A Budget-Aware algorithm for Scheduling Scientific Workflows in Cloud

Vahid Arabnejad, Kris Bubendorfer and Bryan Ng
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
Email: {vahid, kris and bryan.Ng}@ecs.vuw.ac.nz

Abstract—**Commercial clouds are quickly becoming the goto platform for hosting on-demand and dynamically scalable services for scientific analyses and computation. Dynamic provisioning of resources is a critical element when utilising the cloud for executing large-scale and complex scientific analyses. In particular, hosting and managing data intensive applications on the cloud raises new challenges in terms of workflow scheduling. In this paper, we introduce a new Budget Aware Trickling (BAT) algorithm that addresses eScience workflow scheduling in the cloud. Our main focus in this paper is on data intensive applications that appear in scientific domains dealing with a large amount of data. The BAT algorithm builds upon the concept of Constrained Critical Paths (CCP) to execute a set of tasks on the same instance to lower the cost of communication and data movement between instances. Our BAT algorithm distributes budget based on the dependency structure inherent in workflows and we show that it yields 30% reduction in makespan while maintaining consistent success rate.**

I. Introduction

The scale and scope of cloud enabled scientific research is increasing dramatically due to the cloud offering pay-per-use high performance computing facilities. Such cloud enabled research often produces a vast amount of data that requires large scale computing resources for execution. Data intensive computing is defined as production, manipulation and analysis of data from mega bytes to peta bytes [1]. Data intensive applications in different domains from science to social networking, produce large scale data that need to be analyzed and processed with parallel processing and distributed techniques. Data operations consist loading input files, data processing, distribution and aggregation, and execution is typically modelled and characterized by workflows.

Scientific workflows are one of the key technologies in the development of data intensive scientific experiments. Scientific workflows vary in size from a couple of tasks to millions of tasks with heterogeneous characteristics in terms of resource demands and dependencies. The cloud provides on demand pay-per-use provisioning of a range of instance types, no matter where the services or requestor are hosted. Advantages and benefits of using cloud computing have resulted in a move towards the hosting of e-Science applications on the cloud [2].

Once a scientific experiment is defined by a workflow, it needs to be executed – which requires resources to be requested from and provisioned by the cloud. This is essentially a problem of ensuring an appropriate set of instances is provisioned and then scheduling tasks to those instances. This is a combinatorial problem involving multiple constraint satisfaction and is *NP*-complete. In addition to the heterogeneous task execution requirements, dependencies and instance types, other constraints that are generally considered significant in scientific domains are execution time and cost. The resource provisioning phase determines the amount and type of resources required and requests the resources for workflow execution. Resource allocation is defined as the assignment of reserved resources to tasks in the workflow. The scheduling phase then maps the workflow tasks to the provisioned resources.

The financial cost and total execution time of a workflow depends on the number and types of instances requested during resource provisioning. The cost plays a significant role in a cloud environment as users wish to minimise costs, as grant budgets are finite. In this paper we will focus on the issue of scheduling budget constrained workflows on commercial pay-per-use clouds while trading off cost and time. In essence, the goal is how to best spend the budget for the best performance.

The main contribution of this paper is the development and evaluation of our novel, budget constrained scheduling algorithm – Budget Aware Trickling (BAT). The BAT algorithm manages the scheduling of workloads on dynamically provisioned cloud resources and achieves a reduction in makespan, most significantly, when the budget is limited. In particular we consider:

- Budget distribution: distributing budget based on the dependency structure embedded in a workflow.
- Trickling: trickling unspent budget (residuals) down to unscheduled tasks to better utilize the budget.
- Task selection: selecting Constrained Critical Paths (CCP) to execute a set of tasks on the same instance with the goal of reducing communication cost between instances.

The remainder of this paper is organized as follows: Section II gives an overview of existing approaches to scheduling workflows. In Section III, we define the workflow scheduling problem and describe our system model. In Section IV, we present our workflow scheduling algorithm. In Section V and VI, we outline our CloudSim-based simulation followed by results and performance evaluation. Finally, we summarize our work in Section VII.

II. Related Work

In cloud environments, there have been significant research on QoS constrained workflow scheduling for deadline and budget. The main idea of most deadline constrained algorithms is how to distribute the user-defined deadline among workflow tasks or levels [3]–[6]. There are several recent and comprehensive surveys [7], [8] on workflow scheduling. We only include a review of workflow schedulers that take into account budget constraints in a pay-per-use cloud environment.

The related research in the this area can be categorized into two main classes: cost optimization and budget constrained. This classification helps understand the variety of existing in cloud scheduling algorithms by exposing its strategies and objectives.

A. Cost optimization

In [9], Li et al. presented an extension of the HEFT [10] algorithm called the Cost Conscious Scheduling Heuristic (CCSH). The CCSH algorithm, first constructs a priority list of tasks and then assigns the task with the highest priority value to the most cost-efficient virtual machine (VM). However, only one VM type and one pricing model is considered.

A security-aware and budget-aware workflow scheduling scheme is presented in [11] while trying to minimize the workflow execution time. However, in their presented algorithms, they do not control the budget if in any steps the algorithm exceeds the user-defined budget. For this purpose, we categorize [11] in the cost optimization category.

A cost efficient task scheduling heuristic is presented in [12]. However, the cost model that is considered in this paper is based on the number of used CPU cycles. The total cost in this model is the sum of the costs of all tasks. This model is not consistent with most cloud providers, like Amazon, that charge users for a minimum period of time, even if the instance is only used for a fraction of the minimum period.

B. Budget Constrained

The closest work to ours appears in [13] and [14] whereby Zheng et al. put forward the concept Budget constrained Heterogeneous Earliest Finish Time (BHEFT) which is an extension of HEFT algorithm [10]. In BHEFT, a current task budget (CTB) factor is introduced to distribute spare budget among unscheduled tasks. Their budget distribution is different from ours as the task budget and spare budget are calculated task by task. The task selection part in [14] is one by one based on upward ranking [10]. In our case, a group of tasks, termed as Constrained Critical Path (CCP) is selected to lower the cost of communication and data movement. Moreover, their work is set within the context of a Grid environment which is not directly applicable to cloud environments. Workflow scheduling in cloud differs from grid primarily in the elastic resource provisioning and pay-per-use charging model.

Scheduling Bags of tasks under budget constraints in cloud was presented in [15]. One of the assumptions in [15] is that tasks are preemptive which means they can be interrupted, delayed and then re-triggered sometime later. Our model is different from [15] in such a way, we have non-preemptive dependent tasks in our workflow model, a less forgiving constraint.

In [16], Zeng et al. presented a budget-aware backtracking algorithm for executing large scale many task workflows, referred to as ScaleStar. The cost model considered in [16] is based on the use of fractional resources. However, in most of the cloud providers, like Amazon EC2, it is more realistic to consider costs based on a longer interval for example 60 or 90 minutes.

A budget constrained auto scaling multiple workflows in cloud is considered in [17] by Mao et al. Their work is different from ours in the way that they consider available budget in the form of dollar per time units. The same authors go on to propose two new auto scaling techniques in [18] to solve the budget constrained scheduling for a workload consisting multiple workflows. In this work, budget is distributed to different workflows proportionally based on assigned priorities.

The authors in [19] and [20] presented an algorithm with budget constraints called minimum end-to-end delay under cost constraint (MED-CC). Firstly, each task in a workflow is assigned to an instance. In the next step, all critical tasks are considered for rescheduling with the proposed Critical Greedy algorithm. The algorithm in [20] was chosen in order to evaluate the performance of our BAT algorithm by comparison.

III. Problem Setup

This section is broken down into three parts to facilitate the explanation on the problem setup, namely: (i) the workflow model (ii) definitions of variables and (iii) cloud service model.

A. Workflow Model

A Directed Acyclic Graph (DAG) is the most common representation of a workflow [21]. A workflow is defined as a graph $G = (T, E)$ where $T = \{t_0, t_1, ..., t_n\}$ is a set of tasks represented by vertices and $E = \{e_{i,j} \mid t_i, t_j \in T\}$ is a set directed edges denoting data or control dependencies between tasks t_i and t_j.

An edge $e_{i,j} \in E$ represents the precedence constraint as a directed arc between two tasks t_i and t_j where $t_i, t_j \in T$. The edge indicates that task t_j can start only after completing the execution of task t_i with all data received from t_i and this implies that task t_i is the parent of task t_j, and task t_j is the successor or child of task t_i. Each task can have one or more parents or children.

B. Definitions

The Earliest Start Time (EST) of a task t_i is calculated on the instance with the shortest execution time and defined as:

$$EST(i) = \begin{cases} 0 & , t_i = t_{entry} \\ \max\limits_{t_j \in pred(t_i)} \left\{ EST(t_j) + w_{t_j} + C_{i,j} \right\} & \text{, Otherwise,} \end{cases} \tag{1}$$

where w_{t_j} is the execution time of task t_j on the fastest instance type.

The $\mathrm{ECT}(t_i)$ denotes the Earliest Completion Time (ECT) of task t_i over all instances and defined as:

$$\mathrm{ECT}(t_i) = EST\,(t_i) + w_{t_i}. \tag{2}$$

The cost of executing task t_i on instance p_j is calculated as:

$$TaskCost_{t_i}^{p_j} = \left\lceil \frac{w_{t_i}^{p_j}}{N_t} \right\rceil * c_j, \tag{3}$$

where c_j is the cost of instance p_j for one time interval. The execution time of task t_j on instance p_j is denoted by $w_{t_i}^{p_j}$ and N_t is the number of intervals.

The overall cost of executing all tasks in a workflow is defined as:

$$Cost_o = \sum_{t_i \in G} TaskCost_{t_i}^{p_j}. \tag{4}$$

Constrained Critical Path (CCP): A Critical Path (CP) is the longest path from the entry to exit node of a task graph [22]. The length of critical path ($|CP|$) is calculated as the sum of computation costs and communication costs, and can be considered as the lower bound for scheduling a workflow. The set of tasks containing only the tasks ready for scheduling constitutes a constrained critical path (CCP) [23]. A task is ready when all its parents have been executed and all data required by the task has been provided.

C. Cloud service model

We assume that cloud vendors provide access to unlimited number of instances and the instances are heterogeneous (denoted by $P = \{p_0, p_1 \ldots p_h\}$, where h is the index of the instance type). We also assume that all instances and storage services are located in the same region and also assume that the average bandwidth between the instances are identical.

IV. The BAT algorithm

Our budget-Aware Trickling (BAT) scheduling algorithm attempts to minimize the execution time while meeting the budget constraints. The BAT algorithm recognizes the significance of levels in task scheduling and thus distributes budget share based on levels.

The BAT algorithm is divided into four main phases:

(A) Workflow partitioning: The workflow is partitioned into dependency free bags of tasks, called levels.
(B) Budget Distribution: The user-defined budget is then allocated to each defined level.
(C) Task Selection: A set of tasks expressed in Constrained Critical Path (CCP) is selected based on its priority in the ready list for execution.
(D) Instance Selection: The instances are chosen to meet the available budget.

A. Workflow partitioning

The workflow partitioning process maximizes task parallelism by arranging tasks in levels, where within each level no tasks have dependencies on another in the same level. Each level can therefore be thought of as a bag of tasks (BoT) containing a set of independent tasks.

We describe the level of a task t_i as an integer representing the maximum number of edges in the paths from task t_i to the exit task. The level number (denoted by N_L) associates a task to a BoT as follows:

$$N_L\,(t_i) = \max_{t_j \in succ(t_i)} \{N_L\,(t_j) + 1\}, \tag{5}$$

where $succ(t_i)$ denotes the set of immediate successors of task t_i. We categorize tasks in the bottom-top direction (Level number 1 $\rightarrow$ Level number 5 in Fig 1) to allocate all tasks into different levels. For the exit task, the level number is always 1.

All tasks are then grouped into Task Level Sets (TLS) based on their levels and this is expressed as:

$$\mathrm{TLS}(\ell) = \{t_i | N_L\,(t_i) = \ell\}, \tag{6}$$

where ℓ is an integer denoting the level in $[1 \ldots N_L\,(t_{entry})]$.

B. Budget Distribution

The main idea of budget distribution is simple as distribute budget among different levels and try to schedule each task on an instance considering the available sub-budget assigned to a level. In this paper, we used two strategies to distribute budget among levels:

- Area: In this strategy, the combination of height and width for a workflow is considered. This is informed by the supposition that the structure of a workflow impacts scheduling. The area strategy takes into account: (i) the height of the workflow whereby budget distribution in each level is proportional to its distance from the entry node (ii) the width of the workflow whereby budget distribution is proportional to the number of tasks within that level.
- All in: Places the entire budget on the entry level and any remainders are trickled down to later levels.

C. Task Selection

Tasks in a workflow are selected for execution based on the CCP. To find all CCPs in a workflow we use the *upward rank* and *downward rank* introduced in our previous work [6], and these are defined as follows:
modified upward rank :

$$Mrank_u(t_i) = \overline{w_i} + \sum_{t_j \in succ(t_i)} (\overline{c_{i,j}}) + \max_{t_j \in succ(t_i)} (rank_u(t_j)) \tag{7}$$

modified downward rank :

$$Mrank_d(t_i) = \sum_{t_k \in pred(t_i)} (\overline{c_{k,i}}) + \max_{t_k \in pred(t_i)} (\overline{w_k} + rank_d(t_k)) \tag{8}$$

where $\overline{w_i}$ and $\overline{c_{i,j}}$ are the average execution time and average communication time of task t_i, respectively.

The modified rank aggregates a task's predecessors' or successors' communication time instead of selecting the maximum (as in [10]). With the modified ranks, those tasks with higher out-degree or in-degree have higher priorities. As a result, they have a greater chance to execute first and more tasks on the next CCP can be considered as ready tasks. All tasks are first sorted based on the sum of $Mrank_d(t_i) + Mrank_u(t_i)$ values. Tasks with highest values are chosen as the first CP. A more detailed explanation on CCP can be found in our previous work [6].

D. Instance Selection

In the Instance selection phase we aim to identify the most appropriate instance to execute CCPs. Note: all tasks in a CCP are executed in the same instance with the goal of avoiding communication cost between them. We start by calculating both the time and the cost of executing each task on each instance type, given by equations 10 and 11, forming two sets of Cost and Time (the sets are indexed by p_j and (CCP_i)).

The cost for executing current CCP, denoted by (CCP_i), on the instance p_j is given by $C(CCP_i, p_j)$. In the equation 10, $subBudget_{CCP_i}$ is the sum of assigned $subBudget$ of all tasks in different levels for a CCP which is given by:

$$subBudget_{CCP_i} = \sum_{\substack{t_i \in CCP_i \\ t_i \in \mathrm{TLS}(\ell)}} subBudget(\ell). \tag{9}$$

$$\mathrm{Cost}^{p_j}_{CCP_i} = \frac{subBudget_{CCP_i} - C_{(CCP_i,p_j)}}{subBudget_{CCP_i} - C_{best}}, \tag{10}$$

where C_{best} denotes the minimum cost for executing current CCP among all instances.

In the Time set, the required time for the current CCP on instance p_j is a function of $\mathrm{ECT}(CCP_i, p_j)$ and this is expressed in equation 11. The ECT is the earliest time that a CCP can complete execution on a instance (defined in equation 2 for a single task). The maximum and minimum completion time of executing the CCP_i among all instances are $\mathrm{ECT}(max)$ and $\mathrm{ECT}(min)$, respectively.

$$\mathrm{Time}\ ^{p_j}_{CCP_i} = \frac{\mathrm{ECT}(max) - \mathrm{ECT}(CCP_i, p_j)}{\mathrm{ECT}(max) - \mathrm{ECT}(min)}. \tag{11}$$

To find the best instance, we use the Time Cost Adjustment Factor (TCAF) in equation 12.

$$\mathrm{TCAF}\ ^{p_j}_{CCP_i} = \begin{cases} 0 & , \mathrm{Cost}\ ^{p_j}_{CCP_i} = 0 \\ \dfrac{\mathrm{Time}\ ^{p_j}_{CCP_i}}{\mathrm{Cost}\ ^{p_j}_{CCP_i}} & , \text{Otherwise.} \end{cases} \tag{12}$$

The instance with the highest TCAF value is considered the best candidate to execute the current CCP. There is a possibility that the total assigned budget for the level ℓ has already been spent ($subBudget_{CCP_i}$=0) while there are still some unscheduled tasks. If this condition is occurs, the $\mathrm{Cost}^{p_j}_{CCP_i}$ in equation 10 is zero. Therefore, we can not launch a new instance as there is no budget left. Note that the value of equation 12 becomes zero as well. Moreover, if the value of equation 10 becomes negative, it means the cost of execution on the selected vm is higher than available $subBudget_{CCP_i}$.

When an instance is provisioned, the user is charged for the entire billing interval even if the task completes before the end of the interval. One way to reduce the cost of executing tasks is by using leftover capacity (residuals) in provisioned instances that have been already paid for. Therefore, if other tasks can execute on an existing instance with a residual, their execution costs can be considered zero. Moreover, the utilization of cloud resources depends on how tasks are placed together. Instance fragmentation and resource wastage occurs if tasks are not packed efficiently. The BAT algorithm utilises these residuals for executing ready tasks, which reduces makespan at no additional cost.

After assigning a CCP to an instance that requires $C(CCP_i, p_j)$ cost, we update the remaining budget for each level. Allocating a higher budget to the earliest tasks in a workflow generally leads to a lower makespan. The pseudo code presented in algorithm 1 explains how the cost is updated.

Algorithm 1 **Update Remaining budget**

```
1: procedure UPDATE BUDGET(CCP, cost)
2:     while $|CCP|$ and $cost > 0$ do
3:         $t_i \longleftarrow$ last task in CCP
4:         remove $t_i$ from CCP
5:         $LB \longleftarrow subBudget(\ell)_{t_i \in \mathrm{TLS}(\ell)}$
6:         if $LB > cost$ then
7:             $LB \longleftarrow LB - cost$
8:             $cost \longleftarrow 0$
9:         else
10:            $cost \longleftarrow cost - LB$
11:            $LB \longleftarrow 0$
12:        end if
13:    end while
14: end procedure
```

An important concept in our algorithm is trickling down unused budget and this is expressed by equation 13. We define Spare Level Budget (SLB) as the amount of money remains after allocating all tasks in the level ℓ and this is expressed as:

$$SLB_\ell = subBudget_\ell - \sum_{t_i \in \mathrm{TLS}(\ell)} C_i. \tag{13}$$

The leftover budget is trickled to the next level ($\ell + 1$).

E. An illustrative example:

We now present an example to show how the budget is distributed among different levels. Figure 1 shows the structure of a sample workflow with ten tasks and their dependencies.

In this figure, the left column shows level numbers calculated by equation 5. The right column is obtained by counting tasks in each level starts from the exit task. In this example N_{max}=5 which is the maximum level in the workflow and a budget of 165 is assumed.

- Height: Each level is assigned a weighted share of budget relative to its height in the workflow. This is calculated by:

$$L_{weight} = \sum_{k=1}^{N_{max}=5} k = 15.$$

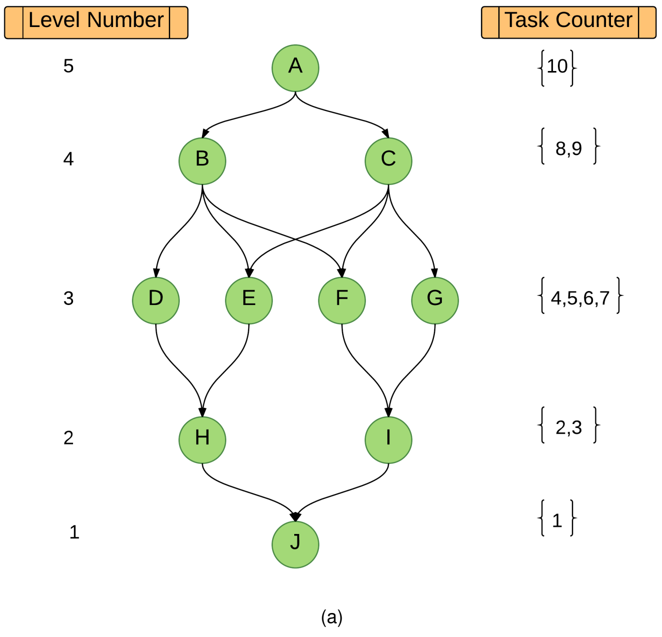


Fig. 1: A Sample Workflow with ten tasks.

The Budget Factor (BF) is calculated by:

$$BF = \frac{budget}{L_{weight}} = \frac{165}{15} = 11.$$

For instance, level 4 consists of tasks B and C are assigned a share of the budget equal to $4 \times BF = 4 \times 11 = 44$.

- Width Proportional: Each level gets a share of budget depending the number of tasks in corresponding level:

$$BF = \frac{budget}{tasknumbers} = \frac{165}{10} = 16.5.$$

For instance, the budget share assigned to level 4 with two tasks is $2 \times BF = 2 \times 16.5 = 33$.

- Area Proportional: In this strategy, the budget share allocated to each level is a combination of height and width strategies. Calculated by:

$$L_{weight} = \sum_{k=1}^{10} k = 55.$$

The Budget Factor (BF) is calculated by:

$$BF = \frac{budget}{L_{weight}} = \frac{165}{55} = 3.$$

The budget then is distributed based on the sum of numbers in the right column in Fig. 1. For example, level 3 is allocated the share $(4+5+6+7) \times BF = 22 \times 3 = 66$.

- All in: The total budget is assigned to level 5. After scheduling all tasks in this level, any spare budget is trickled to the next level.

TABLE I: Budget distribution for each strategy over each level for a total budget of 165 in Figure 1.

	Budget Distribution Strategy			
	Height	Width	Area	"All in"
Level 5	$5 \times BF = 55$	$1 \times BF = 16.5$	$10 \times BF = 30$	165
Level 4	$4 \times BF = 44$	$2 \times BF = 33$	$17 \times BF = 51$	0
Level 3	$3 \times BF = 33$	$4 \times BF = 66$	$22 \times BF = 66$	0
Level 2	$2 \times BF = 22$	$2 \times BF = 33$	$5 \times BF = 15$	0
Level 1	$1 \times BF = 11$	$1 \times BF = 16.5$	$1 \times BF = 3$	0

V. Evaluation

Public clouds provide instance types containing different amounts of CPU, memory, storage and network bandwidth at different prices. In this paper we use a resource model based on the Amazon Elastic Compute cloud, where instances are provisioned on demand. The pricing model is a pay-as-you-go with minimum hourly billing. Under this pricing model, if an instance is used for one minute, a user pays for the whole hour. The costs and instance types used in this paper are given in Table II, and were accurate in March 2016.

TABLE II: Instance Types

Type	ECU	Memory(GB)	Cost($)
m3.medium	3	3.75	0.067
c4.large	8	3.75	0.105
c3.xlarge	14	7.5	0.21
m4.2xlarge	26	32	0.479
c4.4xlarge	62	30	0.838
c3.8xlarge	108	60	1.68

Our simulation scenario is configured as a single data-center with six different instance types. The characteristics of the instances are based on the Amazon EC2 instance configurations presented in Table II. The average bandwidth between instances is fixed to 20 MBps [24]. The processing capacity of an EC2 unit is estimated at one Million Floating Point Operations Per Second (MFLOPS) [25]. The estimated execution times are scaled by instance type CPU performance.

In an ideal cloud environment, there is no provisioning delay in resource allocation. However, some factors such as the time of day, operating system, instance type, location of the data center, and number of requested resources at the same time, can cause delays in startup time [26]. Therefore, in our simulation, we adopted a 97-second boot time based on previous measurements of EC2 [26].

We use three common data intensive scientific workflows: Cybershake, Montage and LIGO, to evaluate the performance of our algorithms with a realistic load. The characteristics and task composition of these workflows have been analyzed in [27], [28]. We vary the budget from tight to relaxed and record the both the cost and suceess rate. Additionally, we calculate the fastest schedule (denoted by FS) as a baseline schedule.

Effectively, this baseline is the fastest possible execution - ignoring costs and is computed as:

$$FS = \sum_{t_i \in CP} (w_i^j), \tag{14}$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j. If all tasks on the CP of a workflow are executed on the fastest instance type, the fastest schedule will be achieved.

We define the budget as a function of the fastest schedule and this budget is expressed in equation 15 in which the budget varies from tight to relaxed:

$$\text{budget} = \alpha * FS, \quad 1 < \alpha < 6. \tag{15}$$

The budget factor α starts from 1 to consider very tight budgets (typically approaches the fastest schedule) and is increased by one up to a value of 6, which results in relaxed budget.

The Amazon EC2 instances charge on an hourly interval from the time of provisioning. We configure our simulator to reflect this charging model and we use a time interval of 60 minutes in our simulations. To compare performance with respect to different workflow sizes we evaluated workflows with 50, 100, 200, 500 and 1000 tasks. However, as these results did not vary significantly we present here only workflows with 1000 tasks.

We used the Pegasus workflow generator [27] to create representative workflows of Cybershake, Montage and LIGO. For each workflow structure, and each budget factor, 100 distinct Pegasus generated workflows were scheduled in CloudSIM and the performance of the scheduling algorithms are detailed in the following section.

VI. Experimental Results

In this section we compare the performance of both strategies of our BAT algorithm (in conjunction with two strategies Area and All in) with Critical-Greedy (CG) [20]. The CG algorithm is one of few works addressing workflow scheduling with budget constraint and presented in cloud environment. The makespan and success rates of three data intensive workflows are presented in Fig. 2. The budget vs. execution time (makeSpan) performance of each algorithm is the most significant basis for evaluating their performance. Increasing the budget allows a scheduler to launch more powerful instances with better storage, CPU, and network performance. Therefore, its not surprising to see a moderate reduction in overall workflow execution time as the budget increases.

In terms of makespan, the All in strategy of BAT algorithm has the best performance in all datasets. The CG algorithm [20] performs rescheduling and attempts to reassign tasks until the specified budget is met. This feature of the CG increases the time complexity of the CG algorithm.

The CG algorithm has the worst performance when budget is limited. For example, in the first two budget ranges across all three workflows, CG could only find a schedule with a three times the makespan. The All in and Area budget distribution with BAT algorithm have lower overall makespan than CG while in most cases there is only a small difference between Area and All in strategy. The results show that the BAT algorithm outperforms CG in terms of makespan.

Success rate reflects the success of the algorithm in scheduling a workflow to meet the budget. As expected, relaxing the budget leads to increased success rate. The Area strategy with BAT algorithm has 100% failure in MONTAGE in the first interval, which means it cannot find a schedule. The worst performance in finding a suitable schedule occurs in LIGO - where the CG algorithm records a 20% to 30% failure rate over the first three intervals.

VII. Conclusion

In this paper, we introduce the Budget Aware Trickling (BAT) algorithm for scheduling data intensive workflows in dynamically provisioned cloud resources. Our algorithm focuses on distributing budget based on the dependency structure inherent in workflows. Moreover, the BAT algorithm builds upon the concept of Constrained Critical Paths (CCP) to execute a set of tasks on the same instance in order to lower the cost of communication and data movement between instances. The makespan and success rates are evaluated for our algorithm using three data intensive real word workflows. Assigning more budget to the earliest levels and utilising the budget trickling mechanism resulted in faster execution (30% reduction in makespan) and overall a better workflow scheduling success rate – leading to more viable workflow schedules at lower cost.

References

[1] R. Moore, T. A. Prince, and M. Ellisman, "Data-intensive computing and digital libraries," Commun. ACM, vol. 41, no. 11, pp. 56–62, Nov. 1998. [Online]. Available: http://doi.acm.org/10.1145/287831.287840

[2] I. Foster, K. Chard, and S. Tuecke, "The discovery cloud: Accelerating and democratizing research on a global scale," in proceedings of the IEEE International Conference on Cloud Engineering, 2016.

[3] 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.

[4] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in e-Science and Grid Computing, 2005. First International Conference on, July 2005, pp. 8 pp.–147.

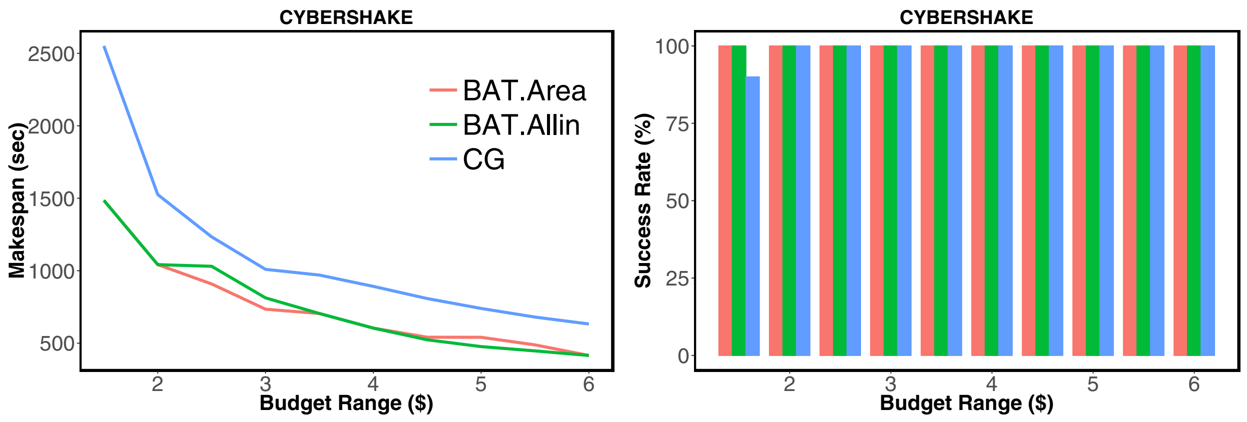
[5] V. Arabnejad and K. Bubendorfer, "Cost effective and deadline constrained scientific workflow scheduling for commercial clouds," in Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on, Sept 2015, pp. 106–113.

[6] V. Arabnejad, K. Bubendorfer, B. Ng, and K. Chard, "A deadline constrained critical path heuristic for cost-effectively scheduling workflows," in 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Dec 2015, pp. 242–250.

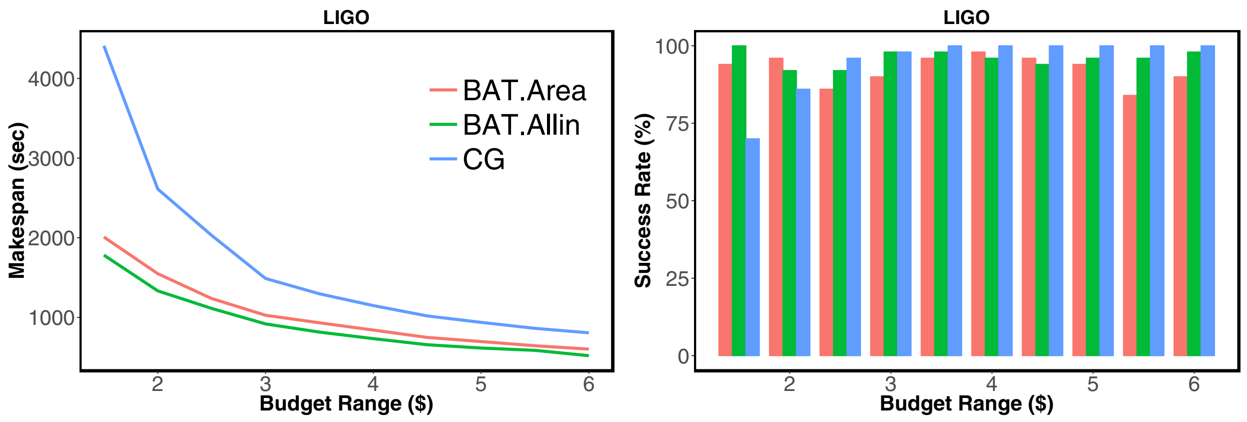
[7] E. N. Alkhanak, S. P. Lee, and S. U. R. Khan, "Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities," Future Generation Computer Systems, 2015.

[8] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," Journal of Systems and Software, vol. 113, pp. 1 – 26, 2016.

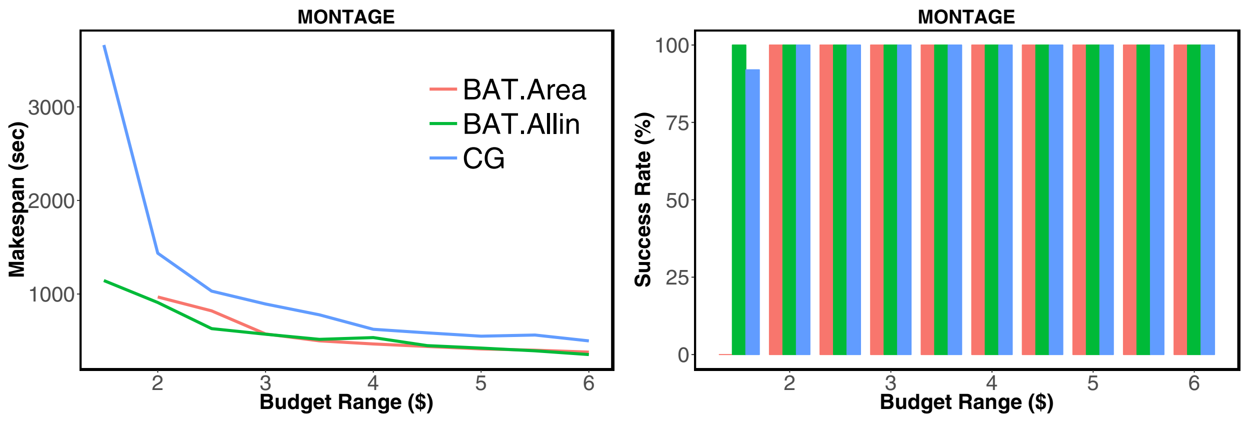
[9] J. Li, S. Su, X. Cheng, Q. Huang, and Z. Zhang, "Cost-conscious scheduling for large graph processing in the cloud," in High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, Sept 2011, pp. 808–813.



(a) Budget vs. MakeSpan and Success Rate for Cybershake



(b) Budget vs. MakeSpan and Success Rate for LIGO



(c) Budget vs. MakeSpan and Success Rate for MONTAGE

Fig. 2: Performance Comparison of algorithms

[10] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, no. 3, pp. 260–274, Mar 2002.

[11] L. Zeng, B. Veeravalli, and X. Li, "Saba: A security-aware and budget-aware workflow scheduling strategy in clouds," Journal of Parallel and Distributed Computing, vol. 75, pp. 141 – 151, 2015.

[12] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," Parallel Computing, vol. 39, no. 45, pp. 177 – 188, 2013.

[13] W. Zheng and R. Sakellariou, Economics of Grids, Clouds, Systems, and Services: 8th International Workshop, GECON 2011, Paphos, Cyprus, December 5, 2011, Revised Selected Papers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Budget-Deadline Constrained Workflow Planning for Admission Control in Market-Oriented Environments, pp. 105–119.

[14] W. Zheng and R. Sakellariou, "Budget-Deadline Constrained Workflow Planning for Admission Control," Journal of Grid Computing, vol. 11, no. 4, pp. 633–651, 2013.

[15] A. M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, Nov 2010, pp. 351–359.

[16] L. Zeng, B. Veeravalli, and X. Li, "Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud," in Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on, March 2012, pp. 534–541.

[17] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, May 2013, pp. 67–78.

[18] ——, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, May 2013, pp. 67–78.

[19] X. Lin and C. Q. Wu, "On scientific workflow scheduling in clouds under budget constraint," in 2013 42nd International Conference on Parallel Processing, Oct 2013, pp. 90–99.

[20] 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, April 2015.

[21] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Workflows for e-Science: Scientific Workflows for Grids. Springer Publishing Company, Incorporated, 2014.

[22] Y.-K. Kwok and L. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," Parallel and Distributed Systems, IEEE Transactions on, vol. 7, no. 5, pp. 506–521, 1996.

[23] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," Parallel Computing, vol. 38, no. 4, pp. 175–193, 2012.

[24] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?" in Proceedings of the 2008 international workshop on Data-aware distributed computing. ACM, 2008, pp. 55–64.

[25] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in Cloud Computing, ser. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, D. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds. Springer Berlin Heidelberg, 2010, vol. 34, pp. 115–131.

[26] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, ser. CLOUD '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 423–430.

[27] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on, Nov 2008, pp. 1–10.

[28] 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, special Section: Recent Developments in High Performance Computing and Security.