using an on-demand pricing model and are charged per billing period bp, with any partial usage being rounded up to the nearest billing period. Our work considers a heterogeneous environment with various VM types vmt that have different processing capacity PC_{vmt} and different cost per billing period c_{vmt}. The processing capacity of a VM is measured in Million of Instruction per Second (MIPS). We assume the CPU performance of VMs is not stable as reported by Leitner and Cito [16] and that providers advertise the maximum CPU capacity achievable by VMs. Furthermore, we assume an unlimited number of VMs can be leased from the provider.

The runtime of a task t in a VM of type vmt is denoted as RT^t_{vmt} and is calculated based on the size S_t of the task and the processing capacity PC_{vmt} of the VM. This is shown in Eq. 1. Note that this value is an estimate and our approach does not rely on it being completely accurate. Also, we assume that VMs with more CPU capacity are more expensive to lease than VMs with less capacity. In this way, the task runtime estimated using the cheapest VM type leads to the largest value (slowest runtime) but potentially the lowest cost.

$$RT^t_{vmt} = S_t / PC_{vmt} \tag{1}$$

$$B_{vmt}(t) = (B_{vmt} / Tr_t) \tag{2}$$

$$GS_{read}(t) = (GS_{read} / Tr^{read}_t) \tag{3}$$

$$GS_{write}(t) = (GS_{write} / Tr^{write}_t) \tag{4}$$

We consider a global storage system such as Amazon S3 for data sharing between tasks. Each task retrieves its input data D^t_{in} from the global repository and stores its output data D^t_{out} on the same. The read and writing speeds of the global storage are GS_{read} and GS_{write} respectively. Additionally, each VM has a bandwidth B_{vmt} associated with it. This bandwidth and the I/O speeds of the storage system change over time, based on the number of transactions Tr running at time t. This is depicted in Eqs. 2, 3 and 4. The time it takes to retrieve the input data from the storage to a VM is shown in Eq. 5. Similarly, the time it takes to transfer the output data from a VM into the storage is shown in Eq. 6.

$$T^{D^t_{in}}_{vmt} = (D^t_{in} / B_{vmt}) + (D^t_{in} / GS_{read}) \tag{5}$$

$$T^{D^t_{out}}_{vmt} = (D^t_{out} / B_{vmt}) + (D^t_{out} / GS_{write}) \tag{6}$$

$$PT^t_{vmt} = RT^t_{vmt} + T^{D^t_{in}}_{vmt} + T^{D^t_{out}}_{vmt} \tag{7}$$

$$C^t_{vmt} = \lceil (PT^t_{vmt} + T_{pdelay} + T_{ddelay}) / bp \rceil * c_{vmt} \tag{8}$$

We consider a model in which the global storage system and VMs are located in the same region or availability zone. Hence, data transfer between storage and VMs is free of charge, as is the case for most IaaS providers. Nevertheless, we assume that the output data of a task is also stored on the VM's local storage. In this way, child tasks executing on the same VM do not need to read their input data from the global storage. By implementing this mode, the amount of time spent transferring data can be considerably reduced. Hence, the total processing time PT^t_{vmt} of a task on a VM of type vmt is shown in Eq. 7. Furthermore, the cost C^t_{vmt} of a task that runs on a VM of type *vmt* considering the VMs provisioning delay T_{pdelay} and deprovisioning delay T_{ddelay} is shown in Eq. 8.

IV. Scheduling Algorithm

Our algorithm consists of two phases:

1) Budget Distribution: The workflow's budget is distributed to each task using two approaches, FFTD, and SFTD.
2) Resource Provisioning and Scheduling: The tasks are selected based on the ascending order of their Earliest Finish Time (EFT) and the resources are provisioned based on the task's sub-budget whenever there are no idle VMs to reuse.

Our resource provisioning and scheduling strategy is based on the EPSM algorithm [2], a dynamic heuristic-based algorithm for Workflow-as-a-Service (WaaS) frameworks that can handle a continuously arriving workload of heterogeneous workflows. The algorithm's objective is meeting the deadline constraint of each workflow while minimizing the overall cost. To achieve this, EPSM uses containers as means to reuse VMs across different workflows. We modify this algorithm to consider budget as a constraint and schedule a single workflow, eliminating the need for containers.

A. Budget Distribution

The amount of available budget drives the scheduling process. It will determine the resources that can be used to run a task and hence will have a direct impact on the workflow's makespan. Our strategy is based on the intuitive idea that by choosing the fastest resources that are affordable within the budget, the probability that a task's runtime exceeds the billing period of a VM because of performance degradation is decreased. In this way, the probability of having higher costs and exceeding the budget is also decreased as the chances of incurring in additional billing periods are reduced.

This budget distribution is a challenging problem, mainly due to the coarse-grained billing periods enforced by IaaS vendors. For instances, in some cases, assigning a budget to a task based solely on its runtime estimation without considering billing periods, may lead to budget insufficiency, a condition in which the task's sub-budget is not enough to provision a new VM for it (since the cost of the VM is larger than the estimated cost).

Consider the illustration in Fig. 1. Suppose there are two VM types available, a small type which costs \$1/hour and large type that costs \$3/hour. Then, we have a workflow that consists of seven tasks with the estimated runtime for each task being $RT_A = 100s$, $RT_B = 400s$, $RT_C = 400s$, $RT_D = 200s$, $RT_E = 100s$, $RT_F = 100s$, and $RT_G = 100s$. The budget for this workflow is \$7. If we distribute the budget based on the runtime of each task and ignore the VM billing period, the budget for task A, E, F and G will be insufficient because

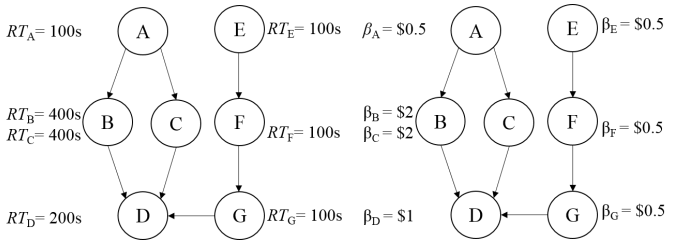


Fig. 1: Sample of budget insufficiency scenario

of the allocated budget ($0.5) would be less than the cost of small type VM ($1). Since task A and E are the entry tasks of workflow, if their sub-budgets are insufficient to provision the resources, the workflow cannot be further executed.

Our budget distribution algorithm is based on the execution order of tasks. Entry tasks in the first level of the workflow are executed first, followed by their children on the next level, and so on. Hence, we assign each task a level based on the Deadline Top Level (DTL) [17] technique as seen in Eq. 9, as opposed to Deadline Bottom level (DBL) [18] which starts the level allocation from the exit task.

$$level(t) = \begin{cases} 0 & ifPred(t) = \emptyset \\ \max\limits_{p \in Pred(t)} level(p) + 1 & otherwise \end{cases} \quad (9)$$

Let's consider the example of Fig. 1, DTL allocates task A and task E to the same level (1) while DBL assigns the tasks to a level starting from task D as level (1). Consequently, using DBL, task A is assigned to level (3) which would be different to the level of task E (4). Although a workflow is, of course, being processed starting from the entry tasks, allocating them into different levels will cause have an impact on those algorithms that execute the tasks based on their level. Also, to determine the order of tasks in a level, we sort them based on the ascending order of their Earliest Finish Time (EFT) as shown in Eq. 10.

$$eft(t) = \begin{cases} PT^t_{vmt} & ifPred(t) = \emptyset \\ \max\limits_{p \in Pred(t)} eft(p) + PT^t_{vmt} & otherwise \end{cases} \quad (10)$$

The tasks in the workflow in Fig. 1 would be sorted in the following order: A [level(1)] → E [level(1)] → F [level(2)] → B [level(2)] → C [level(2)] → G [level(3)] → D [level(4)]. The budget distribution algorithm then iterates over this sorted list and distributes the budget to each task while considering the task's estimated runtime and the cost per billing period of each VM type. As a result, the sub-budget allocated to a task is equivalent to at least the cost of one full billing period. With this approach, we are overestimating the cost of a task and as a result, it is possible that several tasks will not get any sub-budget allocation. In these cases, the algorithm will delay the tasks with no budget so that they can reuse existing idle VMs. The budget distribution algorithm is shown in Algorithm 1.

The algorithm uses two approaches to estimate the task sub-budget based on the VM type chosen, namely Fastest-First Task-based Budget Distribution (FFTD) and Slowest-First

Algorithm 1 Budget Distribution

```
1:  procedure DISTRIBUTEBUDGET(β, T)
2:      S = task's estimated execution order
3:      for each task t ∈ T do
4:          allocateLevel(t, l)
5:          initiateBudget(0, t)
6:      for each level l do
7:          T_l = set of all tasks in level l
8:          sort T_l based on ascending Earliest Finish Time (EFT)
9:          put(T_l, S)
10:     while β > 0 do
11:         t = S.poll
12:         vmt = chosen VM type
13:         allocateBudget(C^t_vmt, t)
14:         β = β − C^t_vmt
```

Task-based Budget Distribution (SFTD). The FFTD approach selects the fastest VM type that is affordable within the workflow's budget. Since the algorithm allocates the fastest resources to the earlier tasks, their successor's will have the opportunity to reuse these VMs which may be faster than what they can actually afford. Hence, it also increases the possibility of obtaining lower task processing times, even though it may involve a waiting delay for the VMs to become available. On the contrary, the SFTD approach chooses the cheapest resources for the tasks. Whenever there is any additional remaining budget after all tasks are allocated sub-budgets, the algorithm uses this extra budget to greedily lease VMs with more CPU capacity than what is affordable by the individual task's sub-budget. SFTD ensures most of the tasks get a portion of the budget allocated, so they do not need to wait for the VMs to become idle. In both approaches, allocating resources to the entry tasks guarantees execution of the workflow.

B. Resource Provisioning and Scheduling

Once the sub-budgets are assigned to each task, the algorithm processes entry tasks of the workflow and puts them into the queue sorted by their EFT in ascending order. Then it schedules each task in the queue in the following way. First, the algorithm tries to reuse an idle VM that has the input data of the task in its cache. If such a VM exists, then the task is assigned to it, always favouring VMs in which executing the task would lead to the lowest cost. Notice that the lowest cost ($0) is achieved when a VM can finish the task before its next billing period. If there is no idle VM with cached input data, then the algorithm tries to reuse any currently idle VM. If no idle VMs are available, then a new VM of the fastest type that is affordable with the task's sub-budget is provisioned. The resource provisioning and scheduling algorithm are shown in Algorithm 2.

After a task is completed, the algorithm updates the budget distribution of all the remaining tasks to reflect the budget spent so far. Also, it maintains any unused sub-budgets of the tasks that have reused idle VMs as a spare budget. This spare budget is used either when updating the budget distribution or in the provisioning phase to lease faster VM types. The budget-updating algorithm can be seen in Algorithm 3.

Algorithm 2 Resource Provisioning and Scheduling

procedure SCHEDULEQUEUETASKS(q)
  sort q by ascending Earliest Finish Time (EFT)
  sb = spare budget
  while q is not empty **do**
    $t = q.poll$
    $vm = null$
    $delayFlag =$ false
    if there are idle VMs **then**
      VM_{idle} = set of all idle VMs
      VM_{idle}^{input} = set of $vm \in VM_{idle}$ that have t's input data
      $vm = vm \in VM_{idle}^{input}$ that can finish t with minimum risk of incurring a new billing period
      if $vm = null$ **then**
        $vm = vm \in VM_{idle}$ that can finish t with minimum risk of incurring a new billing period
    else
      vmt = cheapest VM type
      if $t.budget < C_{vmt}^{t}$ **then**
        $delayFlag =$ true
      if $delayFlag =$ false **then**
        vmt = fastest VM type within *t.budget*
        if there are faster VM type than vmt AND *sb* is enough **then**
          $vmt = leaseFasterVMT()$
        $vm = provisionVM(vmt)$
    $scheduleTask(t, vm)$

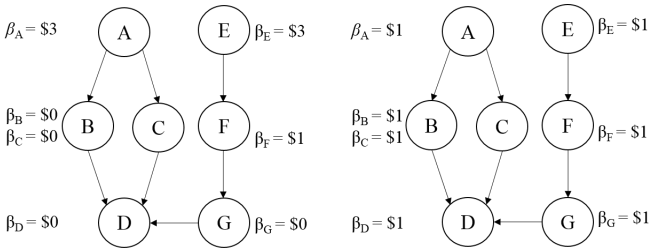
Algorithm 3 Budget Update

procedure UPDATEBUDGET(T)
  t_f = completed task
  T_c = set of $t \in T$ that are children of t_f
  β_c = total sum of *t.budget*, where $t \in T_c$
  T_u = set of unscheduled $t \in (T - T_c)$
  β_u = total sum of *t.budget*, where $t \in T_u$
  sb = spare budget
  if $C_{vmt}^{t_f} \leq (t_f.budget + sb)$ **then**
    $sb = (t_f.budget + sb) - C_{vmt}^{t_f}$
  else
    $debt = C_{vmt}^{t_f} - (t_f.budget + sb)$
    $\beta_c = \beta_c - debt$
    DISTRIBUTEBUDGET(β_c, T_c)
    if $\beta_c < 0$ **then**
      $\beta_u = \beta_u + \beta_c$
      DISTRIBUTEBUDGET(β_u, T_u)

C. Illustrative Example

This section explains how the workflow that is shown in Fig. 1 would be scheduled using both of the proposed strategies. In this example, we assume that PC_{vmt} of each VM type is linearly proportional to c_{vmt}. The resulting budget distribution produced by FFTD and SFTD are shown in Fig. 2a and Fig. 2b respectively and their corresponding schedule in Fig. 3a and 3b. Furthermore, the *Queue of Ready Tasks* column represents the tasks that are ready for execution while the *Deployment of Tasks & VMs* presents the ready tasks deployment to the leased VMs and the *spare budget* shown are the values after the tasks are deployed in each step.

We can see that steps 1 to 3 for FFTD and SFTD are similar except for the VM type provisioned and the spare budget. The VM type chosen is different based on the sub-budget allocated to the entry tasks for which new VMs are provisioned. In Step 4 of FFTD, the algorithm delays task B because its sub-budget is



(a) FFTD (b) SFTD

Fig. 2: Sample of budget distribution scenario.

\$0. After task F finishes, task G becomes ready for execution. The algorithm prioritizes task G to be scheduled because it has lower EFT than task B. Then, task G reuses v_2(*large*), and task B reuses the same VM after task G finishes. Task D becomes ready for execution after task B and C execute and reuses v_1(*large*). Finally, FFTD costs \$6 for the execution; it has \$1 spare budget left and obtains a makespan of 900s.

In Step 4 of SFTD, the algorithm schedules task B and provisions a new small VM based on its sub-budget. Eventually, there is enough extra budget for leasing a faster VM type. Hence, the algorithm leases a new large VM for task B. After task F completes, task G becomes ready for execution and reuses v_2(*small*). Then, task D becomes ready for execution after task B and C finish and reuses v_3(*large*), In the end, SFTD costs \$5 for the execution; it has \$2 spare budget left and produces a 1700s makespan.

V. PERFORMANCE EVALUATION

To evaluate the algorithms' performance, we used synthetic workflows created based on the characteristics of five well-known real workflow applications from different scientific areas. They were generated using the WorkflowGenerator[1] tool. The runtime estimate generated by this tool for each task was used as the size of the task instead.

The Montage (astronomy) workflow is used to produce sky mosaics from a set of input images. Most of its tasks are I/O intensive, and they do not require much CPU processing. The LIGO (astrophysics) workflow is used to detect gravitational waves. It consists mostly of CPU intensive tasks with high memory requirements. The SIPHT (bioinformatics) workflow is used for automatic searching of sRNA encoding-genes. Most of the tasks in SIPHT have high CPU and low I/O utilization. Epigenomics (bioinformatics) is a CPU intensive workflow that is used for executing various genome-sequencing operations. Finally, the CyberShake (earthquake science) workflow generates synthetic seismograms to characterize earthquake hazards and is considered data-intensive with large memory and CPU requirements.

Different budget intervals were employed in the experiments. We assume that the minimum budget to run the workflow is equal to the cost of running all the tasks on a single VM of the cheapest type. Based on this minimum budget, we define

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

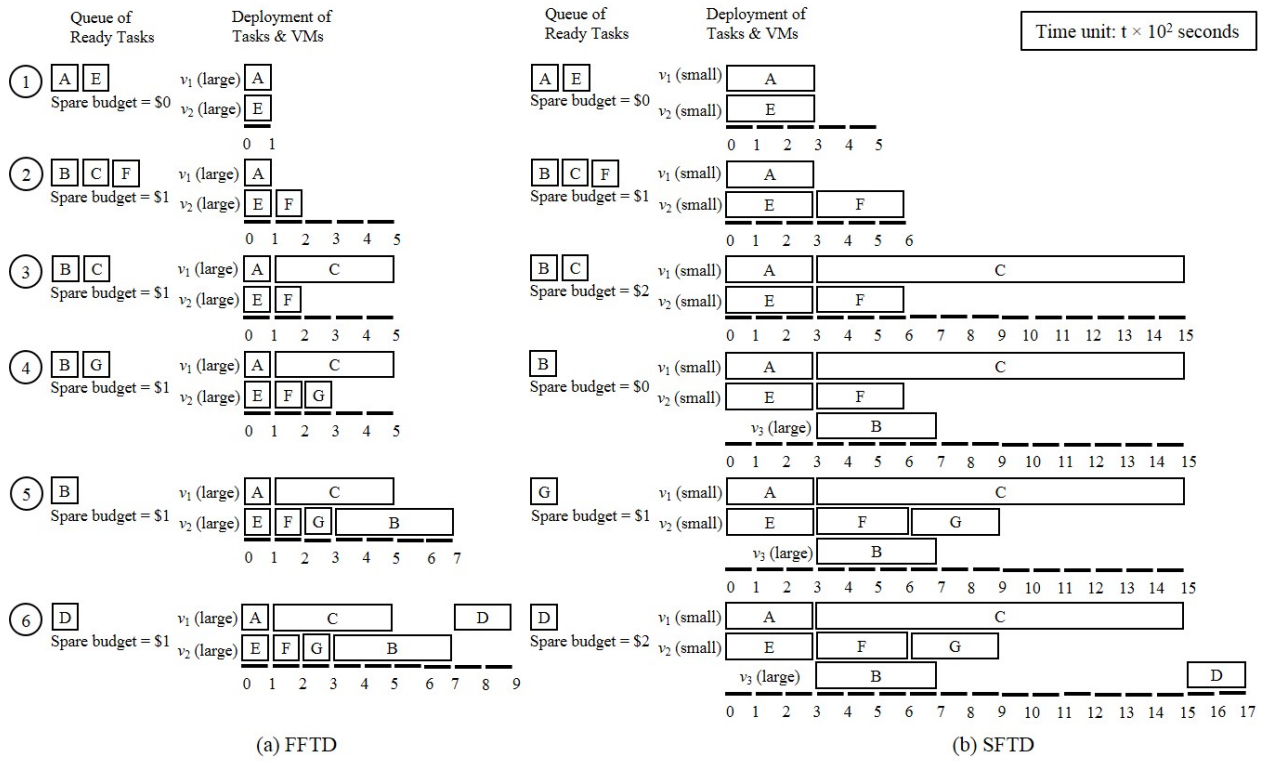


Fig. 3: Sample of scheduling and resource provisioning scenario.

Table. II: VM TYPES AND PRICES USED

Name	Memory (GiB)	vCPU	ECU	Price per Hour ($)
small	3.75	2	7	1
medium	7.5	4	14	2
large	15	8	28	4
xlarge	30	16	56	8

ten different budget intervals as seen in Eq. 11.

$$budget = \alpha * min_{budget} \quad where \quad 0 < \alpha < 11 \tag{11}$$

The tightest budget in the range corresponds to a budget estimated with $\alpha = 1$ while the most relaxed one was calculated using $\alpha = 10$.

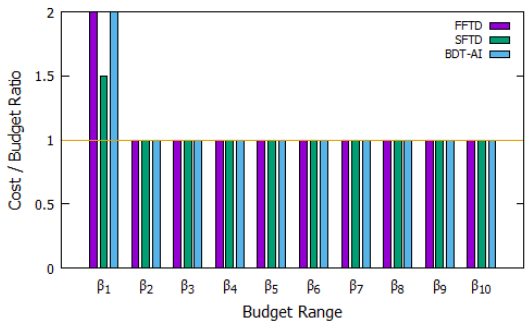
We used CloudSim [19] to model an IaaS provider with a single data center and four types of VMs. The VM type configurations used are shown in Table II. Their CPU capacity and price are a simplified version of the compute optimized (c4) instance types offered by Amazon EC2 that have a linear relationship between processing capacity and price. A VM billing period of one hour was modeled for all VM types, and the VM provisioning delays were set to 97 seconds based on the study by Mao and Humphrey [20]. The CPU performance of VMs was degraded by at most 24% based on a normal distribution with a 12% mean and a 10% standard deviation as reported by Jackson et al. [21].

A. Algorithm Performance

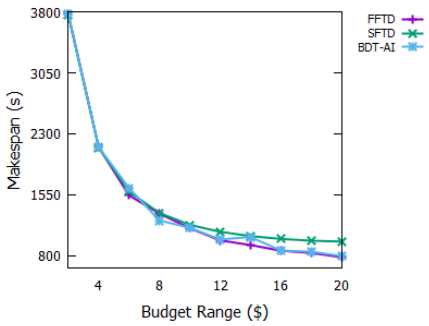
The goal of these experiments is to evaluate the algorithm's performance regarding cost and makespan. The cost performance is measured using the cost to budget ratio to assess the algorithm's ability to meet the budget constraint. Ratio values greater than one indicate a cost larger than the budget, values equal to one mean a cost equal to the budget, and values smaller than one represent a cost smaller than the budget. Furthermore, the experiments for each budget interval were repeated 100 times, and we plot the mean value in the charts.

We compare our algorithm with Budget Distribution with Trickling (BDT) [15], a dynamic level-based budget distribution algorithm that has similar objectives to our solution. BDT schedules the tasks based on their Earliest Start Time (EST) and introduces a Time Cost Trade-off Factor (TCTF), which calculates the trade-off ratio between cost and time for executing a task in a VM type. Then, it selects the VM type with the largest value in the resource provisioning phase. BDT executes all tasks in a level using the available budget and then trickles down the leftover budget to the level below. It delays tasks whenever the budget is not enough to lease new VMs and enforces them to reuse VMs when possible. The authors of BDT introduce several budget distribution strategies, and the 'All-In' scenario presents the best performance. Hence, we use BDT-AI (BDT-All In), to evaluate our proposed algorithm.

1) Budget Constraint Evaluation: To analyze how the algorithms perform in terms of meeting the budget constraint, we plot the cost/budget ratio values for each workflow and budget interval in Fig. 4a, 5a, 6a, 7a and 8a. For the Montage workflow, the results are presented in Fig. 4a. SFTD performs better than the other algorithms in the strictest budget interval; this is probably due to its choice of cheapest VM type when

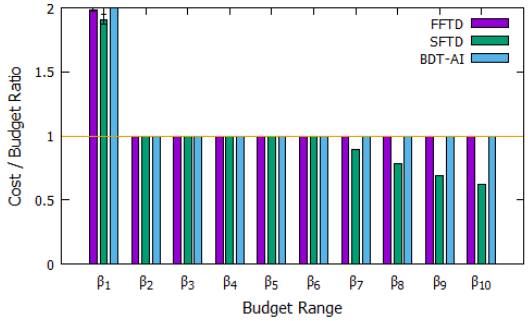


(a) Cost/Budget Ratio

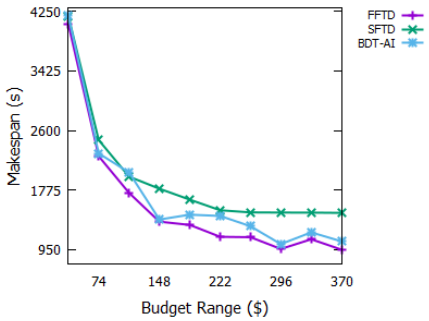


(b) Makespan

Fig. 4: Cost/Budget Ratio and Makespan Performance of Montage Workflow.

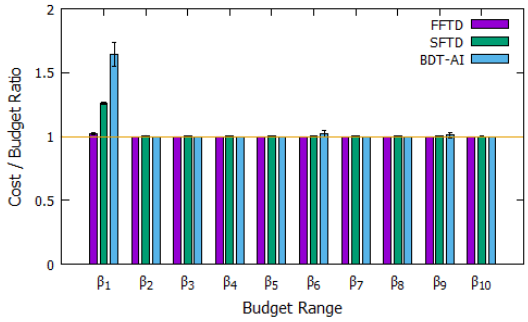


(a) Cost/Budget Ratio

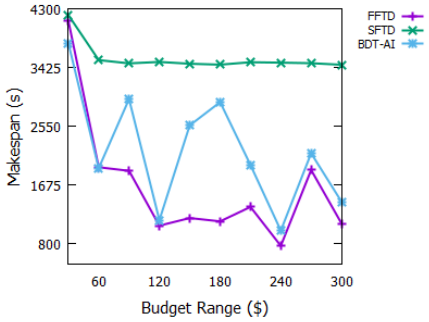


(b) Makespan

Fig. 5: Cost/Budget Ratio and Makespan Performance of LIGO Workflow.



(a) Cost/Budget Ratio



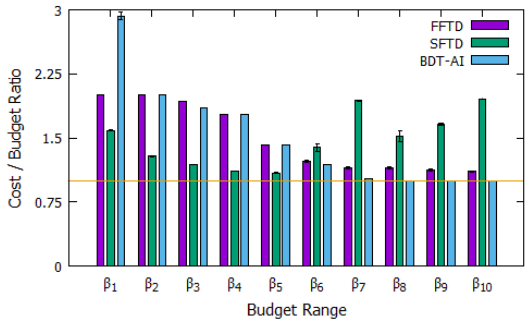
(b) Makespan

Fig. 6: Cost/Budget Ratio and Makespan Performance of SIPHT Workflow.

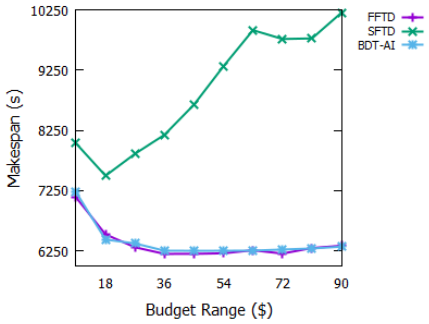
making provisioning decisions. Interestingly, the performance of the algorithms is the same for the remaining intervals, with the ratio values being equal in every case. A possible reason for this is related to the coarse-grained billing period. We can see in Fig. 4b that the makespan obtained by all the algorithms starting from β_2 is much smaller than the length of a billing period. This means that VM performance degradation is not significantly affecting the cost because difference between the makespan and the billing period is wide enough to tolerate this degradation. Although there is no difference in the performance between FFTD and BDT-AI in terms of meeting the budget constraint, we found that FFTD outperforms BDT-AI in terms of the makespan; this is further discussed in Section V-A2.

The results obtained for the LIGO workflow are shown in Fig. 5a. We can see that the first budget interval is too strict. Hence, all algorithms violate the budget constraint. In general, SFTD produces lower cost/budget ratio values even though the other two algorithms are also able to meet the budget constraint for the remaining cases.

Fig. 6a depicts the results for the SIPHT workflow. The

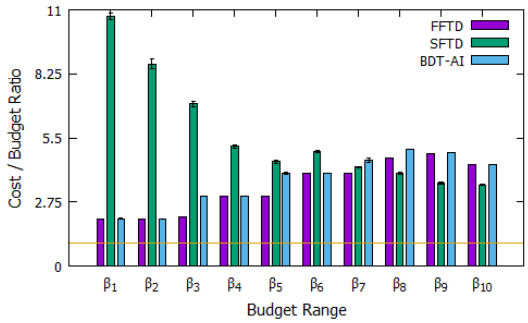


(a) Cost/Budget Ratio

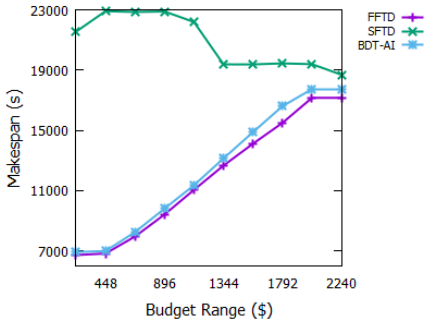


(b) Makespan

Fig. 7: Cost/Budget Ratio and Makespan Performance of CyberShake Workflow.



(a) Cost/Budget Ratio



(b) Makespan

Fig. 8: Cost/Budget Ratio and Makespan Performance of Epigenomics Workflow.

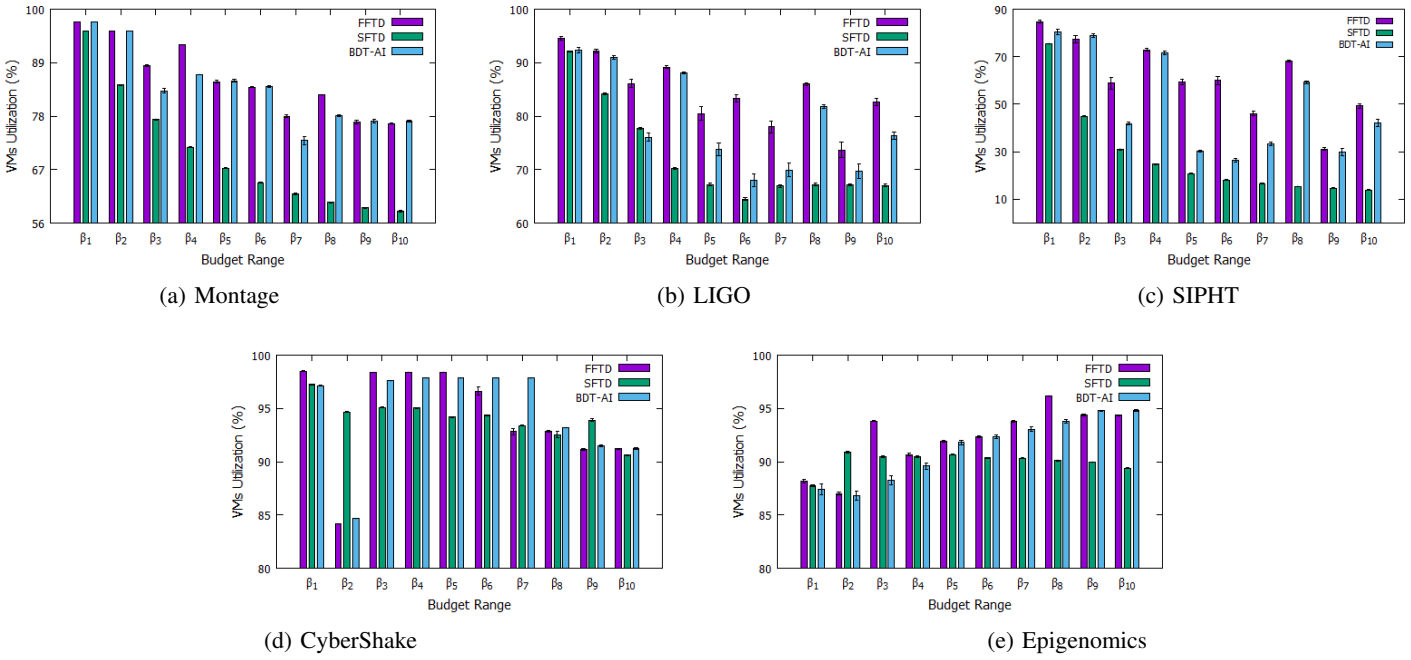
first budget interval is violated by all algorithms with FFTD exceeding the budget by the smallest value. We can observe that the performance variation affects the algorithms' ability to meet the budget constraint from the marginal difference between the cost and the budget in the graphs. In general, FFTD outperforms the other two algorithms in meeting the budget constraints for the SIPHT workflow.

Fig. 7a shows the results obtained for the CyberShake workflow. In general, all algorithms fail to meet the budget constraint in every case, except for BDT-AI which succeeds in achieving its goal in the last three intervals. A possible explanation for these results is the CyberShake workflow characteristics as a data-intensive workflow that involves extensive I/O activities. This I/O overhead is evidenced in the SFTD results; the cost/budget ratio values increase as the number of VMs increases. The larger the number of VMs, the larger the amount of files that have to be transferred over the network and, although all algorithms consider data transfer times when estimating a task's processing time, network traffic congestion causes unpredicted costs. As a result, with SFTD being the algorithm that provisions the larger number of VMs, its schedules are highly impacted by network congestion. It is worthwhile mentioning that the ratio values for FFTD and BDT-AI decrease as the budget increases. While SFTD allocates more VMs as the budget increases, FFTD and BDT-AI use the additional budget to provision faster VMs instead.

The Epigenomics workflow results are shown in Fig. 8a. Contrary to the Cybershake case, in the Epigenomics case SFTD leases faster VMs rather than increasing the number of lesased VMs. It produces very high cost/budget ratio values for the earlier budget intervals; however, the ratios consistently decrease as the budget increases. A possible reason is that the number of VMs provisioned by SFTD remains relatively constant throughout the budget intervals. However, FFTD outperforms the other two algorithms regarding meeting the budget constraint for most of the cases.

Overall, FFTD demonstrates equal or better performance than BDT-AI in 88% of the cases regarding cost/budget ratio values. The only case where BDT-AI outperforms FFTD is for the CyberShake workflow in which 90% of the cases BDT-AI obtains equal or better performance. However, it needs to be mentioned that the difference in performance is marginal in some cases such as the SIPHT workflow.

2) Makespan Evaluation: The first half of budget intervals in the Montage workflow (Fig. 4b) show a marginal difference in makespan. This is due most likely to the characteristics of the tasks in the Montage workflow that highly depend on I/O rather than CPU processing (I/O-bound workflows). Hence, choosing faster VM types does not significantly affect the makespan. However, as the budget increases, the second half of the budget intervals lead to a larger makespan difference. A possible reason is due to the large number of VMs provisioned by SFTD



(a) Montage (b) LIGO (c) SIPHT

(d) CyberShake (e) Epigenomics

Fig. 9: VMs Utilization for Different Workflow Applications.

at runtime that contribute to the increase of the communication overhead. Finally, FFTD obtains lower makespan than BDT-AI in 70% of the cases in which both algorithms achieve the same performance regarding cost/budget ratio.

Fig. 5b depicts the results obtained for the LIGO workflow. Although the graph's trend is similar to the Montage results, the difference between all algorithms is more clearly observed in this case. FFTD shows the lowest makespan for all cases. The results obtained for the SIPHT workflow are depicted in Fig. 6b. Similar to the previous results, FFTD obtains the lowest makespan and displays more stable results than BDT-AI.

The results achieved for the CyberShake workflow are shown in Fig. 7b. FFTD produces slightly lower makespans in 60% of the cases when compared to BDT-AI while SFTD performs the worst. CyberShake is considered as a data-intensive workflow that involves a high number of data transfer activities. This explains SFTD's performance as it provisions a larger number of VMs as the budget increases.

The Epigenomics workflow results are shown in Fig. 8b. The makespan trend is different from the other workflow scenarios in which both FFTD and BDT-AI have a larger makespan as the budget increases. A possible explanation is the fact that Epigenomics consists of tasks that are both CPU and I/O intensive. Having a small number of VMs provisioned is probably the best approach for executing this workflow. It needs to be noted that this behavior should be a guide for the users when defining the budget for Epigenomics workflow. Finally, FFTD shows the lowest makespan in all scenarios.

Overall scenario, FFTD demonstrates lower makespan in 84% of all cases. In the case where FFTD gets an equal performance to BDT-AI in terms of meeting the budget constraint, it obtains a lower makespan in 80% of the cases. Meanwhile, in the cases in which FFTD achieves lower cost/budget ratios, which is usually accompanied by higher makespans, it also successfully obtains lower makespans than BDT-AI in 93% of the cases. Nevertheless, in some cases, the difference in makespan is marginal.

B. VM Utilization

To better understand the behavior of the algorithms, we analyzed the average VM utilization for each workflow. High VM utilization means the algorithm is capable of mapping tasks to VMs efficiently by utilizing idle time slots. Hence, this performance metric is suitable to evaluate the algorithms' ability to deal with coarse-grained IaaS cloud billing periods. The VMs utilization results are shown in Fig. 9.

For the Montage workflow, FFTD presents higher VM utilization in 50% of the cases than BDT-AI. Meanwhile, in the LIGO workflow scenario, FFTD achieves the highest VM utilization in all cases. On average, the SIPHT workflow shows the lowest VM utilization for all of the experiments. However, FFTD obtains a higher VM utilization in 90% of the cases when compared to BDT-AI for SIPHT. Furthermore, BDT-AI presents a higher VM utilization in 60% of the cases than FFTD for the CyberShake workflow; this supports the cost/budget ratio and makespan results. Finally, in the Epigenomics workflow, FFTD produces higher VM utilization than BDT-AI in 80% of the cases.

Overall, in 72% of the cases, FFTD demonstrates better performance than BDT-AI regarding VM utilization. However, for the CyberShake workflow, BDT-AI performs better than FFTD. A possible explanation is because the level-based strategy of BDT-AI works best for data-intensive workflows that have a high degree of parallelism such as CyberShake.

VI. Conclusions and Future Work

In this paper, we presented task-based budget distribution strategies for executing scientific workflows in IaaS clouds with coarse-grained billing periods. The problem was modeled as a workflow resource provisioning and scheduling problem which aims to minimize the makespan while meeting the user-defined budget. Furthermore, the proposed strategy exploits the independent task readiness for executing the workflow.

The algorithm distributes the workflow budget to each task and drives the resource usage through the sub-budget of each task. It schedules tasks individually based on their earliest finish time whenever their parent tasks have completed execution and their input data are available. The algorithm implements a VM reusing policy to utilize the idle time slots that occur due to the coarse-grained billing periods. It provisions the fastest VM type possible within the budget whenever it is necessary due to the unavailability of reusable idle VMs. Every time a task finishes, the algorithm considers the budget spent so far and adjusts the next task scheduling decision if necessary. For each task that reuses idle VMs, its unused sub-budget is kept as spare budget and utilized to update the budget distribution or to lease faster VMs in the resource provisioning phase.

The performance evaluation results demonstrate that our solution has an overall better performance than the state-of-the-art algorithm. It successfully obtains 88% equal or better performance regarding cost/budget ratio values and achieves lower makespans in 84% of the cases. It obtains higher VM utilizations in 72% of the experiments. As future work, we will investigate this approach on a workflow-as-a-Service (WaaS) platform with dynamic workloads of multiple workflows.

Acknowledgments

This research is partially supported by LPDP (Indonesia Endowment Fund for Education) and ARC (Australia Research Council) research grant.

References

[1] M. A. Rodriguez and R. Buyya, "A Taxonomy and Survey on Scheduling Algorithms for Scientific Workflows in IaaS Cloud Computing Environments," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, p. e4041, 2017.

[2] ——, "Scheduling Dynamic Workloads in Multi-tenant Scientific Workflow as a Service Platforms," *Future Generation Computer Systems*, 2017. [Online]. Available: https://doi.org/10.1016/j.future.2017.05.009

[3] M. Mao and M. Humphrey, "Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," in *Proceedings of The IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011, pp. 1–12.

[4] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[5] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling Multilevel Deadline-constrained Scientific Workflows on Clouds based on Cost Optimization," *Scientific Programming*, vol. 2015, p. 5, 2015.

[6] V. Arabnejad, K. Bubendorfer, and B. Ng, "Deadline Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds," in *Proceedings of The ACM/IEEE International Conference on Utility and Cloud Computing*. ACM, IEEE, 2016, pp. 70–78.

[7] Z. Cai, X. Li, and R. Ruiz, "Resource Provisioning for Task-batch based Workflows with Deadlines in Public Clouds," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, p. 1, 2017.

[8] H. Chen, J. Zhu, Z. Zhang, M. Ma, and X. Shen, "Real-time Workflows Oriented Online Scheduling in Uncertain Cloud Environment," *The Journal of Supercomputing*, 2017. [Online]. Available: https://doi.org/10.1007/s11227-017-2060-4

[9] F. Wu, Q. Wu, Y. Tan, R. Li, and W. Wang, "PCP-B2: Partial Critical Path Budget Balanced Scheduling Algorithms for Scientific Workflow Applications," *Future Generation Computer Systems*, vol. 60, pp. 22–34, 2016.

[10] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end Delay Minimization for Scientific Workflows in Clouds under Budget Constraint," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 169–181, 2015.

[11] X. Wang, B. Cao, C. Hou, L. Xiong, and J. Fan, "Scheduling Budget Constrained Cloud Workflows with Particle Swarm Optimization," in *Proceedings of The IEEE Conference on Collaboration and Internet Computing*. IEEE, 2015, pp. 219–226.

[12] A. Verma and S. Kaushal, "Budget Constrained Priority based Genetic Algorithm for Workflow Scheduling in Cloud," in *Proceedings of The International Conference on Advances in Recent Technologies in Communication and Computing*. IET, 2013, pp. 216–222.

[13] ——, "Deadline and Budget Distribution based Cost-Time Optimization Workflow Scheduling Algorithm for Cloud," in *Proceedings of The International Conference on Recent Advances and Future Trends in Information Technology*. IJCA, 2012.

[14] M. A. Rodriguez and R. Buyya, "Budget-driven Resource Provisioning and Scheduling of Scientific Workflow in IaaS Clouds with Fine-grained Billing Periods," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 9, no. 4, 2015.

[15] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds," in *Proceedings of The IEEE International Conference on e-Science*. IEEE, 2016, pp. 137–146.

[16] P. Leitner and J. Cito, "Patterns in The Chaos: A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Transaction of Internet Technology*, vol. 16, no. 3, pp. 1–23, 2016.

[17] J. Yu, R. Buyya, and C. K. Tham, "Cost-based Scheduling of Scientific Workflow Applications on Utility Grids," in *Proceedings of The IEEE International Conference on e-Science and Grid Computing*. IEEE, 2005, pp. 8–pp.

[18] Y. Yuan, X. Li, Q. Wang, and Y. Zhang, "Bottom Level based Heuristic for Workflow Scheduling in Grids," *Chinese Journal of Computers -Chinese Edition-*, vol. 31, no. 2, p. 282, 2008.

[19] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[20] M. Mao and M. Humphrey, "A Performance Study on The VM Startup Time in The Cloud," in *Proceedings of The IEEE International Conference on Cloud Computing*. IEEE, 2012, pp. 423–430.

[21] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance Analysis of High Performance Computing Applications on The Amazon Web Services Cloud," in *Proceedings of The IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 159–168.