Deadline 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*—**
Commercial clouds have become a viable platform for performing a significant range of large scale scientific analyses – due to the offerings of elasticity, specialist hardware, software infrastructure and pay-as-you-go cost model. Such clouds represent a low upfront capital cost alternative to the use of dedicated eScience infrastructure. However, there are still significant technical hurdles associated with obtaining the best performance for the cost - it is easy to provision commercial clouds inefficiently resulting in great and potentially unanticipated expense.

In this paper we introduce a new heuristic scheduling algorithm Deadline Distribution Ratio (DDR) to address the workflow scheduling problem with the objectives of minimizing the cost of Cloud computing resources while satisfying a given deadline. Within this context, we also investigate a range of different deadline distribution strategies and their effect on the overall scheduling performance. We then compare the DDR algorithm against three other published algorithms, using five different scientific workflows generated using the pegasus workflow generator, on a CloudSim simulation that implements a pricing model based on AWS. In general, the DDR algorithm returns the lowest costs across the majority of deadlines and workflows, while maintaining a high scheduling success rate.

1 Introduction

Basic research and consequently, scientific discovery, are in the midst of a disruptive transformation, as traditional experimental and observational approaches are enhanced and even supplanted by computational and data-intensive approaches. Researchers in almost every field are now faced with new opportunities and challenges that impact every stage of the research lifecycle due to the exponential growth of data volumes, data heterogeneity, and exploding analytical complexity. Unfortunately small to medium research laboratories and groups often lack the human and financial resources [1] to acquire and operate such large scale scientific discovery intrastructures.

This has resulted in a migration of science workflows [2], [3] from dedicated infrastructure - to commercial clouds [4], using execution engines such as Globus [5], and more recently paradigm shifts such as that envisioned by the recently proposed discovery cloud [1]. Such schemes require considerable software infrastructure to meet the needs of the science community, and such software needs to operate efficiently and cost effectively. One critical element is the provisioning of pay-per-use instances, and the subsequent scheduling of workflow tasks over them – in essence we need to complete execution on time and within budget. Scientific workflows vary in size from a couple of tasks to thousands, or million of tasks and these need to be scheduled in parallel and dependency order over, potentially, many instances. This is a workflow scheduling problem - and is inherently *NP*-complete.

There are two significant phases to solving such a workflow scheduling problem: selecting the task to be scheduled, and selecting the instance to be provisioned. The choices made in these phases naturally have a significant impact on the overall cost of the resulting schedule and if it can meet its overall deadline. One approach is to divide or distribute the deadline over the workflow as sub-deadlines to ensure a more manageable constraint satisfaction problem, and then to provision the instances to meet these sub-deadlines [6], [7]. This leaves us with two fundamental questions:

- What are the different possible ways to distribute a deadline over a workflow?
- How do these different strategies effect cost?

To answer these questions in this paper, we explore a range of strategies for distributing a deadline over a workflow and evaluate these using a cloudSim simulation. We find that the choice of distribution strategy has a significant impact on performance. We then incorporate these findings as the basis for a new heuristic scheduling and provisioning algorithm, Deadline Distribution Ratio (DDR). We then compare the cost and success rate performance of DDR against three previously published algorithms JIT [8], ICPCP [9] and our prior work PDC [6]. The CloudSim simulation used in the evaluations implements an AWS pricing model and use five different scientific workflows generated using the Pegasus workflow generator.

Our paper is organized as follows: Section 2 gives an overview of existing approaches to scheduling workflows. In Section 3, we define the workflow scheduling problem and describe our system model. In Section 4, we present our workflow scheduling algorithm. In Section 5 and 6, we outline our CloudSim-based simulation followed by results and performance evaluation. Finally, we summarize our work in Section 7.

2 Related Work

Allocating workflow tasks to resources can be separated into two stages, the first being scheduling and the second is provisioning [10]. Given a set of resources, the workflow task scheduling phase aims to determine the optimal execution order and task placement with respect to user and workflow constraints [11], [12]. The resource provisioning phase aims

to determine the number and type of resources required and then to reserve these resources for workflow execution [13], [14].

As there are several recent and comprehesive surveys [2], [3] on workflow scheduling, we need not duplicate the authors efforts. Therefore, for brevity, we only include a review of directly related deadline constrained workflow schedulers designed for a pay-per-use cloud environment.

One of the main strategies in the literature in deadline constrained workflow scheduling is distributing deadline among tasks [9], [15]–[17]. The distribution phase usually consists of two steps, workflow partitioning and deadline assignment. In the workflow partitioning, tasks can be considered as independent tasks in levels versus dependent tasks into different paths.

Deadline Bottom Level (DBL) [15] and Deadline Top Level (DTL) [16] are the most popular deadline distribution heuristics. DBL categorizes tasks in bottom-top direction while DTL partition tasks in the opposite direction, top-bottom. In the DBL heuristic, tasks are grouped in different levels where there are no dependencies between tasks in each level. However, tasks in DTL are categorized into paths as synchronization task or a simple task. A synchronization task is defined as a task which has more than one parent or child [16].

In the deadline assignment step, the overall deadline is divided and distributed in proportion to the minimum execution time of each level. However, in DBL, first the primary estimation on fastest instances is calculated. Then, the difference between the user-defined deadline and the primary estimation is distributed uniformly among all levels.

In Deadline Early Time (DET) [17], tasks are partitioned into two types: critical and non-critical activities. All tasks on the critical path are scheduled using dynamic programming under a given deadline. Non-critical tasks are backfilled between critical tasks. However, the communication time between tasks in a workflow is not taken into account by the DET scheduler.

In [18], we introduced the Proportional Deadline Constrained (PDC) algorithm for scheduling e-Science workflows on commercial clouds. The PDC distributes the deadline proportionally based on the task execution time over the levels.

Latest Finish Time (LFT) is also used to distribute deadline to individual tasks in several algorithms [8], [9]. LFT is the latest time at which a task can finish its execution such that the whole workflow can finish before the user defined deadline.

Infrastructure as a service (Iaas) Cloud Partial Critical Paths (IC-PCP) [9] categorizes tasks in partial critical paths (PCP). In the next step, the deadline is distributed to defined paths. However, after execution of each PCP, the value of LFT needs to be recalculated.

The Just in Time (JIT) algorithm proposed by Sahni and Vidyarthi in [8] is a dynamic cost minimization deadline constrained algorithm. The JIT algorithm attempts to combine pipeline tasks into a single task that can abrogate the data transfer time between co-located tasks. The majority of algorithms prioritize tasks to find the best candidate for execution however, no such policy is used in JIT.

3 Workflow and System Models

A Directed Acyclic Graph (DAG) is the most common representation of a workflow [19]. 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. Task t_i cannot start until all parents have completed.

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

$$TaskCost_{t_i}^{p_j} = \left\lceil \frac{w_{t_i}^{p_j}}{N_t} \right\rceil * c_j, \tag{1}$$

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:

$$Cost_o = \sum_{t_i \in G} TaskCost_{t_i}^{p_j}. \tag{2}$$

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.

4 The DDR algorithm

In this section we outline our new Deadline Distribution Ratio (DDR) algorithm. In addition to the two phases of task selection and instance selection, when we perform deadline distribution, we also need to introduce two additional phases, giving:

(A) Workflow partitioning: The workflow is partitioned into dependency free bags of tasks, called levels.
(B) Deadline Distribution: The user-defined deadline (T_D) is divided and distributed between levels. Each level gets its own level deadline. All tasks in the same level, have the same level-deadline.
(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 deadline.

Critically this section also introduces six base and 14 combined distribution strategies that are evaluated in this paper as part of the overall DDR algorithm.

4.1 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. Each level can therefore be thought of as a bag of tasks (BoT) containing a set of independent tasks. To allocate all tasks into different levels we categorize tasks in bottom-top direction. 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. 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:

$$N_L(t_i) = \max_{t_j \in succ(t_i)} \{N_L(t_j) + 1\}, \quad (3)$$

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.

$$\mathrm{TLS}(\ell) = \{t_i | N_L(t_i) = \ell\}, \quad (4)$$

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

4.2 Deadline Distribution

4.2.1 Initial Estimation: The initial estimated deadline for each level (ℓ) is calculated as:

$$InitialT_{sd}(\ell) = \max_{t_i \in \mathrm{TLS}(\ell)} \{\mathrm{ECT}(t_i)\}, \quad (5)$$

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

$$\mathrm{ECT}(t_i) = \max_{\ell \in pred(t_i)} \{InitialT_{sd}(\ell), EST(t_i)\} + w_{t_i}, \quad (6)$$

where $EST(t_i)$ is defined in equation. 9, $pred(t_i)$ denotes the set of predecessors of task t_i; w_{t_i} denotes the minimum execution duration for task t_i and ℓ indicates the parent level t_i. The task, t_{entry} has no predecessors, its ECT is equal to zero. In equation 5, the maximum ECT of all tasks in a level is used as the overall estimate for that level. This duration is effectively the absolute minimum time that is required for all tasks in a level to complete execution in parallel.

4.2.2 Deadline Distribution Strategies: The main idea of deadline distribution is simple as distribute deadline among different levels and try to complete its execution before any assigned sub-deadline such that the global deadline can be met.

The baseline deadline distribution strategies are:

- Random (R): The deadline is allocated randomly over the levels in the workflow.
- Uniform (U): Each level gets a $1/L$ share of the deadline, where L is the total number of levels.
- Height Proportional (H): Each level gets a share of the user deadline proportional to its distance from the entry node.
- Width Proportional (W): Each level gets a share of the user deadline proportional to the number of tasks within that level.
- Area Proportional (A): Combines width and height strategies to set the deadline for each level.
- Length Proportional (L): Each level gets a share of the deadline non-uniformly based on the proportion of its length. levels with longer tasks gain a larger share of the user deadline.

We now present an example to show how the deadline is distributed among different levels. Random needs no further explanation, so the example will only detail the rest of baseline 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 3. 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 deadline of 165 is assumed. This Number is arbitrary and serves as an example.

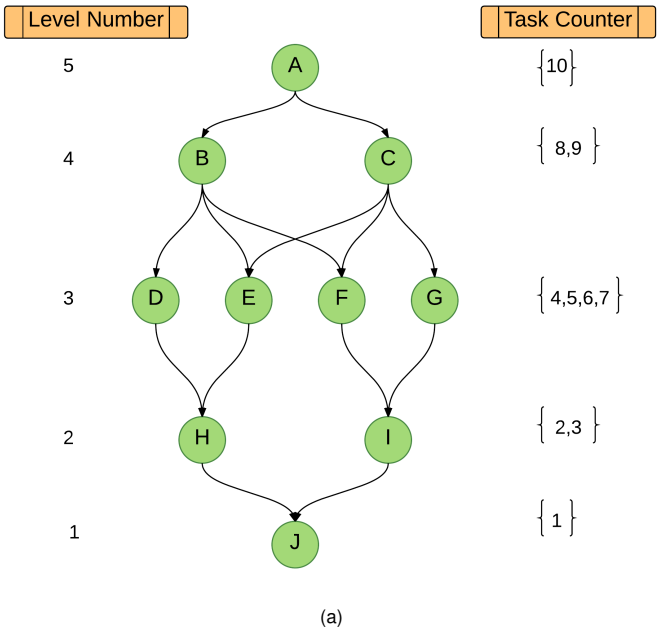


Fig. 1: A Sample Workflow with 10 tasks.

Each strategy distributes the user deadline based on the following basis (see Table I for the complete set of deadline shares and assigned sub-deadline to each level):

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

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

The Deadline Factor (DF) is calculated by:

$$DF = \frac{deadline}{L_{weight}} = \frac{165}{15} = 11.$$

For instance, level 4 consisting task B and C are assigned a share of the deadline equal to $4 \times DF = 4 \times 11 = 44$.

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

$$DF = \frac{deadline}{tasknumbers} = \frac{165}{10} = 16.5.$$

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

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

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

The Deadline Factor (DF) is calculated by:

$$DF = \frac{deadline}{L_{weight}} = \frac{165}{55} = 3$$

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

- Length Proportional: After calculating the estimated deadline value for all levels, we distribute the user deadline among all tasks non-uniformly based on a deadline proportion denoted by $\propto_{deadline}$ in equation 7:

$$\propto_{deadline} = \frac{T_D - InitialT_{sd}(1)}{InitialT_{sd}(1)}, \tag{7}$$

where $InitialT_{sd}(1)$ is the level that contains the exit task.

We then compute length of each level deadline as a function of this deadline proportion to each level as follows:

$$T_{sd}(\ell) = InitialT_{sd}(\ell) + (\propto_{deadline} \times |InitialT_{sd}(\ell)|). \tag{8}$$

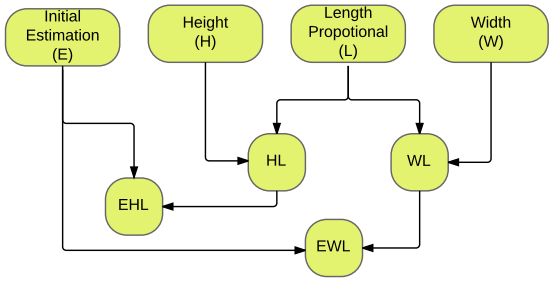
Intuitively, the levels with longer executing tasks gain a larger share of the user deadline.

In Table I, two rows are specified for each level. The first row indicates the share of deadline that each level gets. The second row is the final value of assigned sub-deadline to each level that is calculated based on the cumulative value with previous levels. Clearly, the sub-deadline value for level 1 should be equal to the total deadline. For instance, in Height strategy in level 4, the share of deadline is 44 (indicated in red) and the assigned sub-deadline is 99 (shown in blue). The Length Proportional strategy is not compared in Table I because it involves the execution time for each task that is not part of this example.

We also combine the baseline strategies to produce different variant strategies. For example, combination of the three strategies, Initial Estimation(E), Width(W) and Length Proportional (L) gives us a new strategy to distribute deadline which is named EWL. In this strategy, first, estimated deadline for all levels are calculated. Then, the leftover deadline is distributed based on the combination of Width and Length strategies. This gives a total of 14 different strategies that are evaluated in our experiments. Some of the generated strategies are shown as exemplars in Figure 2.

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

Deadline Distribution Strategy					
		Uniform	Height	Width	Area
Level 5	share of deadline	$\frac{165}{5} = 33$	$5{\times}DF = 55$	$1{\times}DF = 16.5$	$10{\times}DF = 30$
	sub-deadline	33	55	16.5	30
Level 4	share of deadline	$\frac{165}{5} = 33$	$4{\times}DF = 44$	$2{\times}DF = 33$	$17{\times}DF = 51$
	sub-deadline	66	99	49.5	81
Level 3	share of deadline	$\frac{165}{5} = 33$	$3{\times}DF = 33$	$4{\times}DF = 66$	$22{\times}DF = 66$
	sub-deadline	99	132	115.5	147
Level 2	share of deadline	$\frac{165}{5} = 33$	$2{\times}DF = 22$	$2{\times}DF = 33$	$5{\times}DF = 15$
	sub-deadline	132	154	148.5	162
Level 1	share of deadline	$\frac{165}{5} = 33$	$1{\times}DF = 11$	$1{\times}DF = 16.5$	$1{\times}DF = 3$
	sub-deadline	165	165	165	165



(a)

Fig. 2: Producing different strategies.

4.3 Task Selection

In each step of our algorithm, those tasks which are ready to execute are put in the task ready list. A task is ready when all of its parents have been executed and all its required data has been provided. Therefore, there are no dependencies between tasks that are at the same level. In order to select a proper task for execution, all tasks in the ready list are prioritized with their Earliest Start Time (EST). The EST is the soonest possible time that a task can start its execution which depends on finish time of its parent. The Earliest Start Time (EST) of a task t_i is calculated on the instance with the shortest execution

time and defined as:

$$EST(i) = \begin{cases} 0 & , t_i = t_{entry} \\ \max\limits_{t_j \in pred(t_i)} \left\{ EST(t_j) + w_{t_j} + C_{i,j} \right\} & , \text{Otherwise,} \end{cases} \quad (9)$$

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 $C_{i,j}$). For each task the EST on all VMs is calculated. The task that starts first will be the best candidate for execution.

4.4 Instance Selection

In the Instance Selection phase we aim to identify the most suitable instance to execute tasks. The Instance selection decision for each task aims to minimize the total cost of workflow execution while also attempting to meet the tasks sub-deadline.

We introduce an objective function referred to as Instance Comparative Ratio (ICR).

$$ICR\ {}^{p_j}_{t_i} = \frac{Level^{\ell_{t_i}}_{deadline} - \mathrm{ECT}(t_i, p_j)}{TaskCost^{p_j}_{t_i}} \quad (10)$$

where $Level_{deadline}$ is the deadline that is assigned to the level which contains the task t_i. The time needed for the current task, (t_i), to execute on the instance p_j is calculated by $\mathrm{ECT}(t_i)$. The ECT is the earliest time that a task can complete execution on an instance (as defined in equation 6). The value in the numerator of equation 10 assesses the differences between the sub-deadline and earliest completion time of the current task on the instance p_j. The denominator is the cost of current tasks which is defined in 1.

Most cloud providers like Amazon Web Services (AWS) Elastic Compute Cloud (EC2) charge users based on 60 minute intervals. When an instance is provisioned, the user is billed for the entire billing interval even if the task completes before the end of the interval. Therefore, if other tasks can execute on the same instance within the remaining interval, their execution cost can be considered zero. Thus, when allocating instances we prioritize selecting instances with remaining idle billing intervals. The first step of the algorithm explicitly considers instances that have no cost to execute the current task as well as ensuring that the earliest completion time does not exceed the level deadline. The instance with minimum ECT is then selected (the fastest one).

If no instances can be found in the previous step, our algorithm provisions a new instance based on the highest ICR value. In tight deadlines, there is a possibility that cheaper instances cant meet the task level's sub-deadline. Therefore, the value of ICR is negative as its numerator is negative. If this condition is met, more expensive instances are candidates for the current task. In fact, the ICR value is trying to adjust the cost and time for current task among all instances. For cost minimization purpose, most of the proposed algorithms like [20] try to schedule a task on a cheapest available instance (slower) while still meeting its assigned sub-deadline. However, with this strategy, tasks can take much longer time if the resources are slower and leads to some delay in the EST of its children. To avoid this, the key concept of introducing ICR in 10 is to make a trade-off between time and cost.

5 Evaluation

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 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 and 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, based on the average bandwidth published by AWS [21]. The processing capacity of an EC2 unit is estimated at one Million Floating Point Operations Per Second (MFLOPS) [22]. 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 [23]. Therefore, in our simulation, we adopted a 97-second boot time based on previous measurements of EC2 [23].

We use five common scientific workflows: Cybershake, Epigenomics, Montage, LIGO and SIPHT, to evaluate the performance of our algorithms with a realistic load. The characteristics and task composition of these workflows have been analyzed in published works [24], [25]. We vary the deadlines 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:

$$FS = \sum_{t_i \in CP} (w^j_i) \quad (11)$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j. A Critical Path (CP) is the longest path from the entry to exit node of a task. If all tasks on the CP of a workflow are executed on the fastest instance type, the fastest schedule will be reached.

We define the deadline as a function of the fastest schedule and this deadline is expressed in equation 12 in which the deadline varies from tight to moderate to relaxed:

$$\text{deadline} = \alpha * FS, \quad 0 < \alpha < 20. \tag{12}$$

The deadline factor α starts from 1 to consider very tight deadlines (typically approaches the fastest schedule) and is increased by one up to a value of 20, which results in a very relaxed deadline.

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 [24] to create representative workflows with the same structure as five real world scientific workflows (Cybershake, Epigenomics, Montage, LIGO and SIPHT). For each workflow structure, and each deadline factor, 100 distinct Pegasus generated workflows were scheduled in CloudSIM and the performance of the scheduling algorithms are detailed in the following section.

6 Experimental Results

In this section, we first compare the performance of 14 deadline distribution strategies in our algorithm. Then, the evaluation of the presented algorithm with other state-of-the-art algorithms [8], [9], [18] is presented in 6.2. The main metrics evaluated in our comparison are the cost and success rate (SR).

To compare the monetary cost between the algorithms, we consider the cost of failure in meeting a deadline. For this purpose, a weight is assigned to average cost returned by each algorithm. Let k denote the set of a simulation runs that successfully meets the scheduling deadline, thus the weighted cost is calculated as:

$$Cost_w = \frac{\sum_k Cost_o\,(k)}{SR}, \tag{13}$$

where $Cost_o\,(k)$ is the cost for the experiments that meet the deadline and SR denotes the success rate.

6.1 Cost comparison for distribution strategies

To observe the precise behavior of each strategy, for each dataset, we selected ranges for deadline from 5 to 10. This range was chosen that gives us more detail about the performance of all strategies to simplify interpretation of results. Table III lists the legends notation in Fig. 3. The first observation is that different strategies have unstable trends in tested datasets. For example, the Width (W) strategy has the worst performance in CYBERSHAKE while gaining the almost lowest cost in MONTAGE. Similarly, the EL strategy resembles the same trend in MONTAGE and SIPHT. This is attributed to the fact that workflows differ remarkably in their characteristics including structure, size, computation and communication requirements.

Algorithm Name	Description
R	Random
U	Uniform
H	Height
W	Width
A	Area
L	Length
EU	Estimation&Uniform
EH	Estimation&Height
EW	Estimation&Width
EA	Estimation&Area
EL	Estimation&Length
EHL	Estimation&Height&Length
EWL	Estimation&Width&Length
EAL	Estimation&Area&Length

TABLE III: Explanation of legends in Fig. 3

Workflows consists of various components including process, pipeline, data distribution, data aggregation and data redistribution [24]. The behavior of the EWL strategy indicates a good performance with different datasets. This strategy consider the number of tasks in a level and length of a level, simultaneously. In the next section, we consider this strategy to distribute the deadline in our algorithm.

While cost differences may seen negligible between some of the strategies, for executing big datasets, the differencing could be significant. This shows that the deadline distribution strategy could play a key role in minimizing the cost.

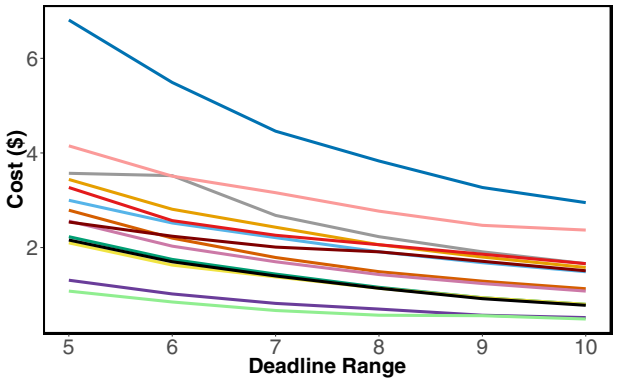
6.2 Cost Comparison with other algorithms

Three state-of-the-art algorithms, IC-PCP [9], JIT [8] and PDC [18] were selected in order to compare with our DDR algorithm. The cost and success rates of five scientific workflows in different deadline intervals are presented in Fig. 4.

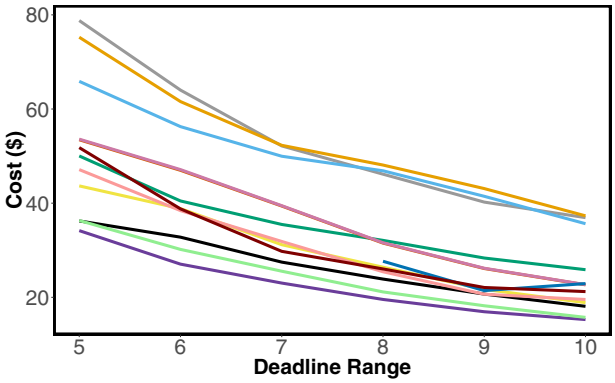
The general results show that the JIT underperforms others in terms of cost in all cases. For almost all workflows, the DDR algorithm has the best performance including lowest cost and highest success rate. There are instances where their performance is poorer, but these are few and far between. For example, in CYBERSHAKE and EPIGENOMICS with very tight deadlines, our algorithm is unable to schedule some workflows.

The IC-PCP algorithm has the worst success rate specially in tight deadlines. The relaxing of the deadline should lead to increase the success rate of each algorithm. However, the behavior of IC-PCP in different intervals is contrary to expectations. The highest failure occurs in EPIGENOMICS while even in relaxed deadline, IC-PCP can find a schedule for less than 25% before its deadline is reached. JIT performs very well in most of the deadline intervals with nearly 100% success rate. However, its the most expensive algorithm to find

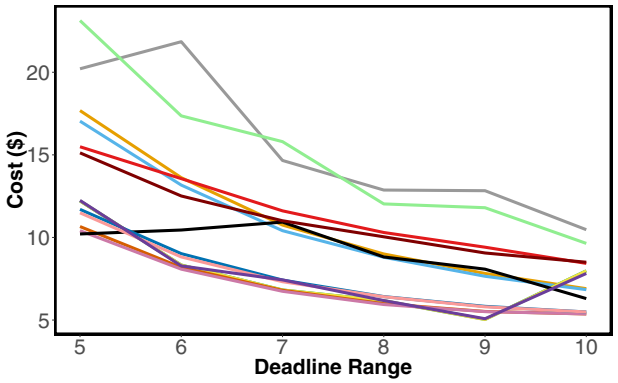
(a) Algorithm Name



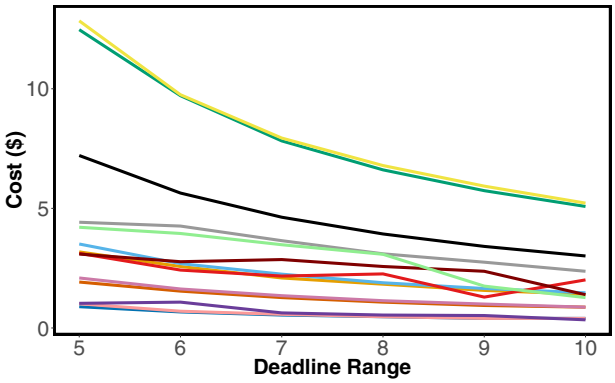
(b) CYBERSHAKE



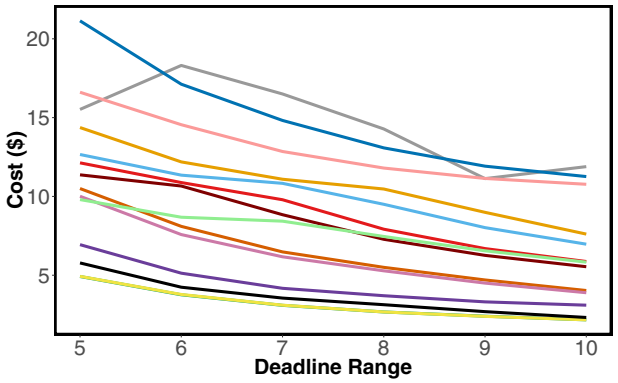
(c) EPIGENOMICS



(d) LIGO



(e) MONTAGE



(f) SIPHT

Fig. 3: Cost vs. deadline for different deadline distribution strategies

a schedule over all workflows and instance configurations. We attribute that the main reason for this behavior in JIT is how instances are selected [8].

7 Conclusion

In this paper we have presented the Deadline Distribution Ratio (DDR) algorithm for scheduling workflows in dynamically provisioned commercial cloud environments. Our approach focuses on addressing different ways of deadline distribution in scientific workflow. For that purpose, we introduced new strategies for deadline distribution assessed the effectiveness of these strategies in terms of cost and success rate. Some strategies exhibit performance that is strongly dependent on the workflow size and structure including process, pipeline, data distribution, data aggregation and data redistribution. In general, the strategy which takes into consideration the execution time of each level as well as number of tasks in the level, yields the lowest cost.

References

[1] 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.

[2] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," The Journal of Supercomputing, pp. 1–46, 2015.

[3] 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.

[4] D. Lifka, I. Foster, S. Mehringer, M. Parashar, P. Redfern, C. Stewart, and S. Tuecke, "XSEDE cloud survey report," Technical report, National Science Foundation, USA, Tech. Rep., 2013.

[5] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, and I. Foster, "The Globus Galaxies platform: delivering science gateways as a service," Concurrency and Computation: Practice and Experience, pp. n/a–n/a, 2015. [Online]. Available: http://dx.doi.org/10.1002/cpe.3486

[6] V. Arabnejad and K. Bubendorfer, "Cost effective and deadline constrained scientific workflow scheduling for commercial clouds," in proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA). Cambridge, MA USA: IEEE, September 2015.

[7] V. Arabnejad, K. Bubendorfer, B. Ng, and K. Chard, "A deadline constrained critical path heuristic for cost-effectively scheduling workflows," in proceedings of the 8th IEEE International Conference on Utility and Cloud Computing (UCC)). Limassol, Cyprus: IEEE, December 2015.

[8] J. Sahni and D. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," Cloud Computing, IEEE Transactions on, vol. PP, no. 99, pp. 1–1, 2015.

[9] 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, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[10] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," The Journal of Supercomputing, pp. 1–46, 2015.

[11] R. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," Parallel and Distributed Systems, IEEE Transactions on, vol. 25, no. 7, pp. 1787–1796, July 2014.

[12] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series. Springer-Verlag, 2007.

[13] K. Chard, K. Bubendorfer, and P. Komisarczuk, "High occupancy resource allocation for grid and cloud systems, a study with drive," in proceedings of the ACM International Symposium on High Performance Distributed Computing (HPDC), Chicago, Illinois, June 2010. [Online]. Available: publications/HPDC2010.pdf

[14] R. Chard, K. Chard, K. B. andLukasz Lacinski, R. Madduri, and I. Foster, "Cost-aware cloud provisioning," in the IEEE 11th International Conference on E-Science, August 2015.

[15] 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.

[16] 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.

[17] Y. Yuan, X. Li, Q. Wang, and X. Zhu, "Deadline division-based heuristic for cost optimization in workflow scheduling," Information Sciences, vol. 179, no. 15, pp. 2562–2575, 2009.

[18] V. Arabnejad and K. Bubendorfer, "Cost effective and deadline constrained scientific workflow scheduling for commercial clouds," in proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA). Cambridge, MA USA: IEEE, September 2015.

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

[20] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 22:1–22:11.

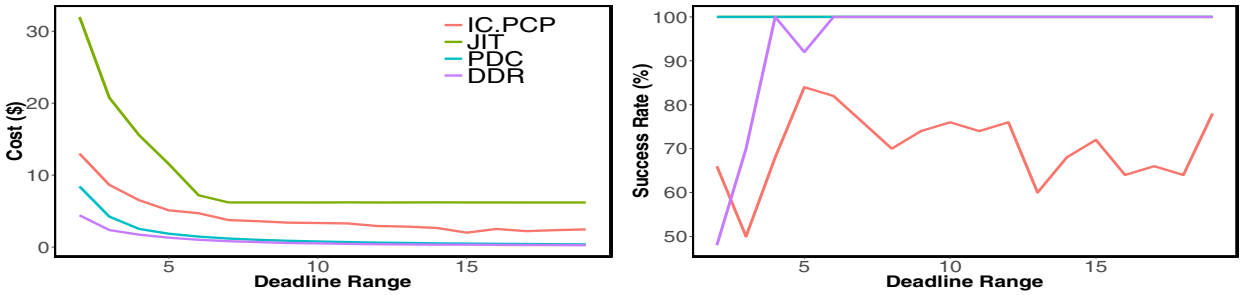
[21] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?" in Proceedings of the 2008 international workshop on Data-aware distributed computing. ACM, 2008, pp. 55–64.

[22] 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.

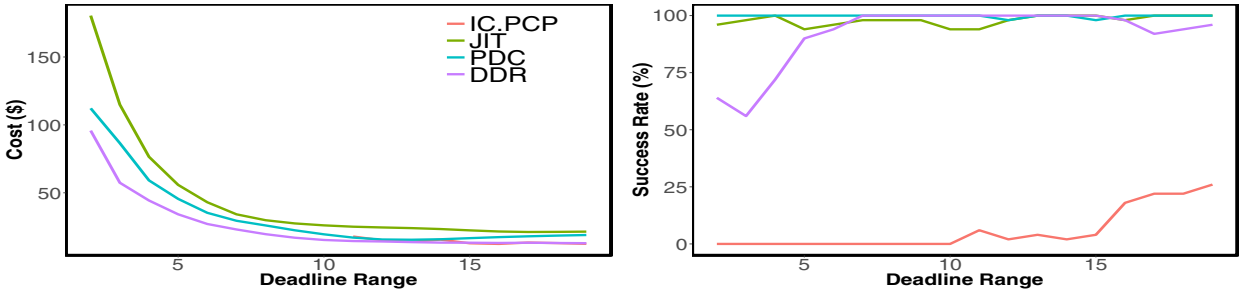
[23] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, ser. CLOUD '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 423–430.

[24] 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.

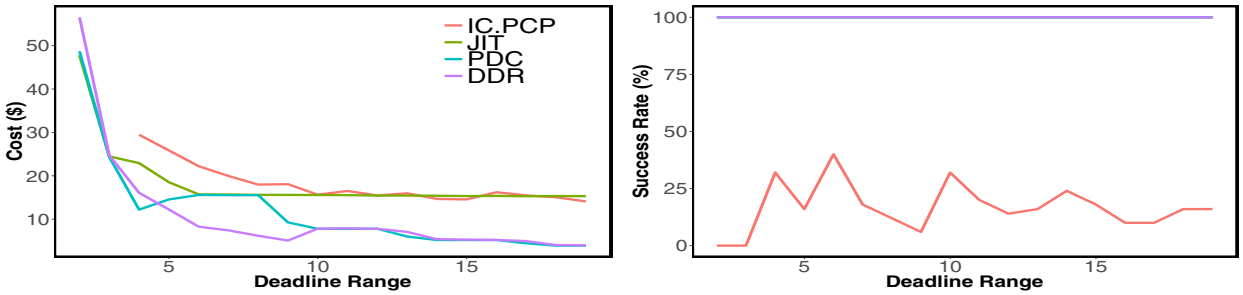
[25] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," Future Generation Computer Systems, vol. 29, no. 3, pp. 682 – 692, 2013, special Section: Recent Developments in High Performance Computing and Security.



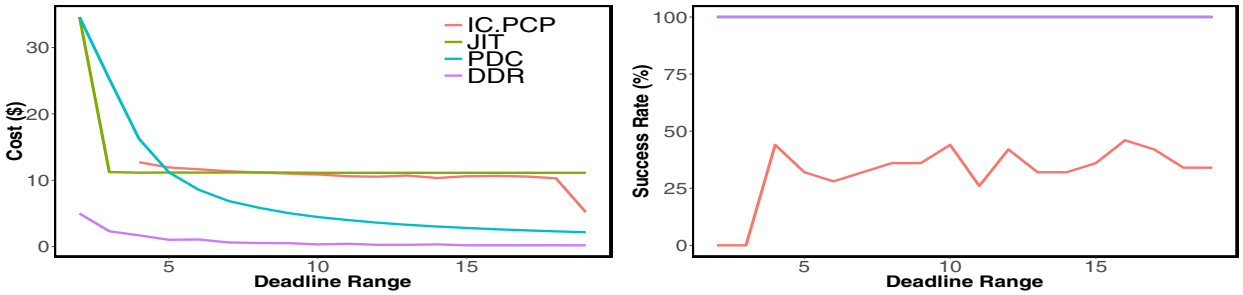
(a) CYBERSHAKE



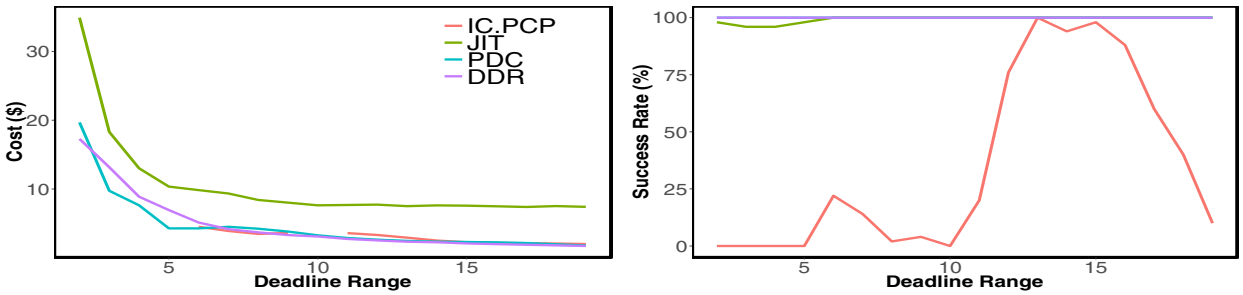
(b) EPIGENOMICS



(c) LIGO



(d) MONTAGE



(e) SIPHT

Fig. 4: Cost vs. deadline for five different datasets