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

Muhammad Hafizhuddin Hilman, Maria Alejandra 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 study of large scale scientific workflows deployment into cloud computing is prevalent. Infrastructure as a Service (IaaS) clouds, offer access to the resources required for execution of scientific workflows in which can be provisioned on-demand to match the number of workflow tasks that need to be executed. This flexibility leads to a trade-off between two conflicting Quality of Service (QoS) requirements: time and cost. Most studies in this field have only focused on meeting the deadline constraint while minimizing the cost and few of them exploited the budget spending to minimize the makespan. Nevertheless, several strategies have been introduced for budget constraint scheduling problem including dynamic scheduling of tasks as they ready for the execution in which enables the algorithm to make decisions based on the current state of the system. Since the strategy is task-based, the overall workflow budget must be distributed to each individual task. In this paper, we propose Fastest-First Task-based Distribution (FFTD) and Slowest-First Task-based Distribution (SFTD) budget distribution algorithms designed to minimize the makespan of workflows with budget constraints in IaaS public clouds. Our performance evaluation results show that in 88% of the cases, FFTD shows equal or better performance in terms of cost / budget ratio values; obtains lower makespan in 84% of all scenario and presents higher VM utilization in 72% of the staged experiments than state-of-the-art algorithm.**

***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 ever more meaningful and precise experiments and simulations. These scientific experiments are generally expressed as workflows (i.e., applications that are composed of multiple computational tasks linked together). In scientific workflows, tasks represent different computations and links between tasks represent data dependencies. They are large-scale applications and require large computational resources to process their input data in a reasonable time. Thus, distributed environments are commonly used to process them, and cloud computing has become a popular platform for their deployment in recent years.

Infrastructure as a Service (IaaS) Clouds offer access to the resources required for the execution of scientific workflows. Specifically, they provide access to computational power by leasing Virtual Machines (VMs) of varying configurations (i.e., VM types). These VMs can be rented and released on-demand and are charged based on their usage in increments of a billing period defined by the provider. For example, Amazon EC2 charges VMs by the hour and a machine used for one and a half hours is billed as if it was used for two full hours. This flexibility and 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 time.

However, the flexibility and ability to easily scale number of resources lead to a trade-off between two conflicting Quality of Service (QoS) requirements: time and cost. The reason is very simple, 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. This has led to extensive research [1] in this topic, with most works proposing algorithms that aim for minimizing the total execution cost while finishing the workflow execution before a user-defined deadline. In this work, we focus on optimizing the resources usage so that the total execution time of the workflow (i.e., makespan) is minimized while meeting a budget constraint.

There are various strategies that 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 of 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. To improve the scalability issue, other algorithms use lighter-weight heuristics to produce static schedules, 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 enables the algorithm to easily scale 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 individual 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. This is a challenging problem mainly due to the pricing model offered by IaaS clouds providers.

It is common for the average execution time of workflow tasks to be considerably smaller than the coarse-grained billing periods (e.g., one hour) offered by IaaS vendors. Thus, scheduling algorithms must aim to efficiently utilize idle time slots on leased

VMs as a cost-controlling mechanism. This creates a challenge when deciding the portion of the budget that should be allocated to tasks as their cost must be estimated by either rounding up their execution time to one (or more) billing periods or by estimating their cost in time units. Deciding how to factor VM provisioning delays when estimating the costs of tasks is another important challenge. To avoid this, some algorithms choose to consolidate tasks per workflow level and assign a collective budget to them to be spent greedily. Depending on how the budget is split, this may result in insufficient budget assigned 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 individual 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. In addition, 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 heterogenous computing resources, VM provisioning delays, and the dynamic and uncertain behaviour of VMs performance. We propose the Fastest-First Task-based Distribution (FFTD) and Slowest-First Task-based Distribution (SFTD) algorithms designed to minimize the makespan of workflows with budget constraints in IaaS public clouds. The algorithms are dynamic and schedule tasks whenever they are ready for execution. Our simulation results demonstrate that FFTD can adapt to unexpected delays and meeting the budget constraint while achieving lower makespans when 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 system model, including the application model and the resource model. The proposed algorithms are explained in section IV. A performance evaluation and results is presented in section V. Finally, the conclusions and future work are depicted in Section VI.

II. Related Work

Scientific workflows deployment in IaaS clouds environment has been extensively researched. Most algorithms focus on two QoS parameters: time and cost. The majority of these algorithms have objectives of meeting the deadline constraint while minimizing the resources leasing cost. Examples include the solutions presented by Mao and Humphrey [2], Abrishami et al. [3], Malawski et al. [4], Arabnejad et al. [5], Cai et al. [6] and Chen et al. [7].

Only a few of the existing algorithms focus on meeting the budget constraint while minimizing the workflow makespan. An example is the Partial Critical Paths Budget Balanced (PCP-B^2) algorithm [8]. 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 algorithm [9] uses a similar strategy by iteratively refining and initiating schedule plan to encourage the use of more powerful VM types if there is budget remaining. Other works with the same objectives use Particle Swarm Optimization (PSO) [10] and Genetic Algorithms [11] to develop a static budget plan that minimizes a workflows makespan before runtime. These algorithms rely on calculating a near-optimal schedule by using computationally intensive meta-heuristic techniques. This 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.

BAGS [12] 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) algorithm [13] uses similar strategy of consolidating tasks on the same workflow level. The budget is distributed to each level and the algorithm trickles down any leftover 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 the tasks are scheduled based on task readiness independently. Whenever all task's parents are finished and the data input is available, it will be scheduled regardless of their level.

III. Application and Resource Model

Our work is designed to schedule scientific workflows that is modelled as Directed Acyclic Graphs (DAGs), graph that consists of directed edges without any cycles. A workflow W consists of a set of tasks $T = (t_1, t_2, ..., t_n)$ and a set of directed edges $E = (e_{12}, e_{13}, ..., e_{mn})$ in which an edge e_{ij} represents data dependency between task's parent t_i and task's child t_j, hence, it implies that t_j will only being executed after t_i is completed. In addition, a size of task S_t is measured in Millions of Instructions (MI).

Virtual Machines (VMs) are leased using on-demand pricing model that is charged the usage per billing period *bp* in which any partial usage will be rounded up to the nearest billing period. Furthermore, our work considers heterogeneous environment, where exists several VM type *vmt* that have different processing capacity PC_{vmt} and different cost per billing period c_{vmt}. The processing capacity PC_{vmt} is measured in Million Instruction per Second (MIPS) and it corresponds with task size S_t. We assume that VM performance varies over time and PC_{vmt} values published by IaaS cloud providers is the maximum capacity. In addition, the runtime of a task is denoted as RT^t_{vmt}, it is calculated using simple approach based on task size S_t and processing capacity PC_{vmt} as shown in Eq. 1. Note that this value is simply an estimate and does not rely on it being one hundred percent accurate.

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

We consider global shared storage system for data sharing between tasks where each task is getting input data D^t_{in} from global-shared repository and sent back the output data D^t_{out} produced. Furthermore, each VM has bandwidth B_{vmt} while the rates of global storage read and write is GS_{read} and GS_{write} respectively. The bandwidths and I/O rates are changing over time based on number of transactions *Tr* running at the time *t* as seen in Eq. 2, 3 and 4.

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

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

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

The time for taking input data from global shared storage into VM is shown in Eq. 5.

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

on the counterpart, time for storing output data is similar, as seen in Eq. 6.

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

We assume the global storage and VMs are located on the same region or availability zone, hence, data transfer between these is free of charge, as in the case for most IaaS providers such as Amazon EC2, Google Cloud Storage and Rackspace Block Storage. Nevertheless, the output data is always kept on VM local temporary storage. It needs to be mentioned here that there is no need to read the input data if it is available in VM local temporary storage where the task will be executed. Hence, this scenario minimizes the time for getting the input data. The total processing time of a task PT^t_{vmt} is shown in Eq. 7.

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

The cost C^t_{vmt} of a task for using a VM with type *vmt* and price c_{vmt} for a billing period *bp* is including the provisioning delay T_{pdelay} and deprovisioning delay T_{ddelay}, as seen in Eq. 8.

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

We use the cost estimation in the budget distribution as well as in the resource provisioning phase when choosing the VM types.

IV. Resource Provisioning and Scheduling Algorithm

Our work is based on the EPSM algorithm [14], a simple dynamic heuristic-based algorithm for Workflow-as-a-Service (WaaS) framework that can handle dynamic workload of multiple workflows. The algorithm's objective is meeting the deadline constraint while minimizing the cost. Furthermore, EPSM uses the container technology to reuse VMs and implements sharing resources policy between workflows.

We modify the EPSM to a budget-constrained algorithm without the implementation of container to fit the single workflow environment. Our algorithm consists of two phases:

1) Budget Distribution: The workflow's budget is distributed into each individual task using two approach, 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 per the task's sub-budget whenever the idle VMs to reuse is not available.

A. Budget Distribution

The budget drives whole scheduling process. Hence, choosing the fastest resources that are affordable within the budget, decrease the probability of exceeding the VM billing period caused by performance variation that affects the execution time. The total cost increases if a task's execution time is missed from the runtime estimation and exceeds a billing period, even for just a small fraction. Therefore, budget distribution is challenging due to the existence of coarse-grained IaaS cloud billing periods. In some cases, distributing the budget based on the task's runtime estimation without considering a billing period leads to budget insufficiency, condition where the task's sub-budget is not enough to provision a new VM.

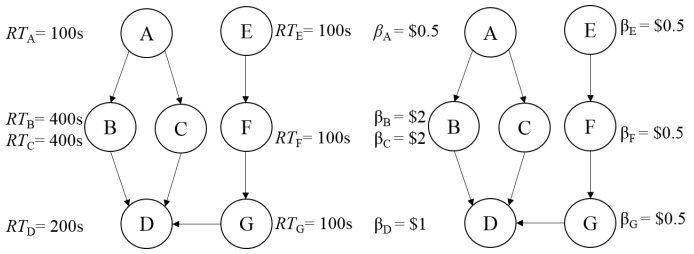


Fig. 1: Sample of budget insufficiency scenario

Consider an illustration in Fig. 1, suppose there are two VM types in IaaS cloud provider, small type which costs $1/hour and large type that costs $3/hour. We assume that more powerful VMs capable of processing a task faster are more expensive than less capacity ones. Then, we have a workflow consists of seven tasks with estimated runtime for each task 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 proportional to the tasks total runtime while ignoring the VM billing period, the budget for task A, E, F and G will be insufficient. 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.

We propose budget distribution based on the task's estimated execution order. The tasks are executed starting from the entry tasks which are in the first level of a workflow then proceed to their children that are residing in the next level. Therefore, we use Deadline Top Level (DTL) [15] technique that makes the entry task of workflow as starting point, as seen in Eq. 9, instead of Deadline Bottom level (DBL) [16] technique that start the level allocation from the exit task.

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

If we consider the example from Fig. 1, DTL allocates task A and task E to the same level(1) while DBL allocates the tasks to a level starting from task D as level(1). Consequently, using DBL, task A is allocated to the level(3) which has a different level with task E that is in the level(4). Although a workflow is obviously being processed starting from the entry tasks, allocating them into a different level will cause delay for algorithms that execute the tasks based on their level. In addition, to determine the order of

Algorithm 1 Budget Distribution

procedure DISTRIBUTEBUDGET(β, T)
  S = task's estimated execution order
  for each task $t \in T$ **do**
    allocateLevel(*t*, *l*)
    initiateBudget(*0*, *t*)
  for each level l **do**
    T_l = set of all tasks in level l
    sort T_l based on ascending Earliest Finish Time (EFT)
    put(T_l, S)
  while $\beta > 0$ **do**
    t = *S.poll*
    vmt = chosen VM type
    allocateBudget(C^t_{vmt}, *t*)
    $\beta = \beta - C^t_{vmt}$

tasks in a level, we sort them based on the ascending order of their Earliest Finish Time (EFT) as shown in Eq. 10.

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

The workflow in Fig. 1 is sorted into sequence of A [level(1)] $\rightarrow$ E [level(1)] $\rightarrow$ F [level(2)] $\rightarrow$ B [level(2)] $\rightarrow$ C [level(2)] $\rightarrow$ G [level(3)] $\rightarrow$ D [level(4)], then, the budget distribution algorithm works sequentially distributing the budget to each individual task while considering task's estimated runtime and cost for each VM type. Furthermore, the budget allocation considers the VMs that are charged per billing period, so, the sub-budget allocated for a task is equivalent to the cost of a full billing period. However, there will be several tasks that are not getting the sub-budgets allocation. For these tasks, the algorithm will delay the tasks so they can reuse the idle VMs. The budget distribution algorithm is shown in Algorithm 1.

The algorithm uses two approaches on allocating the budget based on the VM type chosen, Fastest-First Task-based Budget Distribution (FFTD) and Slowest-First Task-based Budget Distribution (SFTD). The FFTD approach chooses the fastest VM type that is affordable within the workflow's budget. Since the algorithm allocates fastest resources to the earliest tasks, their successor's chance of reusing faster VMs than what they can afford increases. Hence, it also improves the possibility of obtaining lower task's processing time even though involves some waiting delay for the idle VMs availability. On the contrary, the SFTD approach chooses the cheapest resources for the tasks. Whenever exists budget left after all tasks are allocated sub-budgets, the algorithm lease a faster VM type that is affordable by the leftover budget. The SFTD ensures most of the tasks are getting the budget allocation so they do not need to wait for the idle VMs. In both approaches, allocating resources to the earliest 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 priority queue based on the EFT ascending order. Then, it provisions a new fastest VM type that is affordable within the task's sub-budget. Whenever exist idle VMs, the algorithm finds

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^{input}_{idle} = set of *vm* $\in VM_{idle}$ that have *t*'s input data
      vm = *vm* $\in VM^{input}_{idle}$ 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* $< C^t_{vmt}$ **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*)

list of VMs that contain the cached input data for a task and assigns VM that has spare time of a billing period that can finish the task with minimum risk of incurring a new billing period. Minimum risk means, if there are two VMs that are unable to finish the task without incurring a new billing period, the algorithm chooses the lowest cost one. If the VM with cached input data is not available, the algorithm assigns any idle VM with the same criteria. The resource provisioning and scheduling algorithm is shown in Algorithm 2.

After a task is completed, the algorithm updates the budget distribution to adjust with the budget spent so far. Meanwhile, it keeps unused sub-budgets of the tasks that reuse idle VMs as spare budget. This spare budget is used later in the budget update or in the provisioning phase to lease faster VM type. The budget update algorithm can be seen in Algorithm 3.

C. Illustrative Example

This section explains how the workflow in Fig. 1 would be scheduled using both 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 is the values after tasks deployment in each

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)

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 are different based on the sub-budget allocated to the entry tasks that are provisioned the new VMs. In step 4 of FFTD, the algorithm delays task B because its sub-budget is \$0. After task F is finished, 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 is finished. Task D becomes ready for execution after task B and C are finished and reuses v_1(*large*). Finally, FFTD costs \$6 for the execution; it has \$1 spare budget left and obtains 900s makespan.

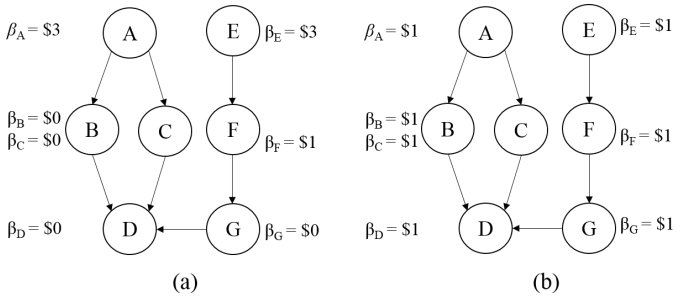


Fig. 2: Sample of budget distribution scenario using (a) FFTD and (b) SFTD

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

V. PERFORMANCE EVALUATION

To evaluate the algorithms' performance, we use five well-known synthetic workflows from different scientific areas. The Montage (Astronomy) workflow is used to generate sky mosaics from a set of input images [17]. 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 [18]. It consists mostly of CPU intensive tasks with high memory requirements. The SIPHT (sRNA Identification Protocol using High-throughput Technology) workflow is used for automatic searching of sRNA encoding-genes in the bioinformatics field [19]. Most of the tasks in the SIPHT have high CPU and low I/O utilization. Another bioinformatics application, the Epigenomics, 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 [20] and is characterized as data intensive with large memory and CPU requirements. These workflows have different structures in terms of task dependencies and task runtimes. Their full description and characterization including the Workflow Generator used for generating the synthetic workflows is presented by Juve et al. [21].

Different budget intervals were used 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 ten different budget intervals as seen in Eq. 11.

$$budget = \alpha * min_{budget} \quad where \quad 0 < \alpha < 11 \qquad (11)$$

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

We use CloudSim [22] to model an IaaS cloud provider with single data center and four types of VMs. The VM type configurations used are shown in Table I. 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 modelled for all VM types, and the VM provisioning delays was set to 97 seconds based on the study by Mao and Humphrey [23]. 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. [24].

Table. I: 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

A. Algorithm Performance

The goal of these experiments is to evaluate the algorithm's performance in terms of cost and makespan. The cost performance is measured using the cost / budget ratio to evaluate the algorithm's ability on meeting the budget constraint. In this way, 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

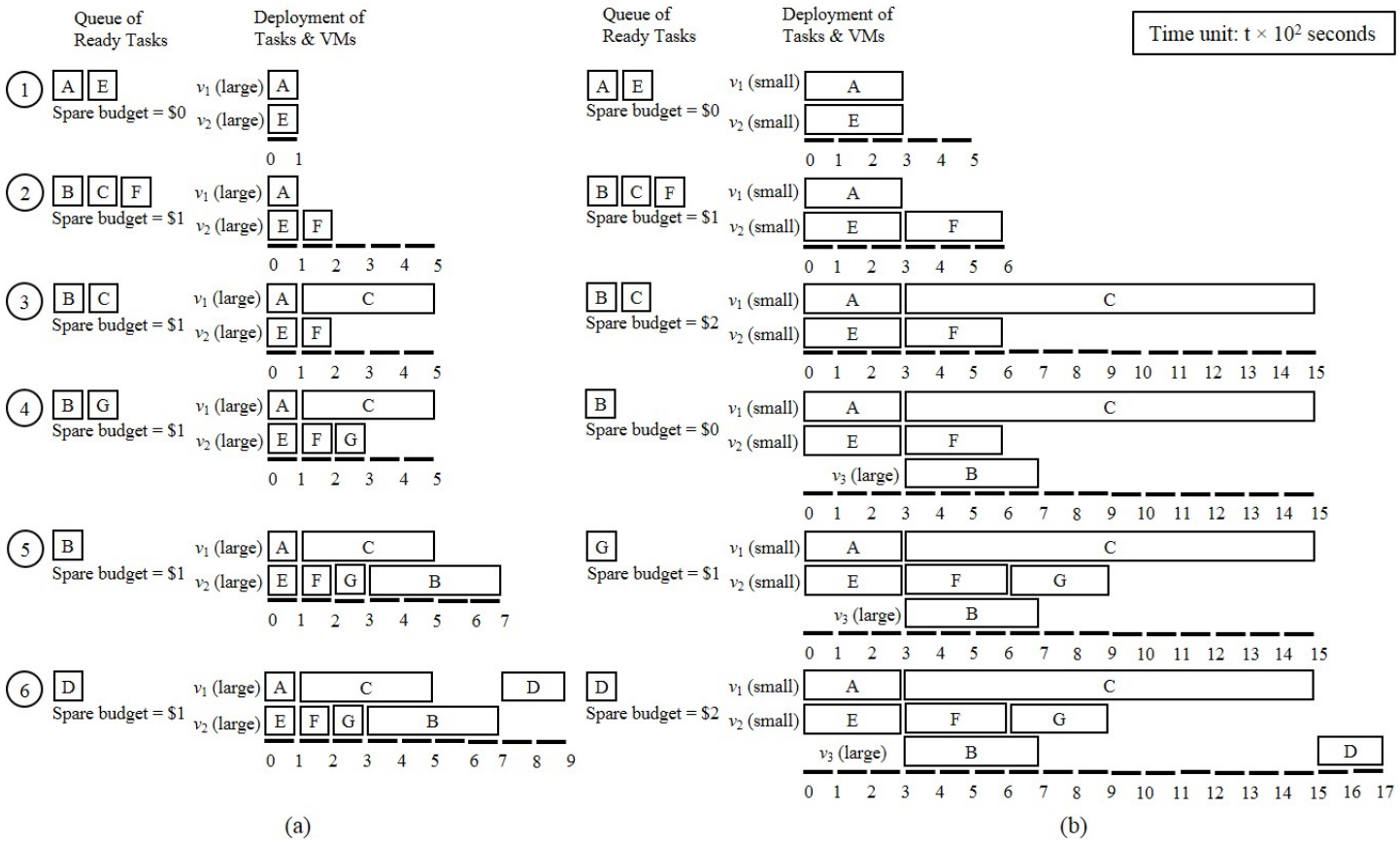


Fig. 3: Sample of scheduling and resource provisioning scenario using (a) FFTD and (b) SFTD.

budget. Furthermore, the experiments for each budget interval are repeated 100 times and we plot the mean value in the charts.

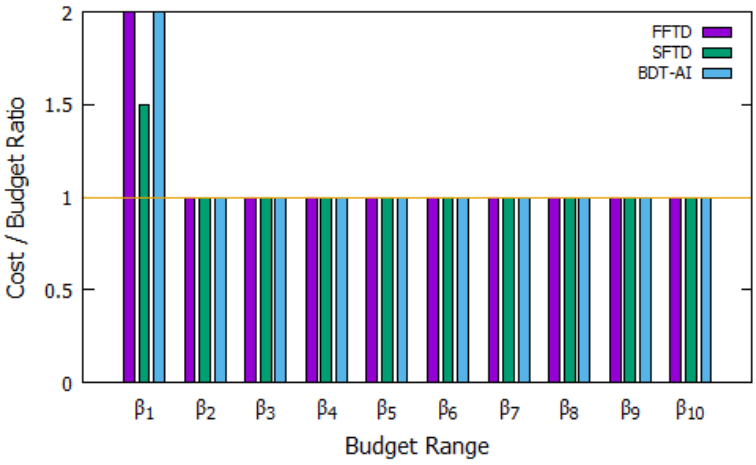
We compare our algorithm with Budget Distribution with Trickling (BDT) [13], a dynamic level-based budget distribution algorithm that has similar objectives to our algorithm. BDT schedules the tasks based on their Earliest Start Time (EST) and introduces Time Cost Trade-off Factor (TCTF) which calculates 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 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' strategy presents the best performance. Hence, we use BDT-AI (All-In), to evaluate our proposed algorithm.

1) Budget Constraint Evaluation: To analyze the algorithms 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 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. The possible reason is related to the coarse-grained billing period. We can see that from Fig. 4b, the makespan obtained by all algorithms starting from the β_2 is quite far from the billing periods. This means, the VM performance variation is not significantly affecting the cost since the difference between the makespan and the billing period is wide enough to tolerate the VM performance 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 makespan; this is further discussed in the 'Makespan Evaluation' section.

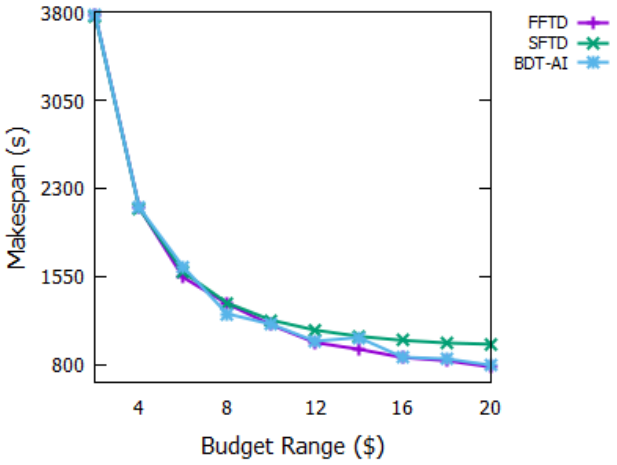
The results obtained for the LIGO workflow are shown in Fig. 5a. We can see that the first budget interval is too strict, hence, all algorithms violate the budget constraint. In general, SFTD produces lower cost / budget ratio values even though the other two algorithms are also able to meet the budget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

Fig. 6a depicts the results for the SIPHT workflow. The first budget interval is violated by all algorithms while FFTD shows the closest margin. We can observe that the performance variation affects the algorithms' ability of meeting 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 this results is the CyberShake workflow characteristics as a data

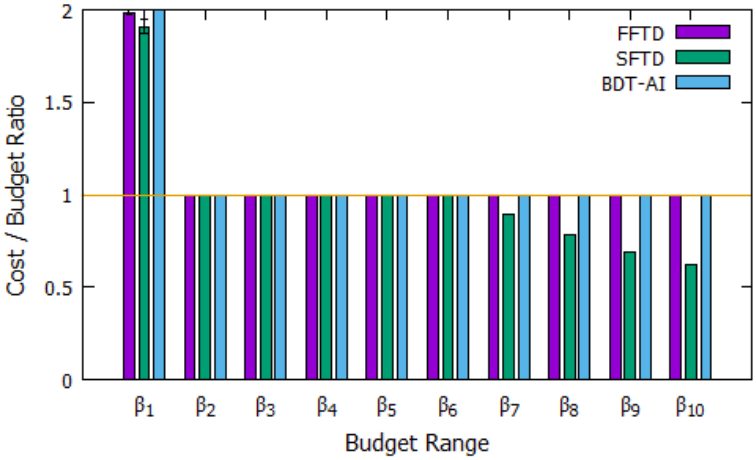


(a) Cost / Budget Ratio

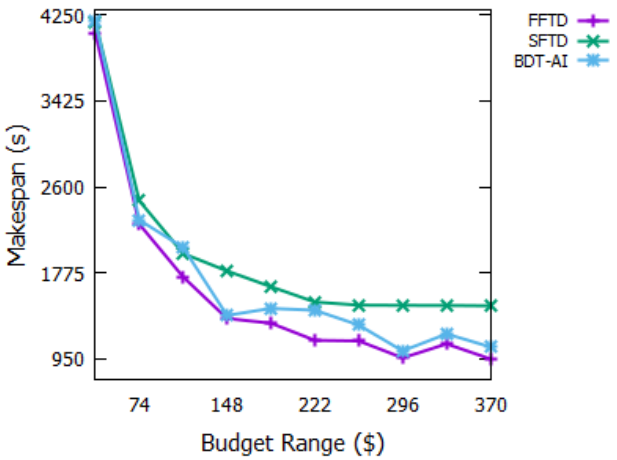


(b) Makespan

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

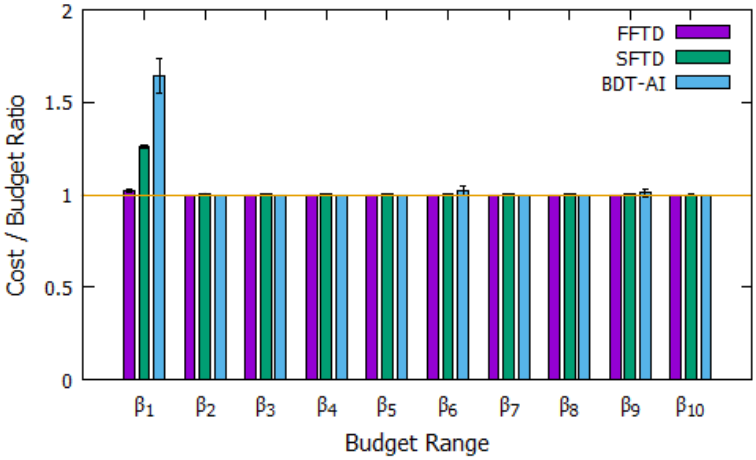


(a) Cost / Budget Ratio

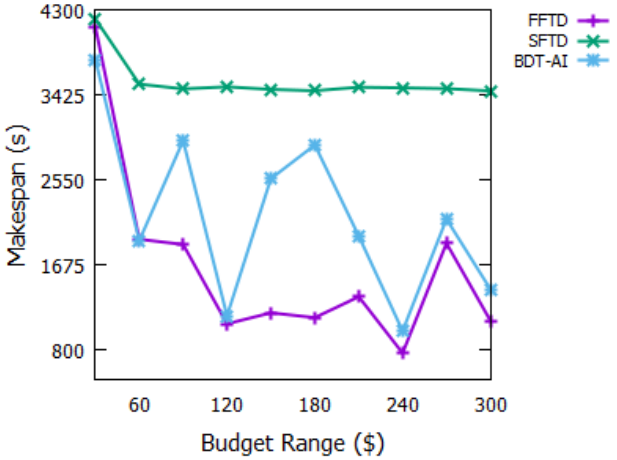


(b) Makespan

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

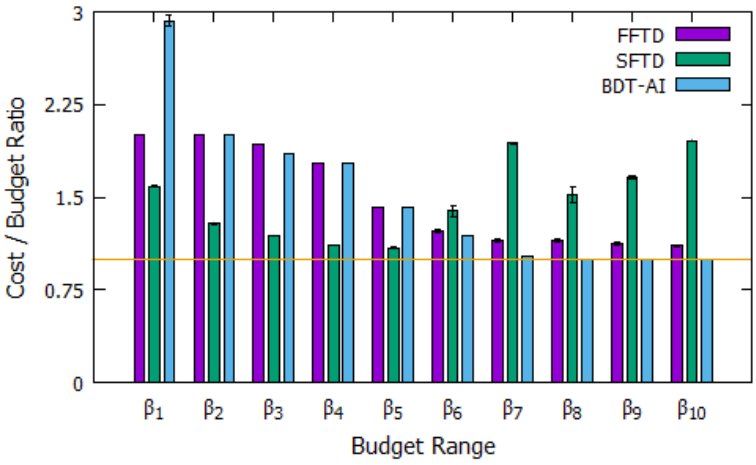


(a) Cost / Budget Ratio

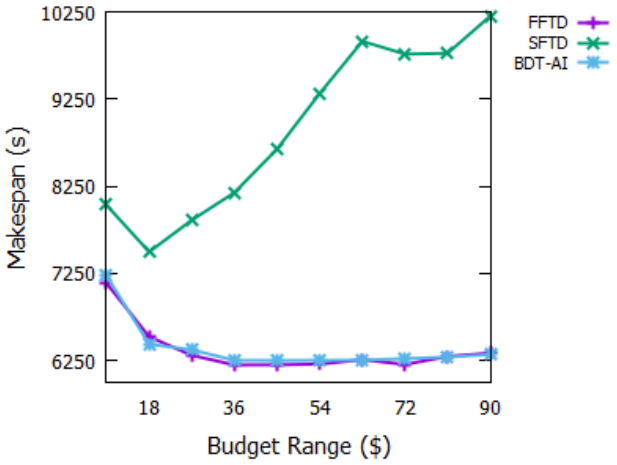


(b) Makespan

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

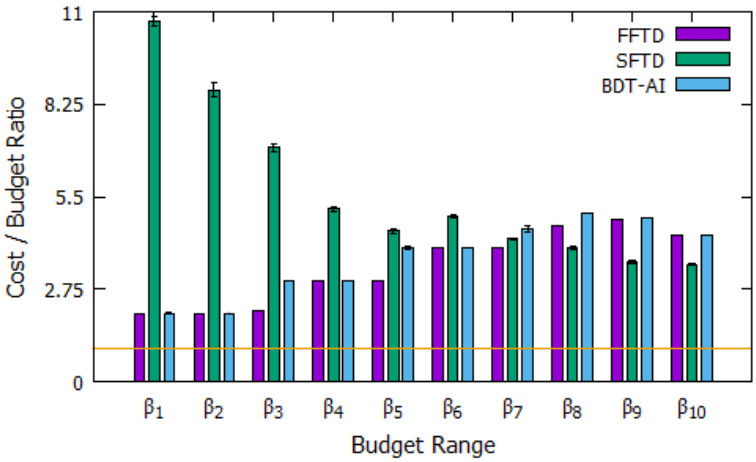


(a) Cost / Budget Ratio

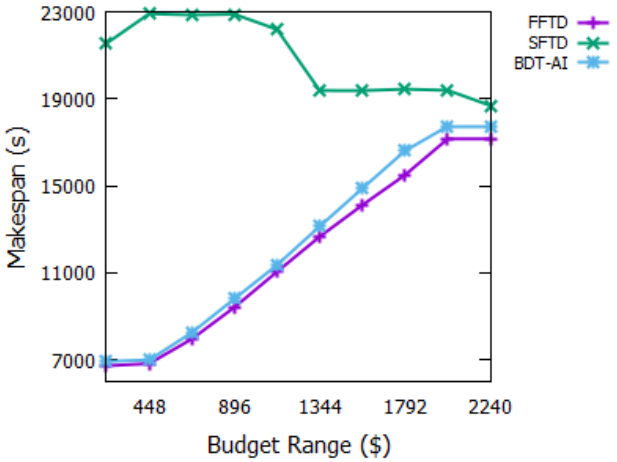


(b) Makespan

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



(a) Cost / Budget Ratio



(b) Makespan

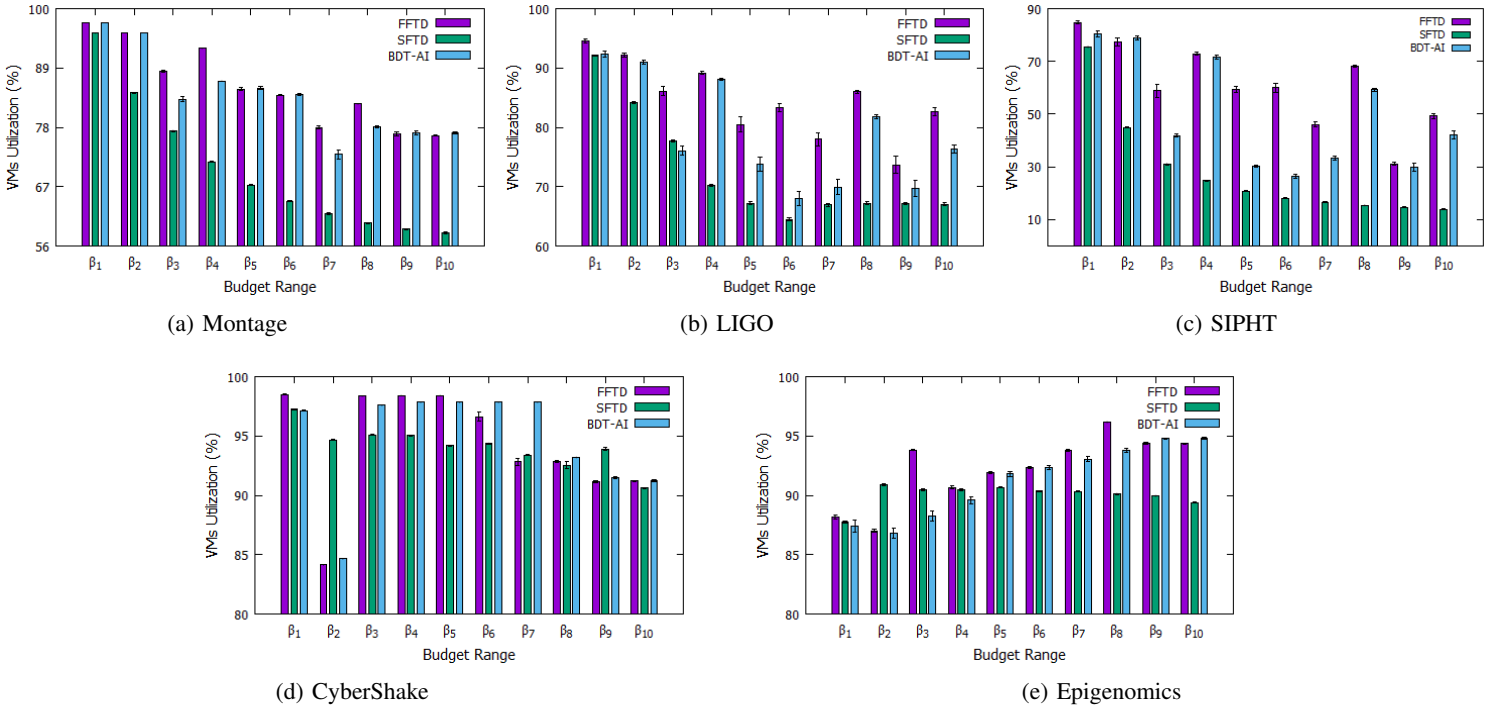
Fig. 8: Cost / Budget Ratio and Makespan Performance of Epigenomics Workflow.

intensive workflow that involves large I/O activities. It is observed from the SFTD results where the ratio values increase as the number of VMs being added. It means, there are huge number of I/O data transfer that causes the overhead. Although all algorithms consider data transfer time when estimating the task's processing time, the network traffic congestion caused unpredicted overhead. Furthermore, the number of running VMs is proportional to the number of I/O activities, hence, SFTD that provisions larger number of VMs than the other algorithms are affected a lot by the network congestion. It is worthwhile mentioning that the ratio values for FFTD and BDT-AI decrease as the budget becomes more relaxed. While SFTD allocates more VMs as the budget increases, FFTD and BDT-AI use the additional budget to provision faster VM type.

The Epigenomics workflow result is shown in Fig. 8a. SFTD shows very high cost / budget ratio values for the earlier budget intervals; however, the ratios consistently decrease as the budget increases. The possible reason is that the number of VMs provisioned by the SFTD remains relatively constant throughout the budget intervals while the algorithm chooses faster VM types as the budget increases while the other algorithms do the opposite. Similar reason with the previous CyberShake case is possible, in which most of the tasks in Epigenomics workflow are also characterized as I/O and CPU intensive. Hence, provisioning more VMs as the budget increases is not improving the performance as seen in FFTD and BDT-AI cases. However, FFTD outperforms the other two algorithms in terms of meeting the budget constraint for all cases.

Overall, FFTD demonstrates equal or better performance than BDT-AI in 88% of the cases in terms of 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.

2) Makespan Evaluation: The first half of budget interval in the Montage workflow (Fig. 4b) shows the marginal difference in makespan. Most likely, this is due to the characteristics of the tasks in Montage workflow that highly depend on the I/O rather than CPU processing. Hence, choosing faster VM types



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

(d) CyberShake (e) Epigenomics

Fig. 9: VMs Utilization for Different Workflow Applications.

are not significantly affecting the makespan. However, as the budget increases, the second half of budget interval shows that the difference is wider. The possible reason is due to the high number of VMs provisioned by SFTD during the runtime contributes to the performance degradation. However, the congestion caused by the I/O activities is not significant since Montage is not a data intensive workflow. Finally, FFTD obtains lower makespan than BDT-AI in 70% of the cases in which both algorithms obtain the same performance in terms of 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 clearer observed. FFTD shows the lowest makespan for all cases. Then, the results obtained for the SIPHT workflow are depicted in Fig. 6b. Similar with the previous results, FFTD obtains the lowest makespan too and shows more stable results than BDT-AI.

The results obtained for the CyberShake workflow are shown in Fig. 7b. FFTD produces slightly lower makespan in 60% of the cases than BDT-AI while SFTD performs the worst. CyberShake is considered as a data intensive workflow that involves high number of data transfer activities. This explains bad SFTD performance as it provisions a large 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 scenario. A possible explanation is the fact that Epigenomics consists of tasks that are both CPU and I/O intensive. Having low number of VMs provisioned is the suitable approach for executing the workflow. It needs to be noted that this behaviour should be a guide for the users when defining the budget for Epigenomics workflow. Finally, FFTD shows the lowest makespan for all scenario.

Overall scenario, FFTD demonstrates lower makespan in 84% of all cases than the other two algorithms. In the scenario where FFTD gets equal performance with BDT-AI in terms of meeting the budget constraint, it demonstrates lower makespan in 80% of the cases. Meanwhile, in the cases where FFTD gets lower cost / budget ratio that is normally should be traded-off by getting higher makespan, it also successfully obtains lower makespan than BDT-AI in 93% of the cases.

B. VM Utilization

To better understand the behaviour of the algorithms, we analyzed the average VM utilization for each workflow. High VM utilization means the algorithm can map the task to VM efficiently and utilizes the idle time slots. Hence, this performance metric is suitable to evaluate the algorithm's ability in dealing 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 obtains the highest VM utilization in all cases. On average, the SIPHT workflow case shows the lowest VM utilization of all experiments. However, FFTD obtains higher VM utilization in 90% of the cases than BDT-AI for SIPHT. Furthermore, BDT-AI presents higher VM utilization in 60% of the cases than FFTD in the CyberShake workflow, it confirms 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 scenario, in 72% of the cases, FFTD demonstrates better performance than BDT-AI in terms of VM utilization. However in the CyberShake workflow, BDT-AI shows competitively better performance than FFTD. A possible explanation is because the level-based strategy of BDT-AI works best for the data intensive workflow that has level-wide structure as the CyberShake.

VI. Conclusions and Future Work

In this paper, we present task-based budget distribution strategies for executing scientific workflows in IaaS clouds with coarse-grained billing periods. The problem is modelled 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 into each individual task and drives the resources usage through the sub-budget of each task. It schedules the tasks whenever their parents tasks are finished and the input data are available for execution based on their Earliest Finish Time (EFT). 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 unavailability of reusable idle VMs. Every time a task is finished, the algorithm considers the budget spent so far and adjusts the next task scheduling decision if necessary. For each task that reuses idle VMs, unused sub-budget is kept as spare budget and utilized to update the budget distribution or to lease faster VM type in the resource provisioning phase.

The performance evaluation results demonstrate that our solution has an overall better performance than state-of-the-art algorithm. It is successfully obtaining 88% equal or better performance in terms of cost / budget ratio values and gaining lower makespan for 84% of the cases while presents higher VM utilization in 72% of the staged experiments. As future works, we will investigate this approach on 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] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011, pp. 1–12.

[3] 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.

[4] 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.

[5] V. Arabnejad, K. Bubendorfer, and B. Ng, "Deadline distribution strategies for scientific workflow scheduling in commercial clouds," in *Proceedings of 9th ACM/IEEE International Conference on Utility and Cloud Computing*. ACM, IEEE, 2016, pp. 70–78.

[6] 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.

[7] 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

[8] 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.

[9] 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.

[10] X. Wang, B. Cao, C. Hou, L. Xiong, and J. Fan, "Scheduling budget constrained cloud workflows with particle swarm optimization," in *Proceedings of IEEE Conference on Collaboration and Internet Computing*. IEEE, 2015, pp. 219–226.

[11] A. Verma and S. Kaushal, "Budget constrained priority based genetic algorithm for workflow scheduling in cloud," in *Proceedings of 5th International Conference on Advances in Recent Technologies in Communication and Computing*. IET, 2013, pp. 216–222.

[12] 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.

[13] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget distribution strategies for scientific workflow scheduling in commercial clouds," in *Proceedings of 12th IEEE International Conference on e-Science*. IEEE, 2016, pp. 137–146.

[14] M. A. Rodriguez and R. Buyya, "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

[15] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proceedings of 1st IEEE International Conference on e-Science and Grid Computing*. IEEE, 2005, pp. 8–pp.

[16] 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.

[17] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the ACM/IEEE Conference on Supercomputing*. IEEE Press, 2008, p. 50.

[18] S. E. Whitcomb and M. E. Zucker, "Ligo: The laser interferometer gravitational-wave observatory," *Astrophys. J*, vol. 353, p. 344, 1990.

[19] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas," *PloS one*, vol. 3, no. 9, p. e3197, 2008.

[20] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner *et al.*, "Cybershake: A physics-based seismic hazard model for southern california," *Pure and Applied Geophysics*, vol. 168, no. 3-4, pp. 367–381, 2011.

[21] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013.

[22] 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.

[23] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proceedings of 5th IEEE International Conference on Cloud Computing*. IEEE, 2012, pp. 423–430.

[24] 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 2nd IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 159–168.