2015 IEEE 14th International Symposium on Network Computing and Applications

Cost Effective and Deadline Constrained Scientific Workflow Scheduling for Commercial Clouds

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

***Abstract*—Commercial clouds have increasingly become a viable platform for hosting scientific analyses and computation due to their elasticity, recent introduction of specialist hardware, and pay-as-you-go cost model. This computing paradigm therefore presents a low capital and low barrier alternative to operating dedicated eScience infrastructure. Indeed, commercial clouds now enable universal access to capabilities previously available to only large well funded research groups. While the potential benefits of cloud computing are clear, there are still significant technical hurdles associated with obtaining the best execution efficiency whilst trading off cost. Large scale scientific analyses are typically represented as workflows, in order to manage multiple tools and data sets. Mapping workflow tasks on to a set of provisioned instances is an example of the general *scheduling problem* and is NP-complete. In this case, the mapping includes elasticity, where as part of the mapping process additional instances may be provisioned. In this paper we present a new algorithm, Proportional Deadline Constrained (PDC), that addresses eScience workflow scheduling in the cloud. PDC's aim is to minimize costs while meeting deadline constraints. To validate the PDC algorithm, we constructed a CloudSim testbed and compared PDC with two other similar algorithms over three workflows. Our results demonstrate that overall PDC achieves generally lower costs for a given deadline, but more significantly, is usually able to construct a viable schedule with tight deadlines where the other algorithms studied cannot.**

I. Introduction

Workflows are the most widely adopted tool for modelling and managing complex distributed scientific applications. Cloud resources are available in a range of capacities, configurations, localities, availability zones, and may include specialist hardware such as GPUs. Amazon EC2 users, for example, are charged based on the number of hours provisioned, instance type selected and whether the instance is: On-Demand, Reserved, Dedicated or a Spot instance.

Access to such pay-per-use commercial elastic computing capacity has resulted in a move towards the hosting of e-Science applications on commercial clouds – eighty cloud-based science gateways and workflow systems were identified in Lifka et al. [1]. In terms of larger scale e-Science, the Globus Galaxies platform [2] is used by more than 300 researchers across 30 institutions. However, the use of commercial clouds does not inevitably result in cost-effective or timely scientific computing.

When combined with the *NP*-complete task scheduling [3] - even small provisioning inefficiencies, including failure to meet workflow dependencies on time, exceeding charging intervals, and selecting the wrong instance type for a task, can result in significant monetary costs [4] [5].

To address these issues, we present the Proportional Deadline Constrained (PDC) algorithm for scheduling eScience workflows on commercial clouds. The PDC algorithm focuses on deadline constraints while minimizing costs. The PDC algorithm consists of four stages: (i) workflow levelling to extract the maximum inherent parallelism of workflow; (ii) proportional deadline distribution to appropriately partition the user deadline over the levels defined in the first stage; (iii) task selection to prioritise ready tasks; and (iv) instance selection to determine the best instance choice based on the execution estimates calculated in the distribution stage. This final step focuses on finding the best tradeoff of cost vs time, and includes both re-use of pre-provisioned instances and the creation of new instances on demand. We then compare the PDC algorithm in a CloudSim simulation with two other workflow scheduling algorithms, IC-PCP [6] and GAIN [7]. Overall, in comparison with the other algorithms, PDC is shown to achieve lower costs and increase the workflow task scheduling success rate leading to more viable workflow schedules.

The rest of this paper is structured as follows. In Section II we address related work, giving an overview of the approaches to scheduling workflows. We follow this with Section III defining the elements of workflow scheduling and then in Section IV we present the PDC algorithm. We outline our CloudSim simulation and the parameters of our experiments in Section V, and present our experimental results in Section VI. Finally we summarise our contributions and conclusions in Section VII.

II. Related Work

Task scheduling is an *NP*-complete problem, and many algorithms, including those based on heuristic, search-based, and meta-heuristic strategies have been proposed for efficient resource scheduling. In such systems tasks are either considered independent (bag of tasks) or dependent (workflows) [8], [9], [10], [11]. The task of allocating work to resources can be separated into two stages, the first being scheduling, the second provisioning. Algorithms such as GAIN [7] can be classified as pure scheduling, while systems such as DRIVE [12] focus on provisioning. The majority of Cloud scheduling systems necessarily include both scheduling and provisioning stages.

The two most significant classes of workflow scheduling are best effort and QoS constraint scheduling [13]. In best effort scheduling algorithms, minimizing the makespan is the common objective while ignoring other important factors such as cost. Simple heuristics such as Min-Min, Max-Min and Suffrage [14] are applied in the workflow scheduling problem to find the shortest makespan. The Min-Min algorithm calculates the Minimum Completion Time (MCT) for all resources for all tasks. The task that will be completed in the minimum of time is selected and assigned to the corresponding resource. The Max-Min algorithm is similar to Min-Min, the difference is that the task which is executed has the overall maximum completion time.

QoS constrained scheduling attempts to meet user defined requirements of which deadline and budget are the most common. Deadline is the maximum amount of time users need to wait to receive the result of the execution of their request. Budget is the amount of money the users wish to spend when using the resources. QoS-constrained workflow scheduling is closer to real-world scientific (and other) applications in contrast to best-effort scheduling [9].

Many guided random searches such as Genetic Algorithm(GA) [15][16], Ant Colony Optimization(ACO) [17] and Particle Swarm Optimization (PSO) [18][19][20] have been used to tackle the workflow scheduling problem with multiple constraints. Guided random searches usually produce acceptable answers in cloud environment. However, they are usually time-consuming algorithms based on their need for an initialization phase to reach the final answer.

In terms of budget or cost optimization, researchers have explored a number of approaches. In [7], two algorithms based on the local optimization in Grids, LOSS and GAIN, were proposed. These algorithms attempted to find a schedule to meet the user-specified budget. Both algorithms start with one of two different initial assignments:

- Best assignment: a time optimized assignment in which the execution time is the minimum possible. For example, the HEFT algorithm [21] is used as an initial assignment for the LOSS algorithm.
- Cheapest assignment: a cost-optimized assignment wherein all tasks are assigned to resources having the least execution cost [7]. For example, GAIN uses the cheapest assignment as the initial assignment.

Tasks are repeatedly selected for reassignment till the user-constrained budget is reached. In Bittencourt and Madeira [22] the authors presented HCOC, the Hybrid Cloud Optimized Cost scheduling algorithm, that uses a combination of resources from private and public clouds. The initial schedule starts to execute tasks on resources that belong to a private cloud – if the initial scheduling cannot meet the user deadline, additional resources are leased from a public cloud.

In Abrishami et al. [6] the authors proposed the Infrastructure as a service (Iaas) Cloud Partial Critical Paths (IC-PCP) algorithm which aims at minimizing the execution cost while meeting the user defined deadline in cloud. All the tasks in a partial critical path (PCP) are scheduled to same cheapest applicable instance that can complete them by the given deadline. This avoids incurring communication costs for each PCP. However, the IC-PCP algorithm did not consider the boot and deployment time of VMs. Calheiros et al. [23] proposed an Enhanced IC-PCP with Replication (EIPR) that uses the idle time of provisioned instances and budget surplus to replicate tasks. The experimental results showed the likelihood of deadline meeting is increased by using task replication. However, in EIPR task replication comes at an oportunity cost to the user - and in this paper we utilise the idle time to reduce overall costs, while still meeting the deadlines.

In [24], the authors present a mixed integer nonlinear programming problem to solve the scheduling of large scale scientific applications on hybrid clouds, where the optimization objective is the total cost, with a deadline constraint. Zhu et al. [25] present a two step heuristic, the High throughput Workflow scheduling Algorithm with Execution time bound (HiWAE). Their approach aims to reduce the workflow response time and energy consumption simultaneously.

In [26] the Partitioned Balanced Time Scheduling (PBTS) algorithm aims to minimize the cost of workflow execution while meeting a user deadline constraint. The PBTS algorithm estimates the minimum number of instances that require to be leased in order to minimize the execution cost.

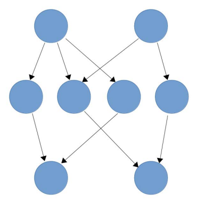
Malawski et al. [27] present three algorithms for scheduling set of workflows in Iaas cloud. Their algorithms aim to maximize the number of workflows that can be finished while meeting given budget and deadline constraints. However, in both [26][27], the authors consider only one type of VM rather than the wide variety of types that are currently available. In this paper we consider multiple instance types.

III. Problem Definition

A workflow is represented as a weighted Direct Acyclic Graph (DAG) $G = (T, E)$, where T is the set of tasks and E is the set of dependencies between tasks. $e_{i,j} \in E$ represents the precedence constraint as directed arcs between t_i and t_j, $t_i, t_j \in T$ which indicates that task t_j can start only after completing the execution of task t_i. Task t_i is named the predecessor or 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. In the DAG, a task without any parents is called an entry task, and a task without any children is called an exit task. It is possible to have more than one entry or exit task, as in Fig.1(a). In order to ensure the DAG has only one input and one output, two dummy tasks that have zero execution cost are added to the DAG (Fig.1(b)). The overall completion time of a whole workflow is called the schedule length or makespan. Scheduling is successful if the makespan is less than the user defined deadline.

The target cloud platform is composed of a set of instances with different characteristics. Accordingly, our environment is a heterogeneous system. A Cloud provider offers a set of computing units and storage services with different characteristics such as different CPU types, different memory size and


(a) A sample DAG with 8 tasks

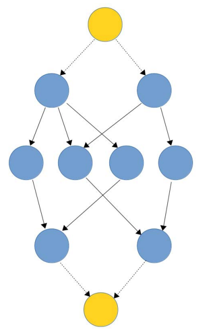

(b) A DAG with 2 dummy tasks

Fig. 1: Example workflow structure.

different prices.

IV. The PDC Algorithm

In this section we present our new algorithm, Proportional Deadline Constrained (PDC). The PDC algorithm produces a deadline constrained schedule that minimize the financial cost of execution and has a generally lower failure rate in constructing schedules for tighter deadlines. PDC consists of four parts:

1) Workflow Levelling: Each task is categorised in a level by analysing its synchronisation requirements to maximize the achievable parallelism from the workflow. In this paper, we used the DBL [28] algorithm to group tasks into different levels.
2) Deadline Distribution: A 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.
3) Task Selection: A task is selected based on its priority in the ready list for execution.
4) Instance Selection: The best instances are chosen to meet the deadlines at the minimum cost.

A. Workflow Levelling

All tasks are allocated into different levels to maximize the parallelism that can be extracted from the workflow while preserving any dependencies. Essentially workflow levelling extracts the embedded Bags of Tasks (BoT) from a workflow. There are no dependencies between tasks that are at the same level.

The level of task t_i is an integer value representing the maximum number of edges in the paths from task t_i to the exit task, see Fig. 1(b). The level number identifies which BoT a task belongs to. For the exit task, the level number is always 1, and for the other tasks, it is determined by:

$$\text{level–number}(t_i) = \max_{t_j \in \succ(t_i)} \{\text{level–number}(t_j) + 1\} \quad (1)$$

where $\succ(t_i)$ denotes the set of immediate successors of task t_i. All tasks are then grouped into the same Tasks Level Set (TLS) based on their level numbers.

$$\text{TLS}(\ell) = \{t_i | \text{level–number}(t_i) = \ell\} \quad (2)$$

where ℓ indicates the level number which is between $[1 \ldots \text{level–number}(t_{entry})]$.

B. Proportional Deadline Distribution

Once all tasks are assigned to their respective level, this phase proportionally distributes a share of the user deadline (T_D) across the identified levels. Each sub-deadline assigned to a level is termed the *level deadline* ($Level_{deadline}$). We need to ensure that every task in a level can complete its execution before any assigned level deadline. Firstly, the initial estimated deadline for each level (ℓ) is calculated by:

$$Level^{\ell}_{deadline} = \max_{t_i \in \text{TLS}(\ell)} \{\text{ECT}(t_i)\} \quad (3)$$

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

$$\text{ECT}(t_i) = \text{exec}_{min}(t_i) + \max_{t_k \in \prec(t_i)} \left\{ Level^{\ell_{t_k}}_{deadline} + \overline{c_{i,k}} \right\} \quad (4)$$

where $\prec(t_i)$ denotes the set of predecessors of task t_i, $\text{exec}_{min}(t_i)$ denotes the minimum execution time for task t_i, $\overline{c_{i,k}}$ indicates the average communication transfer time between task t_i and its parent (t_k) and ℓ_{t_k} indicates the level number of parent t_i. As the task, t_{entry} has no predecessors, its ECT is equal to zero. In equation 3, the maximum ECT of all tasks in a level is used as the overall estimate for that level. This is effectively the absolute minimum time that is required for all tasks in a level to complete their execution in parallel.

After calculating the estimation deadline value for all levels, we distribute all user deadline among all tasks non-uniformly based on the proportion of $Level^{\ell}_{deadline}$.

$$\propto_{deadline} = \frac{T_D - Level^{1}_{deadline}}{Level^{1}_{deadline}} \quad (5)$$

where $Level^{1}_{deadline}$ is the level that contains the exit task.

Then, add this proportion to each level based on the length of each level deadline:

$$Level^{\ell}_{deadline} = Level^{\ell}_{deadline} + (\propto_{deadline} * |Level^{\ell}_{deadline}|) \quad (6)$$

The key point is that the levels with longer tasks gain a larger share of the user deadline.

C. 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. These tasks then need to be prioritised for execution - for the prioritisation we used Downward Rank [21], which is widely used for task ranking. Task selection starts from the $entrynode$ and is computed recursively by traversing the DAG to the $exit\ node$.

$$\text{rank}_d(t_i) = \max_{t_k \in \prec(t_i)} (\overline{w_k} + \overline{c_{k,i}} + \text{rank}_d(t_k)) \tag{7}$$

where $\overline{w_i}$ and $\overline{c_{i,j}}$ are the average execution time and average communication time of task t_i. The downward rank value for the t_{entry} is equal to zero. $\text{rank}_d(t_i)$ is the longest distance from the $entry\ node$ to task t_i, excluding the computation cost of task itself [21]. The Downward Rank is calculated once based on the different instance types available. Therefore, task selection with Downward Ranks does not increase the time complexity of the algorithm.

D. Instance Selection

By the time we perform instance selection, we have already assigned each task to a level, determined the deadline for each level, and the priority of each ready task. During instance selection, we therefore need to trade off execution time and cost. We start by calculating both the time and the cost of executing each task on each instance type, given by equations 8 and 9, forming two sets Time and Cost.

Firstly, the time needed for the current task, t_i, on the instance p_j is calculated by $\text{ECT}(t_i, p_j)$. The ECT is the earliest time that a task can finish on an instance which is defined in equation 4. Using this, we can then compute the Time set using:

$$\text{Time}\ {}^{p_j}_{t_i} = \frac{Level^{\ell_{t_i}}_{deadline} - \text{ECT}(t_i, p_j)}{Level^{\ell_{t_i}}_{deadline} - \text{ECT}(t_i)} \tag{8}$$

In equation (8), $Level_{deadline}$ is the deadline that is assigned to the level which contains the current task. Also, $\text{ECT}(t_i)$ is the minimum execution time among all instances that keeps our current task on schedule.

$\text{Time}\ {}^{p_j}_{t_i}$ assesses how much the estimated level deadline of the current task differs from the earliest completion time of task on the instance p_j. The values of Time set for task t_i are related to instance types, wherein the lower value of Time set means running on a cheaper instance. The reason is that $\text{ECT}(t_i, p_j)$ is bigger on an instance with a lower processing capacity. As a result, in equation (8) the result value will be smaller. Also, if the value of Time is negative, it means that the current task on the selected instance will exceed the level deadline ($\text{ECT}(t_i, p_j) > Level^{\ell_{t_i}}_{deadline}$).

In the Cost set, C_i refers to the cost of scheduling the current task t_i on instance p_j. In equation, (9), the worse cost (maximum cost) and best cost (minimum cost) of executing the task t_i among all instances are C_{worse} and C_{best}, respectively.

$$\text{Cost}\ {}^{p_j}_{t_i} = \frac{C_{worse} - C_i}{C_{worse} - C_{best}} \tag{9}$$

To find the best instance, we use a Cost Time Trade-off Factor (CTTF) in equation (10) that considers a trade-off between cost and time.

$$\text{CTTF}\ {}^{p_j}_{t_i} = \frac{\text{Cost}\ {}^{p_j}_{t_i}}{\text{Time}\ {}^{p_j}_{t_i}} \tag{10}$$

TABLE I: Instance Types

Type	ECU	Cores	Memory(GB)	Cost ($)
m1.small	1	1	1.7	0.06
m1.medium	2	1	3.75	0.12
m1.large	2	2	7.5	0.24
m1.xlarge	2	4	15	0.48
m3.xlarge	3.25	4	15	0.50
m3.2xlarge	3.25	8	30	1

E. The Full PDC Algorithm

The pseudo code for the PDC algorithm in is given in algorithm 1

Algorithm 1 The PDC Algorithm

```
1: procedure FIND INSTANCE(DAG, $T_D$)
2:     for all task $t_i \in$ DAG do
3:         calculate the $\text{level–number}(t_i)$ as defined in equation 1
4:     end for
5:     categorize tasks on tasks level set as defined in (2)
6:     for all levels in DAG do
7:         calculate the level deadline as defined in (6)
8:     end for
9:     if $T_D < Level^1_{deadline}$ then
10:        terminate
11:    end if
12:    put $t_{entry}$ on task ready list
13:    while there is an unscheduled task in DAG do
14:        $t_i \longleftarrow$ select the task with the highest rank
           from ready list
15:        for all instances $p_j \in P$ do
16:            calculate the Time set as defined in (8)
17:            calculate the Cost set as defined in (9)
18:            calculate the CTTF as defined in (10)
19:        end for
20:        $BestInstance \longleftarrow$ choose the Minimum ECT
           instance among those that have zero cost for
           executing the current tasks

21:        if $BestInstance\ is\ null$ then
22:            $BestInstance \longleftarrow$ choose the instance that
               has the highest CCTF value
23:        end if
24:        Schedule task $t$ on $BestInstance$
25:        Update the task ready list
26:    end while
27: end procedure
```

With cloud instances, we know the types of instances with their different characteristics in advance - as this is part of the pay-per-use cloud model. The characteristics of instances used in this paper are based on the Amazon EC2 instance configurations (Table I) – the pricing is from mid-2014.

In addition, when an instance is first provisioned, the instance is billed on an hourly interval until it is terminated.

Therefore the first task assigned to an instance in a particular billing interval is assigned the entire cost of that interval. As a consequence, if other tasks can be executed during that paid interval, once the first task has completed, then their individual execution cost is effectively zero. Therefore, during instance selection, we first prioritise the reuse of such instances (when Cost in equation (9) is 1), providing that the level deadline is not exceeded (when Time in equation (8) is positive). If there are more than one paid instances, the PDC selects the one that has the minimum execution time (faster instances). If no such instances are available, we will attempt to use a provisioned but as yet unused (in this interval) instance, or as a last resort provision an entirely new instance.

V. Performance Evaluation

Simulation can be considered the first phase to evaluate new techniques for workflow scheduling problem. It allows researchers to test the performance and evaluate the proposed algorithms free of cost. For this purpose, all three algorithms are implemented and evaluated in CloudSim [29]. Our simulation is configured with one data-center and five different instance types. The characteristics of instances are based on the Amazon EC2 instance configurations given previously in (Table I).

The processing capacity of EC2 units is estimated at a Million Floating Point Operations Per Second (MFLOPS)as described in [30]. In an ideal cloud environment, there is no delay in resources allocation between requesting to the provision of a instance in data-centers. However, some factors such as the time of day, the operating system type, the instance type, the location of the data center and the number of requested resources at the same time, can cause a delay in the startup time [31]. Therefore, in our simulation, we adopted a 97-second boot time based on the measurements reported in [31] for the Amazon EC2 cloud.

In order to evaluate the performance of the algorithms with a realistic load, we use three different well documented scientific workflows: Cybershake, Montage and LIGO [32]:

- CyberShake: The CyberShake workflow is used by the Southern California Earthquake Center (SCEC) to characterise earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) technique. The workflow structure is shown in Fig 2(a).
- Montage: Montage is used to generate the custom mosaics of the sky using input images in the Flexible Image Transport System (FITS) format. The workflow structure is shown in Fig 2(b).
- LIGO: The Laser Interferometer Gravitational Wave Observatory (LIGO) attempts to detect gravitational waves produced by various events in the universe as per Einstein's theory of general relativity.The workflow structure is shown in Fig 2(c).

To evaluate the performance sensitivity of algorithms, different deadline intervals are assigned to the scientific datasets. These intervals consist of a range of tight deadlines through to to more relaxed ones. To achieve this, the fastest and slowest possible schedules need to be calculated as the baseline schedules. If all the communication costs between tasks are removed and each task is executed on the fastest instance type, the fastest schedule will be reached. This schedule can be considered as an approximation of the lowest possible value of makespan. Another baseline schedule is the slowest schedule, which is calculated by assigning all tasks on the instance with the lowest cost – as the cheapest instances are also the slowest. The slowest schedule is an estimation of the largest value of makespan. To determine deadline values, we define equation 11 in which the deadline varies from tight to moderate to relaxed.

$$\text{deadline} = \text{fastest} + \alpha * \frac{(\text{slowest} - \text{fastest})}{10} \tag{11}$$

The deadline factor α has two different sets:

1) $\alpha \in [0.1, 1]$: The first set is considered for very tight deadline intervals, which starts from 0.1 to 1 with increasing step length of 0.1. Obviously, if α equals 0, the defined deadline is the fastest schedule and is unachievable.
2) $\alpha \in [1, 5]$: The second set starts from 1 with step length of 0.5, ends to 5.

Amazon EC2 instances are charged on an hourly interval from the time of provisioning, even if the instance is only used for a fraction of that period. Therefore in the experiments presented in this paper we use a time interval of 60 minutes. We also ran the experiments with intervals of 5 and 30 minutes - however these results are not significantly different and for brevity we omit them from the paper.

In terms of the size of the three workflows, we evaluated the algorithms with workflows of 50, 100 and 200 tasks. However, as with the interval size, the results did not vary significantly, in this case we chose the middle-ground of 100 task workflows for the results presented in this paper. Finally, each experiment was repeated 50 times.

VI. Experimental Results

Two state-of-the-art algorithms, IC-PCP [6] and GAIN [7] were chosen in order to evaluate the performance of our PDC Algorithm by comparison.

As the GAIN algorithm was originally intended for a grid environment, we modified the algorithm to better suit the cloud. GAIN starts with an initial assignment which needs a minimal amount of money. This initial assignment is called the cheapest plan because all tasks are scheduled in the cheapest and slowest instance. The idea of GAIN is to start from the cheapest plan and try to minimize the makespan by swapping tasks between different instances until the specified budget is met. For this purpose, the algorithm uses the equation 12 to re-assign tasks on different instances.

$$\text{GAIN Weight}(i, m) = \frac{T_{old} - T_{new}}{C_{new} - C_{old}} \tag{12}$$

where T_{old} and C_{old} are the execution time and cost of t_i on the assigned instance by initial assignment, respectively. T_{new}

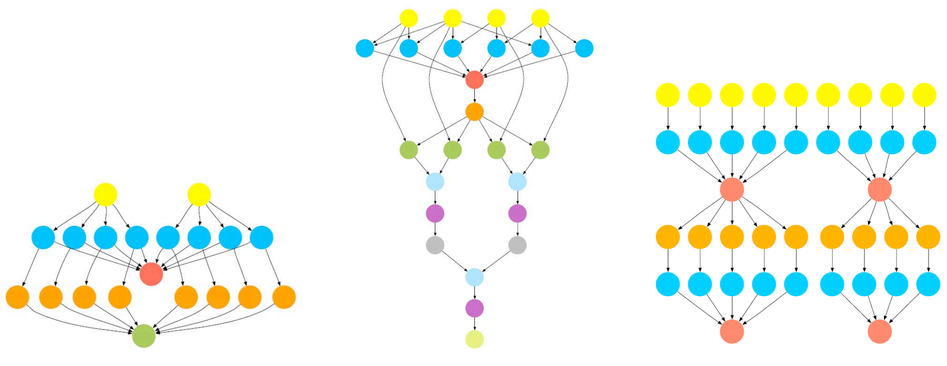


Fig. 2: Scientific workflow structure, reproduced from [32]

is the execution time of t_i on resource m. Also, C_{new} is the cost of executing t_i on resource m. In the original algorithm, reassignment is repeated till the monetary cost of the found schedule is less than the defined budget. We modified the reassignment condition so that the makespan can also meet the deadline.

The IC-PCP has two main phases which are deadline distribution and planning. In the first phase, user-defined deadline is distributed among all tasks. The IC-PCP starts by trying to find a set of tasks which are in critical path. The path from the entry node to exit node that has the longest average execution time is the critical path of the workflow. In critical path, these tasks are called critical tasks. All critical tasks will be executed on the same instance to remove the communication costs. Cost optimization is considered in their algorithm by selecting the cheapest applicable instance, which is the instance that all tasks on the path can schedule before their latest finish time. Recursively, a critical path for the unscheduled task's successors on the partial critical path is calculated and the process is continued until all tasks have been executed.

In related work, much is made of lower execution costs, however, in some cases, these lower costs include a number of tasks that were not able to be scheduled, thus potentially skewing results. We believe that in many cases it is more important to achieve a viable schedule - even at a slightly greater financial cost. Therefore, we elected to look at the Success Rate (SR) of each algorithm – and present those results in a form that allowed the costs to be simultaneously appreciated. The success rates for the three algorithms are presented in Fig. 3 and the associated costs in Figs. 4 and 5.

To compare the monetary cost between the algorithms, we cannot ignore the effect of failure in meeting the deadline in the computation of the average cost. To ensure the results are comparable we introduce the weighted cost:

$$Weighted\ Cost = \frac{\sum_{Cost}}{SR} \tag{13}$$

Here $\sum_{Cost}$ is the sum of the cost for experiments that meet the deadline.

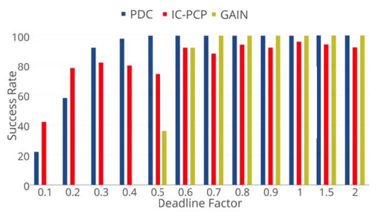
In these experiments, a failure is when an algorithm cannot find a makespan that can meet the required deadline. In the results presented we have selected the most interesting range for defined deadline from 0.1 to 2. What is significant in these results is that the structure and execution characteristics of the workflow appears to have a significant impact on success or failure – across a range of tight to relaxed deadlines. We intend to explore this in future work as understanding this aspect may allow improved algorithms.

The general results show that the PDC algorithm is the most able algorithm at finding a schedule in all but 2 of the 36 intervals (across the three workflows). Examining Figs. 4 and 5 reveals that in most cases this is accompanied by a moderate reduction in overall workflow execution cost. On the other hand, where the costs have increased, the increase is small.

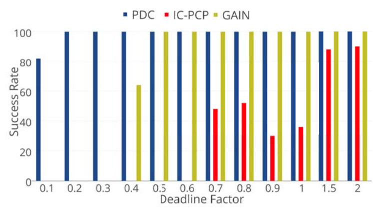
Overall, the GAIN algorithm has the worst success rate when the deadline is tight. In the first three intervals across all three workflows, GAIN is unable to schedule any tasks. However, with more relaxed deadlines, the GAIN algorithm achieves a 100% success rate - albeit at a significantly higher cost than PDC or IC-PCP.

IC-PCP managed fewer failures than PDC in the tightest two deadlines for the Cybershake workflow. Indeed PDC experienced a 100% failure in Cybershake when deadline factor, α, is 0.1. However, this was the exception rather than the rule, and in general IC-PCP across many intervals demonstrates some instability. The highest failure for IC-PCP occurs in Montage, when during the six tight intervals, IC-PCP can not find a schedule. A possible reason for the failure to schedule in IC-PCP is that in each step of the algorithm, a set of tasks on a critical path are selected and scheduled on the same instance to eliminate the communication cost. However, it makes it difficult for other parents or children of the executed task to be scheduled before their latest finish time. In addition the instance boot time is not considered in IC-PCP.

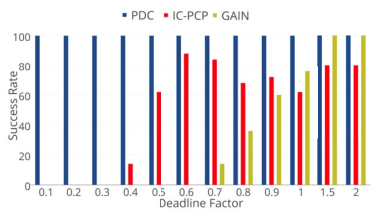
The results for PDC suggest that it favours computation intensive workflows such as LIGO in which it met all of the defined deadlines. IC-PCP and GAIN did not perform as well with tight deadlines in LIGO. The data intensive workflow



(a) Cybershake

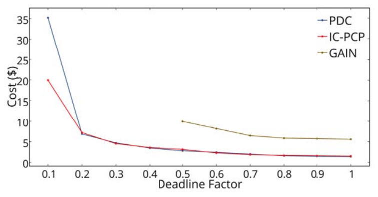


(b) Montage

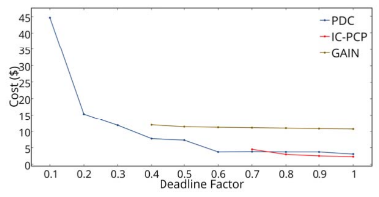


(c) LIGO

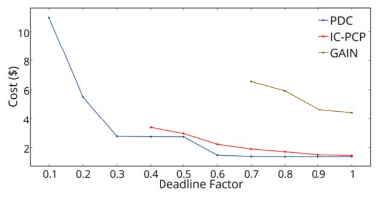
Fig. 3: Success Rate of Algorithms



(a) Cybershake

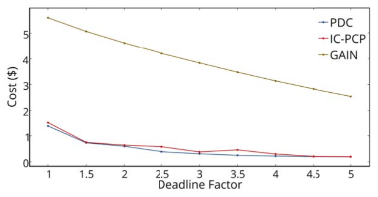


(b) Montage

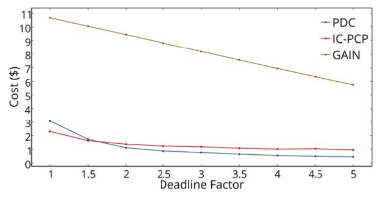


(c) LIGO

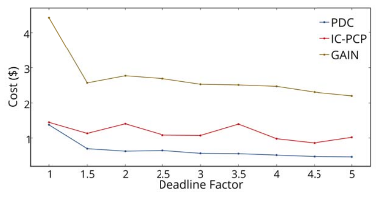
Fig. 4: Weighted Cost in Tight Deadlines



(a) Cybershake



(b) Montage



(c) LIGO

Fig. 5: Weighted Cost in Relaxed Deadlines

Cybershake was the worst case for PDC, with failures in scheduling tasks with tighter deadlines ($\leq$ .3 seconds). IC-PCP performed better in this period.

In terms out outright cost, there is a significant difference between GAIN and the other two algorithms over all three workflows. The GAIN algorithm in each step tries to reassign tasks until makespan reaches the deadline. As a result, there is no limitation on the number of instances that the algorithm can use. In addition, GAIN does not prioritise the use of existing instances. As a result, more instances are provisioned by GAIN, which have a direct effect on cost. This is a direct result of GAIN being originally designed for use in the Grid, where acquiring additional instances does not have a significant penalty.

Both PDC and IC-PCP have lower overall costs than GAIN. While in most cases there is only a small difference between PDC and IC-PCP in cost, in Cybershake, PDC does achieve significantly lower costs than IC-PCP for LIGO and most deadlines for Montage.

In general PDC is able to construct the most viable schedules in the majority of cases, with only significant failure rates for tight deadlines in Cybershake. In all other situations PDC comes out ahead of IC-PCP and GAIN. The success results for GAIN in turn are more stable than IC-PCP which appears to suffer from significant stability issues, where a more relaxed deadline can sometimes result in a lower success rate.

VII. Conclusion

In this paper we have presented the PDC algorithm for scheduling eScience workflows on commercial clouds. The PDC algorithm focuses on deadline constraints while minimizing costs. The PDC algorithm consists of 4 stages; workflow levelling to extract the maximum inherent parallelism of of workflow; proportional deadline distribution to appropriately partition the user deadline over the levels defined in the first stage; task selection to prioritise ready tasks; and instance selection to determine the best instance choice based on the execution estimates calculated in the distribution stage. This final step focuses on finding the best tradeoff of cost vs time, and includes both re-use of pre-provisioned instances and the creation of new instances on demand. The novelty in this work is contained in the proportional deadline distribution and instance selection.

To evaluate the PDC algorithm we constructed a CloudSim simulation and implemented the PDC algorithm along with implementations of GAIN (modified) and IC-PCP in order to evaluate PDC by comparison. For the simulation we utilised three well known eScience workflows, Cybershake, Montage and LIGO and evaluated all three algorithms for cost and scheduling success rate against a range of deadline constraints. All three algorithms showed general reductions in cost as the deadlines were relaxed, however, this was not the always the case in terms of success rate. In particular IC-PCP was shown to exhibit instability in scheduling success over a range of deadlines. Overall PDC is shown to achieve generally lower costs and increase the workflow task scheduling success rate leading to more viable workflow schedules. In the small number of cases where the cost of PDC was bettered by IC-PCP, the cost difference was small and a reasonable tradeoff for the generally higher scheduling success rate.

Our results show that cost-aware provisioning, such as PDC, will result in significant savings for scientists using the cloud, effecting more science for the available funding.

References

[1] 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.

[2] 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*, 2015.

[3] J. Ullman, "Np-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384 – 393, 1975.

[4] S. Yi, A. Andrzejak, and D. Kondo, "Monetary cost-aware checkpointing and migration on amazon cloud spot instances," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 512–524, 2012.

[5] 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.

[6] 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.

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

[8] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810 – 837, 2001.

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

[10] S. Smanchat and K. Viriyapant, "Taxonomies of workflow scheduling problem and techniques in the cloud," *Future Generation Computer Systems*, vol. 52, pp. 1–12, 2015.

[11] 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.

[12] 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.

[13] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for scheduling in distributed computing environments*. Springer, 2008, pp. 173–214.

[14] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107 – 131, 1999.

[15] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3-4, pp. 217–230, 2006.

[16] J. Yu, M. Kirley, and R. Buyya, "Multi-objective planning for workflow execution on grids," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, Washington, DC, USA, 2007, pp. 10–17.

[17] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 39, no. 1, pp. 29–43, Jan 2009.

[18] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, April 2010, pp. 400–407.

[19] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy incloud workflow systems," *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.

[20] M. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, April 2014.

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

[22] L. Bittencourt and E. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

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

[24] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization," *Scientific Programming*, 2015.

[25] M. Zhu, Q. Wu, and Y. Zhao, "A cost-effective scheduling algorithm for scientific workflows in clouds," in *IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, 2012, pp. 256–265.

[26] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011 – 1026, 2011.

[27] 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*, 2012, pp. 22:1–22:11.

[28] 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.

[29] 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.

[30] 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*. Springer Berlin Heidelberg, 2010, vol. 34, pp. 115–131.

[31] 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*, Washington, DC, USA, 2012, pp. 423–430.

[32] 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.