# Budget Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds

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*—Scientific research is increasingly reliant on big compute and big data, the fusion of which is known as data intensive science. Large scale scientific analyses are typically represented as workflows which are the typical model for characterizing e-science experiments in distributed systems. Workflows with a large number of tasks are distributed in parallel across computing resources to speed up analyses. The provision of compute capabilities is undergoing a rapid migration from dedicated infrastructure to the cloud. This migration is fuelled by dynamic infrastructure scalability with changes in demand. Cloud instances incur different costs and execution time with different configurations. A key concern for workflow scheduling is to make an appropriate trade-off between these two factors. In this paper, we introduce the Budget Distribution with Trickling (BDT) algorithm that presents new notions for distributing budget based on the dependency structure inherent in workflows. In addition we propose several new strategies for sharing or distributing the budget, and propose trickling to redistribute unspent budget down to other levels. Our results show that biasing the budget distribution to the earlier computation within a workflow will generally produce a lower makespan within budget.**

## I. Introduction

Basic research, and consequently scientific discovery, are in the midst of a disruptive transformation. Research is increasingly reliant on big compute and big data, the fusion of which is known as data intensive science. The execution of analytics on big data over large compute, is most commonly modelled and managed in workflows [1]. Scientific workflows vary in size from a couple of tasks to thousands or million of tasks. Workflows with a large number of tasks are distributed in parallel across computing resources. The provision of compute capabilities is undergoing a rapid migration from dedicated infrastructure to the cloud [2]. Cloud computing enables significant computational leverage to be applied to many real world problems, be they industrial, medical or scientific.

While cloud platforms provide enormous elastic computing capacity, they also pose unique multi-objective scheduling challenges with respect to cost, time and data movement. Efficiently managing pay-per-use heterogeneous cloud infrastructure for large data and compute within research budgets is a challenge that is, or soon will be, faced in almost every research domain.

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 and providers maximise profits. Most cloud providers, like Amazon, charge users for a minimum period of time - even if the instance is only used for a shorter period.

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. One problem in scheduling budget constrained workflow is how to spend that budget for the best performance. This is essentially a budget assignment problem (BAP), and in existing workflow scheduling approaches [3], [4] is shared proportionally based on a subset of the execution characteristic(s) of the task (or cluster of data related tasks) being scheduled, such as, execution time, CPU requirements or memory requirements.

The novelty of our work is that we look at distributing budget based on the dependency structure embedded in the workflow. Essentially we transform the workflow into internally dependency free "bags of tasks" (called levels [5]) and we then distribute the workflow budget over these levels using six strategies. Three of these strategies are designed explicitly for our means of budget distribution and therefore also represent novel work. We ensure that any budget share that is unused by the level to which it is allocated is trickled down to the next level. For the remainder of this paper we will refer to our approach as Budget Distribution with Trickling (BDT).

Based on our results we suggest two hypotheses worthy of further consideration:

**Hypothesis 1** *The earliest tasks in the workflow are the most critical when constructing a schedule.*

**Hypothesis 2** *Assigning a higher budget to the earliest tasks in a workflow generally leads to a lower makespan.*

The rest of the paper is organised as follows: In Section II, we formalize the workflow and system models. In Section III, we present the BDT building blocks, strategies and algorithm. In Section IV, we describe the evaluation method using CloudSim, followed by results and performance evaluation. In section V we present the related work, and finally, we conclude this paper in Section VI.

## II. Workflow and System Models

### A. Workflow Model

A Directed Acyclic Graph (DAG) is the most common representation of a workflow. 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.

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. Task $t_i$ cannot start until all parents have completed.

### B. System Model

Public cloud provides instance types containing various 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 I, and were accurate in March 2016.

TABLE I: 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 |

We assume that cloud vendors provide access to unlimited number of instances and the instances are heterogeneous (denoted by $P = \{p_0, p_1 \dots p_h\}$, 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 is essentially identical.

## III. The Budget-Aware Scheduling Algorithm

In this section, we describe our budget-aware scheduling algorithm, Budget Distribution with Trickling (BDT). The algorithm is divided into four main phases (each of which relates to a following subsection: III-A- III-D):

(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 using one of six different strategies.
(C) Task Selection: A task is selected based on its priority in the ready list for execution.
(D) Instance Selection: The instances are chosen to meet the available budget.

The focus of this paper is on budget distribution, the other phases are included for completeness.

### A. Workflow Partitioning

We aim to maximize task parallelism by arranging tasks in levels, where within each level no tasks have dependencies on another in the same level. such that there are no dependencies between tasks. Each level can therefore be thought of as a bag of tasks (BoT) containing a set of independent tasks.

There are two main algorithms for allocating tasks to different levels, Deadline Bottom Level (DBL) [5] and Deadline Top Level (DTL) [6]. DBL and DBT categorize tasks in bottom-top direction and top-bottom direction, respectively. In this paper, we use the DBL algorithm to partition tasks into different levels.

We describe the level of task $t_i$ as an integer representing the maximum number of edges in the paths from task $t_i$ to the exit task (see Fig. 1). The level number (denoted by $N_L$) associates a task to a BoT. For the exit task, the level number is always 1, and for the other tasks, it is determined by:

$$N_L(t_i) = \max_{t_j \in succ(t_i)} \{N_L(t_j) + 1\} \quad (1)$$

where $succ(t_i)$ denotes the set of immediate successors of task $t_i$. All tasks are then grouped into Task Level Sets (TLS) based on their levels.

$$\mathrm{TLS}(\ell) = \{t_i | N_L(t_i) = \ell\} \quad (2)$$

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

### B. Budget Distribution

As the principle of distributing budget based on the dependency structure of a workflow (levels) is new – we need to evaluate the performance of a variety of strategies to gauge the value of the approach. We start with the most basic strategies, random and uniform, to provide a baseline comparison. We then explore more complex strategies - width, which is an analogue of prior work on proportional schemes, and then the strategies designed specifically for our BDT approach - height, area and "All in".

The most significant differences in each strategy lie in the calculation of the sub-budget. In some strategies, we have a Budget Factor (BF) that determines a share of the budget for each level. Each sub-budget assigned to a level is termed the level budget.

1) Random: The budget is allocated randomly over the levels in the workflow
2) Uniform: Each level gets a $1/L$ share of the budget, where $L$ is the total number of levels.
3) Height Proportional: Each level gets a share of the user budget proportional to its distance from the entry node. The smaller the distance the greater the share.
4) Width Proportional: Each level gets a share of the user budget proportional to the number of tasks within that level.

5) Area Proportional: Combines width and height strategies to set the budget for each level.
6) All in: Places the entire budget on the entry level and any remainders are trickled down to later levels. This is a refined version of Height proportional, that was formulated as an extreme test of hypotheses 1 and 2, rather than a realistic suggestion. We did not expect this strategy to work in the general case, as we anticipated a reduced success rate. However counterintuitively it returned the best overall performance.

We now present an example to show how the budget is distributed among different levels. Random needs no further explanation, so the example will only detail uniform through "All in" strategies. 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 1. 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. Also, a budget of 165 is assumed.

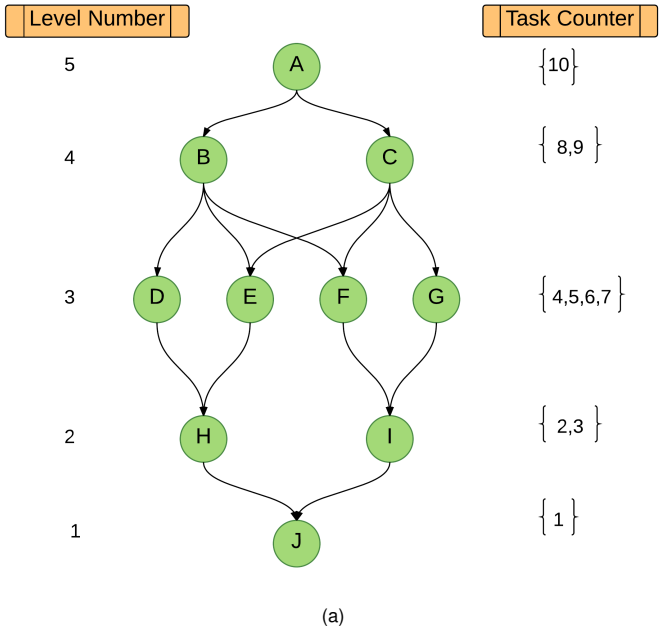



Fig. 1: A Sample Workflow with 10 tasks.

Each strategy distributes the user budget based on the following basis (see Table II for the complete set of budget shares):

- Uniform Proportional: Each level gets 165/5 share of budget as our workflow has five levels.
- Height Proportional: Each level is assigned a weighted share of budget relative to its height in the workflow. This is calculated by:

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

The Budget Factor (BF) is calculated by:

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

For instance, level 4 consisting task 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.

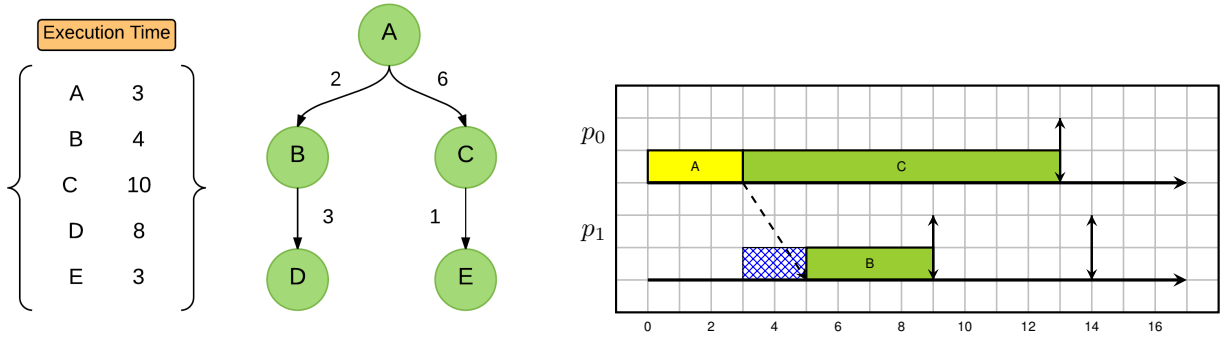
TABLE II: Budget distribution for each strategy over each level for a total budget of 165 in Figure 1.

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

### C. Task Selection

In BDT, tasks are executed level by level which means a task can start execution once all tasks in previous levels have been scheduled. There are no dependencies between tasks that are at the same level. Therefore, all of them are ready to execute and are put to the task ready list. To select a task, at first, all tasks in the ready list should be prioritized. In this paper, tasks are prioritized based on their Earliest Start Time (EST).

It could be argued that it is pointless to prioritize task as all ready tasks are independent and their parent tasks have been




(a) a sample workflow (b) Gant chart

Fig. 2: Task Selection Example

already scheduled, but this is not the case. Let us trace the execution of a sample graph shown in Fig. 2. The number above each edge shows the data transfer time between tasks. Upon executing tasks A, B and C, all its children are placed in the ready list for execution. Note that due to execution of task A and C on the same instance, data is local, and therefore the transfer time is essentially zero. However, task B must wait to receive data from its parent (shown in blue intervals). Task D can start its execution on instance $p_0$ and $p_1$ at time 13 and 9, respectively (shown by ↕). The earliest time that task E can start on instance $p_0$ is 13, and on instance $p_1$ is 14. Therefore, we give a higher priority to task D.

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

$$EST(i) = \begin{cases} 0 & , t_i = t_{entry} \\ \max\limits_{t_j \in pred(t_i)} \left\{ EST(t_j) + w_{t_j} + C_{i,j} \right\} & , \text{otherwise} \end{cases} \quad (3)$$

where $w_{t_j}$ is the execution time of task $t_j$ on the fastest instance type. The amount of data transferred from task $t_i$ to task $t_j$ is called communication time (denoted by $C_{i,j}$).

Task selection starts from the level that consists $t_{entry}$. After executing the first level, all tasks in the next level are put in the ready list to be scheduled.

### D. Instance Selection

The BDT algorithm attempts to minimize the execution time while meeting the budget. Each level receives a computed budget share, that is the maximum that is able to be spent for the tasks within this level. We start by calculating both the time and the cost of executing each task on each instance type, given by equations 4 and 5, forming two sets of Cost and Time.

In the equation 4, $subBudget$ is a share of the budget that assigned to a level. The cost of scheduling for the current task, $t_i$, on the instance $p_j$ is shown by $C_i$. The minimum cost of executing current task among all instances is $C_{best}$.

$$\text{Cost}_{t_i}^{p_j} = \frac{subBudget - C_i}{subBudget - C_{best}}. \quad (4)$$

In the Time set, the required time for the current task on instance $p_j$ is a function of $\text{ECT}(t_i, p_j)$ an is expressed in equation 5. The maximum and minimum completion time of executing the task $t_i$ among all instances are $\text{ECT}(max)$ and $\text{ECT}(min)$, respectively.

$$\text{Time}_{t_i}^{p_j} = \frac{\text{ECT}(max) - \text{ECT}(t_i, p_j)}{\text{ECT}(max) - \text{ECT}(min)}. \quad (5)$$

To find the best instance, we use the Time Cost Trade-off Factor (TCTF) in equation 6.

$$\text{TCTF}_{t_i}^{p_j} = \frac{\text{Time}_{t_i}^{p_j}}{\text{Cost}_{t_i}^{p_j}}. \quad (6)$$

There is a possibility that the total assigned budget for the level $\ell$ has already been spent ($subBudget$=0) while there are still some unscheduled tasks. If this condition is true, it makes equation 4 zero. Therefore, we can not launch a new instance as there is no budget left. Note that the value of equation 6 becomes zero as well.

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 BDT algorithm utilises these residuals for executing ready tasks, which reduces makespan at no additional cost.

An important concept in our algorithm is trickling down unused budget and this is expressed by equation 7. We define Spare Budget (SB) as the amount of money remains after allocating all tasks in the level $\ell$. We then add the leftover to the next level $(\ell + 1)$.

$$SB = subBudget_\ell - \sum_{t_i \in \text{TLS}(\ell)} C_i. \quad (7)$$

## IV. Evaluation

We use five common scientific workflows: Cybershake, Montage, LIGO, Epigenomics and SIPHT to evaluate the performance of our algorithms under realistic load. The characteristics of which have been analyzed in [7]. We use CloudSim [8], configured with one data-center and six different instance types. The characteristics of these instance types are based on the EC2 instance configurations presented in Table I. The average bandwidth between instances is fixed to 20 MBps, based on the average bandwidth provided by AWS [9]. The processing capacity of an EC2 unit is estimated at one Million Floating Point Operations Per Second (MFLOPS) [10].

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 [11]. Therefore, in our simulation, we adopted a 97-second boot time based real world measurements of EC2 [11].

To evaluate the budget sensitivity of the BDT algorithm and associated strategies, we considered different budget ranges for the scientific datasets from lowest possible through to sufficient. The lowest possible budget to schedule a workflow is given by equation 8.

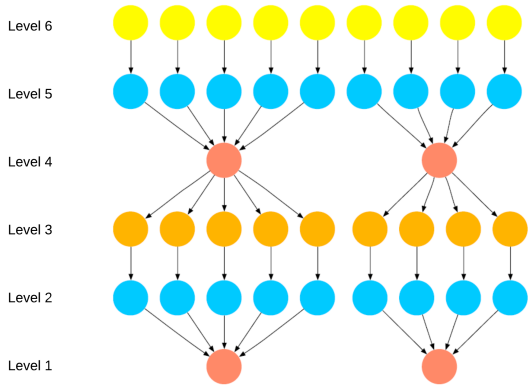
$$Lowset_{cost} = \sum_{\forall t_i \in G} Cost_{t_i} p_j, \tag{8}$$

where $p_j$ is the cheapest instance. To achieve this, all tasks are executed on the instance with the lowest cost (the cheapest instance). This assignment gives us the lowest possible cost required for executing a workflow, irrespective of finishing time. Using this lowest cost, we then calculate the mimimal budget as follows:

$$budget = \alpha * Lowset_{cost} \quad 1 < \alpha < 10. \tag{9}$$

The budget range starts from 1.5 to consider minimum budget with increasing step length of 0.5. The EC2 instances charge hourly basis from the time of provisioning, even if the instance is only used for a fraction of that period. We ran the simulations with lease times of 15, 30, 45 and 60 minutes to evaluate the sensitivity of the algorithm to the length of the lease.

To compare performance with respect to workflow size we evaluated workflows with 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 [7] to create representative synthetic workflows with the same structure as real world scientific workflows (Cybershake, Montage, LIGO and SIPHT). For each workflow structure, and each budget range, 100 distinct Pegasus generated workflows were scheduled in CloudSIM and these results are detailed in the following section.




(a)

Fig. 3: A simple structure of LIGO with six levels

### A. Analysis for LIGO

The Laser Interferometer Gravitational Wave Observatory (LIGO) attempts to detect gravitational waves produced by various events in the universe as per Einsteins theory of general relativity [7]. A simple LIGO-like workflow with six levels is shown in Figure 3.

For a LIGO workflow with 1000 tasks, Table III gives the levels with the corresponding task's numbers. We select the budget range of 5 with the corresponding budget of 7.035 to evaluate the budget distribution strategies. The share of budget that each level gets and the spare budget is also given in Table III. For instance, after scheduling of all tasks in the first level using the Height Proportional strategy, the spare budget of 0.15 (indicated in red) is added to assigned budget of next level (shown in blue) resulting in the total budget of 1.69 for that level (the budget trickling concept). The last row in Table III shows the makespan that was achieved by each strategy.

It is also interesting to look at which instance types and how many of each were provisioned by each strategy, this is given in Table IV. The most significant observation is that the "All in" strategy allocates a small number of powerful instances, reducing the data transfer costs and achieving the lowest makespan as given in Table III. From users' perspective, finding a schedule with a lower makespan for a given budget is the main concern, while from the providers perspective, maximizing utilization is the main concern. Very few research papers report the total number of instances that a workflow needs when provisioning. Although all BDT strategies meet the budget (only possible due to trickle down) and find a schedule, the type and the amount of requested VMs are very different as shown in Table IV. Launching too many instances does not lower makespan. Instead, it causes high scheduling overhead and low instance utilization. Therefore, the budget distribution strategy has a direct impact on resource utilization - which is significantly important.

The makespan and success rate of LIGO for four specified

TABLE III: Example computed budget distribution for each strategy over each level of a LIGO for budget range=5 and budget=7.035

| | | Budget Distribution Strategy | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Uniform | | Height | | Width | | Area | | ”All in” | |
| | Task Numbers | sub Budget | Spare Budget | sub Budget | Spare Budget | sub Budget | Spare Budget | sub Budget | Spare Budget | sub Budget | Spare Budget |
| Level 6 | 229 | 1.173 | 0.01 | 2.01 | 0.15 | 1.61 | 0.01 | 2.85 | 0.01 | 7.035 | 0.001 |
| Level 5 | 229 | 1.173 | 0.03 | 1.675 | 0.01 | 1.61 | 0.03 | 2.11 | 0.03 | 0.001 | 0.001 |
| Level 4 | 21 | 1.173 | 0.05 | 1.34 | 0.03 | 0.14 | 0.006 | 0.15 | 0.01 | 0.001 | 0.001 |
| Level 3 | 250 | 1.173 | 0.01 | 1 | 0.02 | 1.75 | 0.01 | 1.39 | 0.02 | 0.001 | 0.001 |
| Level 2 | 250 | 1.173 | 0.03 | 0.67 | 0.009 | 1.75 | 0.02 | 0.51 | 0.06 | 0.001 | 0.001 |
| Level 1 | 21 | 1.173 | 0.05 | 0.335 | 0.02 | 0.14 | 0.02 | 0.003 | 0.05 | 0.001 | 0.001 |
| Achieved Makespan | | 938.97 | | 798.71 | | 798.61 | | 682.6 | | 603.93 | |

TABLE IV: VM requested types by different strategies based on Table III

| Budget Distribution Strategy | | | | | |
|---|---|---|---|---|---|
| Vm Type | Uniform | Height | Width | Area | “All in” |
| m3.medium | 1 | 1 | 6 | 2 | 0 |
| c4.large | 6 | 3 | 2 | 2 | 1 |
| c3.xlarge | 6 | 3 | 2 | 3 | 1 |
| m4.2xlarge | 0 | 2 | 2 | 2 | 0 |
| c4.4xlarge | 6 | 2 | 2 | 2 | 0 |
| c3.8xlarge | 0 | 2 | 2 | 2 | 4 |
| #VMs | 19 | 13 | 16 | 13 | 6 |
| Total Cost | 6.985 | 7.006 | 7.026 | 6.968 | 7.035 |

lease times are shown in Fig 4. The “All in” strategy again performs significantly better than other strategies for each of the defined budget factors from low to high values. The random strategy has the worst performance for both makespan and success rate. Another observation is that the general trend of all strategies for different instance lease times is largely similar.

### *B. Other workflows*

The makespan and success rates of other workflows are presented in Fig. 5. The general trend is that by increasing the budget, a scheduler can launch more costly services, which in turn leads to a lower makespan.

In terms of makespan, for almost all workflows, the “All in” strategy has the best performance including lowest makespan and highest success rates. An interesting observation is that the structure and type of workflow appear to have a significant effect on success rate. For instance in CYBERSHAKE, the success rates of most of the strategies are generally poor, this is likely due to it being a data intensive workflow - the “All in” strategy of few, high power instances miminised the cost of data movement and produced the best makespan and success rate.

Overall, a budget bias towards the early tasks (and levels) of a workflow appears to produce better overall performance.

## V. Related Work

Scheduling in cloud environment encounters some challenges not present in traditional heterogeneous environments such as the Grid or HPC clusters. The cost model and resource provisioning in cloud are among the main challenging differences. For instance, pricing schemes in cloud systems are based on lease intervals from the time of provisioning, even if the instance is only used for a fraction of that period. A significant number of cost aware workflow scheduling algorithms in Grid have been proposed such as [4], [12]. However, cost in Grid is calculated based on the accumulated cost of requested services [13]. Moreover, workflow scheduling in grid focuses on minimizing the makespan without considering the cost [14].

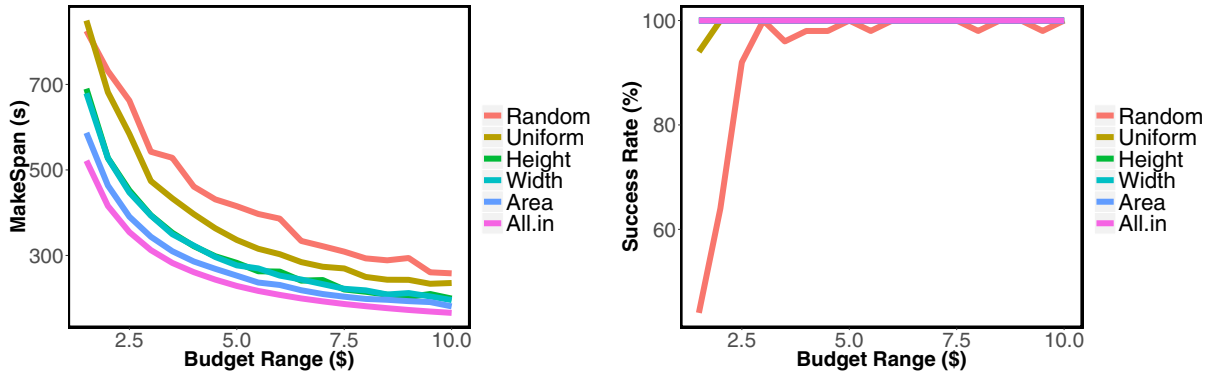
The most similar (although not very) related work to BDT appears in [3] and [15] whereby Zheng et al. proposed Budget constrained Heterogeneous Earliest Finish Time (BHEFT) which is an extension of HEFT algorithm [14]. In BHEFT, a current task budget (CTB) factor is introduced to distribute spare budget among unscheduled tasks. Their budget distribution is different from ours as the task budget and spare budget are calculated task by task. Moreover, their work is set within the context of a Grid environment which is not directly applicable to cloud environments due to differences in the cost model.

In cloud environments, most of the research focuses 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 [5], [6], [16], [17]. In this paper, we are focusing on budget constrained algorithms for cloud environments.

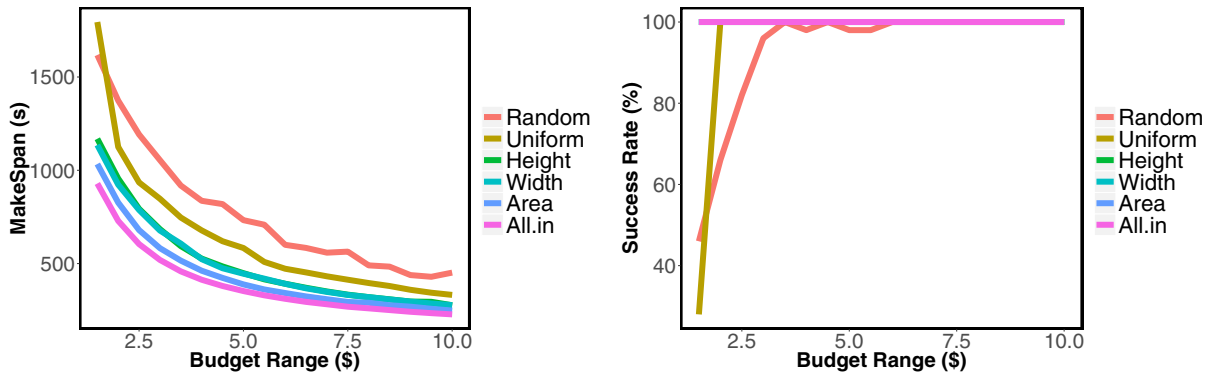
In [18], Zeng et al. presented a budget-aware backtracking algorithm for executing large scale many task workflows, referred to as ScaleStar. The authors in [19] and [20] presented an algorithm with budget constraints called minimum end-to-end delay under cost constraint (MED-CC). Firstly, each task 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 cost model considered in [18]–[20] is based on the use of fractional resources. For instance, in [20] a base processing unit which is ten instructions per time unit is set with price at 0.01 per time unit. However, in most of the cloud providers, like Amazon EC2, users are billed based on a longer interval like one hour.

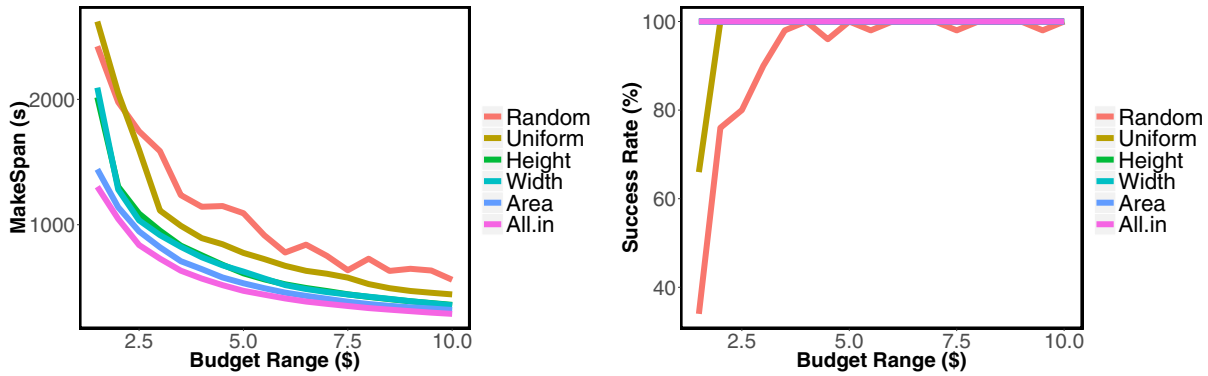
In [21], Li et al. presented an extension of the HEFT [14] algorithm called the Cost Conscious Scheduling Heuristic



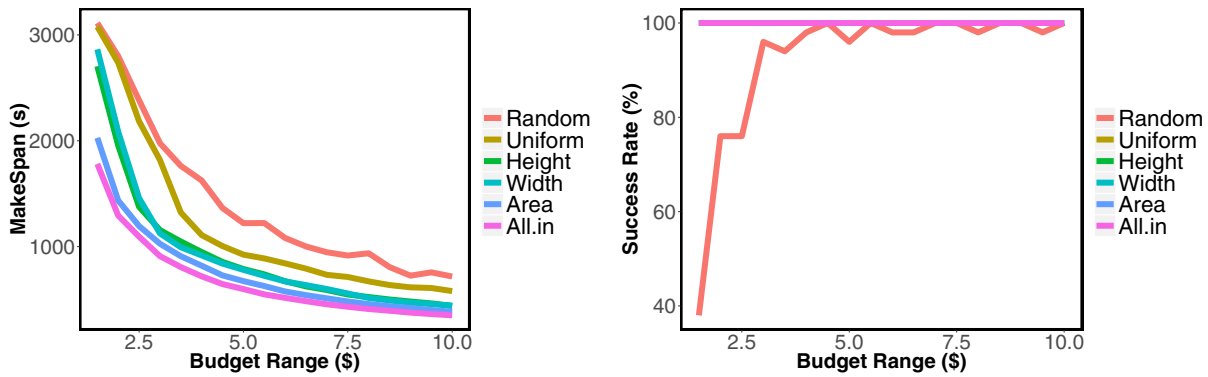

(a) Lease Time=15




(b) Lease Time=30

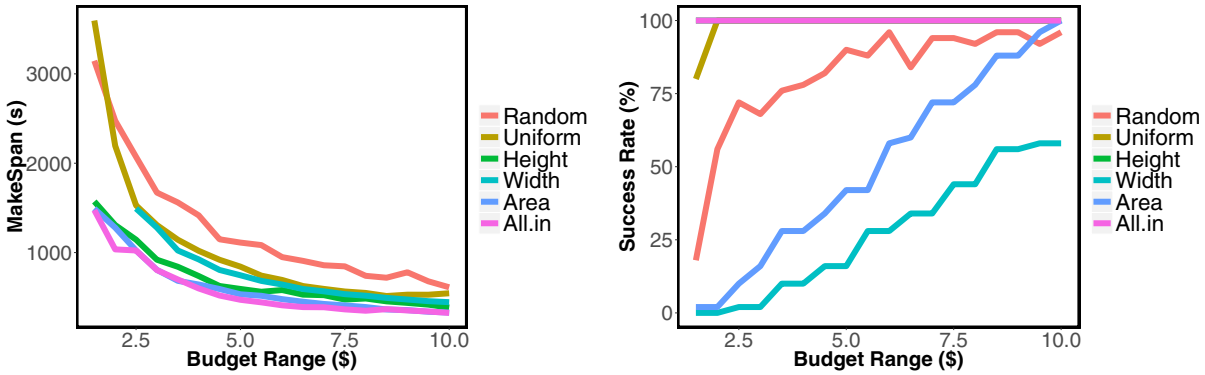



(c) Lease Time=45

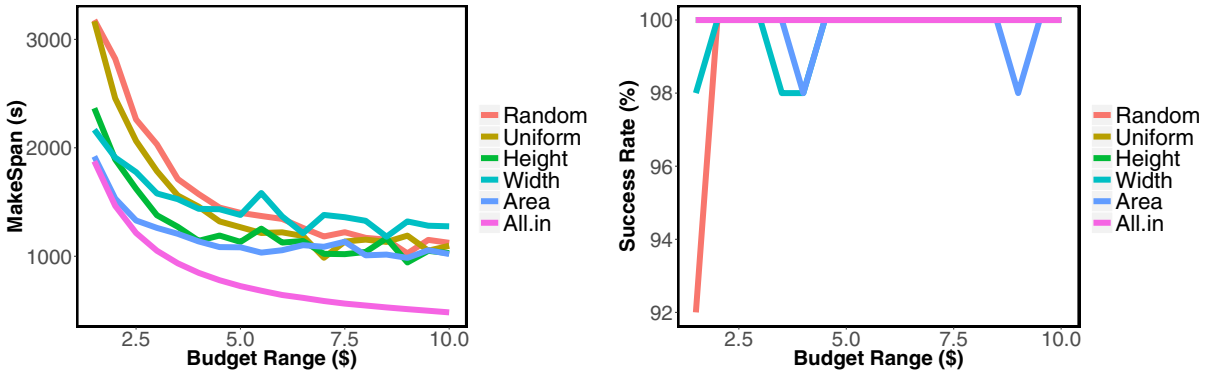



(d) Lease Time=60

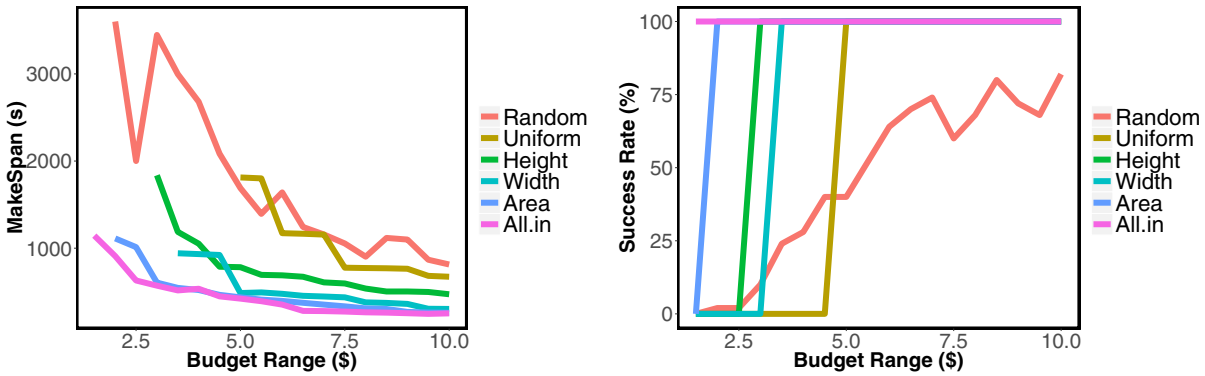
Fig. 4: Makespan and Success rate performance executing LIGO for all strategies for lease time of 15, 30, 45 and 60



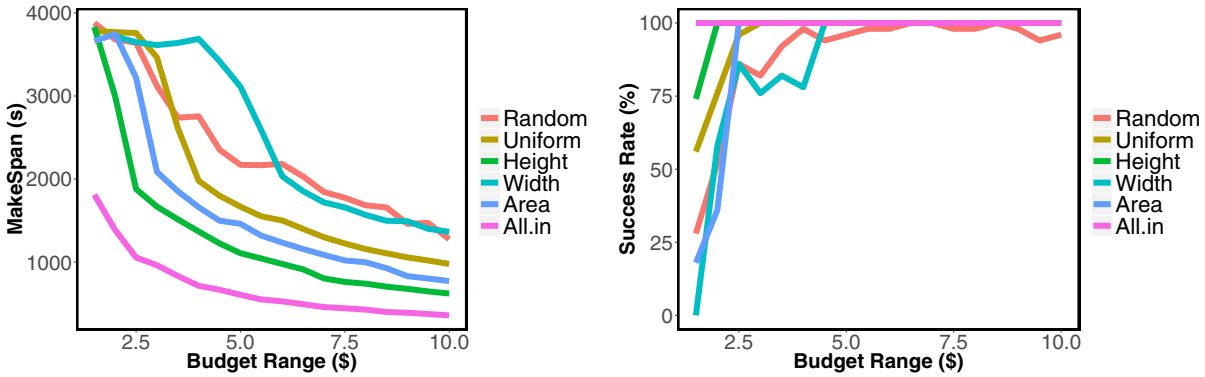

(a) CYBERSHAKE




(b) EPIGENOMICS




(c) MONTAGE




(d) SIPHT

Fig. 5: Makespan and Success rate performance executing of workflows for all strategies for lease time 60

(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.

In [22] two auto scaling techniques to solve the budget constrained scheduling for a workload consisting multiple workflows were proposed. In this work, budget is distributed to different workflows proportionally based on assigned priorities.

Scheduling Bags of tasks under budget constraints in cloud is presented in [23]. One of the assumptions considered by authors is tasks are preemptive which mean they can be interrupted, delayed and then retriggered sometime later. Our model is different from [23] in such a way, we have non-preemptive dependent tasks in our workflow model, a less forgiving constraint.

## VI. Conclusion

In this paper, we introduced the Budget Distribution with Trickling (BDT) algorithm that presents new notions for distributing budget based on the dependency structure inherent in workflows – levels. In addition we proposed several new strategies for sharing or distributing budget over these levels. We also proposed trickling to redistribute unspent budget down to other levels.

We evaluated the makespan and success rate of six strategies using five real-world workflows with different lease times on instances. The width and uniform strategies are largely analogous to existing budget distribution approaches and these are outperformed in makespan and success rate by Area and "All in" BDT strategies. Although, width and uniform also benefit from trickling and the other innovations built into BDT. Overall, the strategy that performed the best for all workflows in terms of both success rate and makespan was "All in". This strategy was proposed largely to test the hypothesis that biasing the budget distribution to the earliest levels was beneficial in terms of reducing makespan. "All in" took this to an extreme by assigning the entire budget to the first level and relying wholly on the trickle down mechanism to distribute budget to later levels. Counterintuitively, "All in" gave the best success rates - and the best performance for data intensive workflows due to the resulting data locality, from fewer more powerful instances.

The main finding of our research is in the importance of biasing budget distribution to early levels in a workflow. This leads to finding a schedule with a lower makespan. From this we have extrapolated two hypotheses that need further investigation. In addition, these results suggest that gaining a better understanding of workflow types (compute and data intensive) and workflow structure can lead to better budget distribution and likely scheduling in general.

## References


[1] I. J. Taylor, E. Deelman, D. Gannon, and M. Shields, Workflows for e-Science. Springer-Verlag London Limited, 2007.

[2] E. Deelman, "Grids and clouds: making workflow applications work in heterogeneous distributed environments," International Journal of High Performance Computing Applications, vol. 24, no. 3, pp. 284–298, 2010.

[3] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28675-9_8

[4] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, Integrated Research in GRID Computing: CoreGRID Integration Workshop 2005 (Selected Papers) November 28–30, Pisa, Italy. Boston, MA: Springer US, 2007, ch. Scheduling Workflows with Budget Constraints, pp. 189–202. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-47658-2_14

[5] 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.

[6] 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.

[7] 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.

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. 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, p. 2350, 2011.

[9] 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.

[10] 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.

[11] 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.

[12] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in 2006 Workshop on Workflows in Support of Large-Scale Science, June 2006, pp. 1–10.

[13] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in 2008 Grid Computing Environments Workshop, Nov 2008, pp. 1–10.

[14] 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.

[15] 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. [Online]. Available: http://dx.doi.org/10.1007/s10723-013-9257-4

[16] 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.

[17] 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.

[18] 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.

[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] 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.

[22] 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.

[23] 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.