A Deadline Constrained Critical Path Heuristic for Cost-effectively Scheduling Workflows

Vahid Arabnejad*, Kris Bubendorfer*, Bryan Ng* and Kyle Chard†
*School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
†Computation Institute, University of Chicago and Argonne National Lab, IL, USA

***Abstract*— Effective use of elastic heterogeneous cloud resources represents a unique multi-objective scheduling challenge with respect to cost and time constraints. In this paper we introduce a novel deadline constrained scheduling algorithm, Deadline Constrained Critical Path (DCCP), that manages the scheduling of workloads on dynamically provisioned cloud resources. The DCCP algorithm consists of two stages: (i) task prioritization, and (ii) task assignment, and builds upon the concept of Constrained Critical Paths to execute a set of tasks on the same instance in order to fulfil our goal of reducing data movement between instances. We evaluated the normalized cost and success rate of DCCP and compared these results with IC-PCP. Overall, DCCP schedules with lower cost and exhibts a higher success rate in meeting deadline constraints.**

I. Introduction

Cloud computing enables significant computational leverage to be applied to many real world problems, be they commercial, industrial, medical or scientific. From a scheduling and budgeting point of view, the most interesting subset of these are those involving complex multi-stage processing operations that can be represented as workflows. Indeed, while cloud platforms provide enormous elastic computing capacity, they also pose unique multi-objective scheduling challenges with respect to cost and time constraints [1]. For example, cloud computing providers offer a wide variety of heterogeneous instance types (with different storage, CPU, and network performance) and implement explicit pay-per-use cost models. This flexibility, while powerful, may also result in inefficient usage with respect to performance and cost when naive provisioning and scheduling approaches are applied [2].

In this paper we introduce a novel deadline constrained scheduling algorithm, Deadline Constrained Critical Path (DCCP), that manages the scheduling of workloads on dynamically provisioned resources. To evaluate our general purpose algorithm we draw on exemplar workflows from eScience – a rich, well documented area utilizing workflow scheduling in a wide range of scientific applications, including astronomy, bioinformatics, earthquake science, and gravitational-wave physics [3]. Indeed, cloud computing is now a mainstream approach for conducting scientific analyses [4] and the development of scalable scientific services, such as the Globus Galaxies platform [5] is an active area of research.

Scheduling workflow tasks to resources while meeting workflow dependencies and meeting defined constraints is known as the Workflow Scheduling Problem (WSP) – a class of problem that is known to be NP-complete [6]. Unlike traditional approaches to this problem we focus on the unique characteristics of workflow execution on elastic cloud platforms. When executing workflows on cloud resources there are two distinct phases: resource provisioning and task scheduling [7]. The resource provisioning phase aims to determine the amount and type of resources required and then to reserve these resources for workflow execution. Given a set of resources, the workflow scheduling phase aims to then determine optimal execution order and task placement with respect to user and workflow constraints [8]. Prior research tends to focus solely on the second phase under the assumption that a pre-identified pool of (often homogenous) resources are used for execution, and with the goal of optimizing workflow execution time (makespan) without considering resource cost.

Our DCCP algorithm belongs to the list-based scheduling category [9] and consists of two main stages: firstly task prioritization, and secondly task assignment. In the first stage, a rank value is assigned to each task and then all tasks are sorted based on their rank. In the task assignment phase, tasks are allocated to suitable instances.

This paper is organized as follows: Section II reviews related work. In Section III, we define the workflow scheduling problem and describe our model. In Section IV, we present the DCCP algorithm. In Section V, we outline our CloudSim-based simulation followed by results and performance evaluation. Finally, we summarize our work in Section VI.

II. Related Work

Various algorithms, based on heuristic, search-based, and meta-heuristic strategies, have been proposed for efficient resource scheduling – where tasks are either considered independent (bag of tasks) or dependent (workflows). As a well studied problem there are several comprehensive reviews of workflow scheduling methods in distributed environments [7], [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 [12] can be classified as pure scheduling, while systems such as DRIVE [13] focus on provisioning. The majority of Cloud scheduling systems necessarily include both scheduling and provisioning stages, and are therefore the focus of this section.

Best effort scheduling algorithms [14]–[18] aim to minimize makespan. Heterogeneous Earliest Finish Time (HEFT) [16]–one of the most common scheduling heuristics–prioritizes

tasks based on communication and execution cost. For each task, upward rank and downward rank is calculated based on execution cost and the cost to communicate variables to all dependent tasks. All tasks are sorted in descending rank order (upward or downward). For each task, starting with the highest priority task, a processor is selected that that results in the earliest completion time for that task. HEFT represents a greedy short-term algorithm, it is therefore not always optimal when scheduling complex workflows. Several extensions to HEFT have been proposed to improve long-term performance, for example by using lookahead information to minimize the estimated completion time of the successors of the task being scheduled [18].

Quality of Service (QoS)-constrained workflow scheduling approaches aim to optimize QoS variables while meeting user-defined constraints. For example, methods sometimes consider budget [12], [19], [20] or deadline [1], [21], [22] constraints while trying to optimize other parameters. In contrast to best-effort scheduling, QoS-constrained approaches are more suitable for real-world scientific (and other) applications [7]; however, these approaches can be intractable when multiple constraints and objectives are expressed.

To tackle the workflow scheduling problem with multiple constraints researchers have explored various meta-heuristic search methods. For example, guided random searches such as Genetic Algorithms (GA) [23], [24], Ant Colony Optimization (ACO) [25] and Particle Swarm Optimization (PSO) [26], [27]. While these approaches usually produce acceptable scheduling performance in cloud environments, they are typically expensive and time-consuming due to their need for an initialization phase and large solution space. Moreover, the overhead of finding a feasible schedule increases rapidly with the size of workflow. Thus, meta-heuristics are considered unsuitable for real-time scheduling of large or complex workflows.

From a cost perspective, researchers have explored various approaches. For example, Deadline Early Tree (DET) [21] is a deadline constrained heuristic which takes into account the optimal cost solution. In DET, 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 allocated using an iterative process to maximize time windows. The Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm [28], focuses on optimizing cloud-bursting from from private to public clouds. The initial schedule starts to execute tasks on private cloud resources; if initial scheduling cannot meet user defined deadlines additional resources are leased from a public cloud.

The Infrastructure as a service (IaaS) Cloud Partial Critical Paths (IC-PCP) algorithm [1] aims to minimize execution cost while meeting user defined deadlines. All tasks in a partial critical path (PCP) are scheduled to the 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 does not consider the boot and deployment time of VMs. The Enhanced IC-PCP with Replication (EIPR) algorithm [8] is proposed as an extension to IC-PCP that is able to use idle instances and budget surplus to replicate tasks. The experimental results show that the likelihood of meeting deadlines is increased by using task replication. However, in EIPR task replication comes at an opportunity cost to the user. In this paper we also utilize idle time to reduce overall costs, while still meeting deadlines.

The Partitioned Balanced Time Scheduling (PBTS) algorithm [22] aims to minimize the cost of workflow execution while meeting a user-defined deadline constraint. The PBTS algorithm estimates the minimum number of instances required in order to minimize execution cost. Malawski et al. [29] present three algorithms for scheduling a set of workflows in IaaS clouds. Their algorithms aim to maximize the number of workflows that can be finished while meeting given budget and deadline constraints. However, in these algorithms as well as PBTS, the authors consider only one instance type rather than the wide variety of types that are currently supported by commercial providers.

III. Problem Definition

In most cases workflows are described as a Directed Acyclic Graph (DAG). A workflow is defined as a $G = (T, E)$ where T is a set of tasks represented by vertices and E is a set of dependencies between tasks (represented by directed edges). 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 and all data from t_i has been received. That is, the edge describes that task t_i is the direct 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. Task t_i cannot start until all parents have completed. 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. In order to ensure the DAG has only one input and one output, two dummy tasks t_{entry} and t_{exit} are added to the workflow. These dummy tasks have zero execution cost and zero communication data to other tasks.

When executing a workflow, users may have one or more QoS constraints under which they wish the workflow to execute. Here we consider one such QoS factor, deadline, while also aiming to limit cost. Deadline is defined as the maximum time, after submitting a workflow, that the workflow is allowed to execute. The objective of this problem can be described as:

$$\text{Minimize } \{Cost\}, \tag{1}$$

subject to,

$$Makespan \leq Deadline \tag{2}$$

IV. The DCCP Algorithm

In this section, we present our Deadline constrained Critical Path (DCCP) algorithm. The DCCP algorithm is a list-based scheduling strategy, that aims to meet user defined deadline (T_D) while minimizing execution cost. Generally, list-based scheduling algorithms are divided into two main phases: (i) task prioritization, and (ii) task assignment. Task prioritization orders tasks according to some ranking function while task assignment allocates each task to the most suitable resource as defined by an objective function. Here, we consider minimizing execution cost and meeting deadlines as the main objective function.

A. Task Prioritization

The first goal of our algorithm is to partition tasks into different levels based on their respective parallel and synchronization requirements. We aim to maximize potential task parallelism by partitioning tasks such that there are no dependencies between tasks in each 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, two algorithms have been propsed: Deadline Bottom Level (DBL) [30] and Deadline Top Level (DTL) [31]. 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 in 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 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:

$$\text{level–number}(t_i) = \max_{t_j \in \succ(t_i)} \{\text{level–number}(t_j) + 1\} \quad (3)$$

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.

$$\text{TLS}(\ell) = \{t_i | \, \text{level–number}(t_i) = \ell\} \quad (4)$$

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

1) Proportional Deadline Distribution: Once all tasks are assigned to their respective level, we next proportionally distribute a share of the user deadline (T_D) across each level. Each sub-deadline assigned to a level is termed the level deadline ($Level_{deadline}$). To meet the overall deadline, we attempt to ensure that every task in a level can complete its execution before the assigned sub-deadline. Firstly, the initial estimated deadline for each level (ℓ) is calculated by:

$$Level^{\ell}_{deadline} = \max_{t_i \in \text{TLS}(\ell)} \{\text{ECT}(t_i)\} \quad (5)$$

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

$$\text{ECT}(t_i) = \text{exec}_{min}(t_i) + \max_{t_k \in \prec(t_i)} \left\{ Level^{\ell_{t_k}}_{deadline} + \overline{c_{i,k}} \right\} \quad (6)$$

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 5, 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 execution in parallel.

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

$$\propto_{deadline} = \frac{T_D - Level^{1}_{deadline}}{Level^{1}_{deadline}} \quad (7)$$

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

We then add this proportion to each level based on the length of each level deadline:

$$Level^{\ell}_{deadline} = Level^{\ell}_{deadline} + (\propto_{deadline} * |Level^{\ell}_{deadline}|) \quad (8)$$

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

2) Constrained Critical Path (CCP): A Critical Path (CP) is the longest path from the entry to exit node of a task graph [14]. The length of critical path ($|CP|$) is calculated as the sum of computation costs and communication costs, and can be considered as the lower bound for scheduling a workflow. Several heuristics have been proposed that utilise critical paths in the workflow scheduling problem [14]–[16]. The set of tasks containing only the tasks ready for scheduling constitutes a constrained critical path (CCP) [32]. A task is ready when all its parents have been executed and all data required by the task has been provided.

In our proposed DCCP algorithm, we find all CCP in a workflow based on HEFT *upward rank* and *downward rank* [16]. We first describe the standard calculation of *upward rank* and *downward rank* before presenting our modified method. The *upward rank* is the length of critical path from the task t_i to the task t_{exit} and is calculated by:

$$rank_u(t_i) = \overline{w_i} + \max_{t_j \in \succ(t_i)} (\overline{c_{i,j}} + rank_u(t_j)) \quad (9)$$

where $\overline{w_i}$ and $\overline{c_{i,j}}$ are the average execution time and average communication time of task t_i, respectively. The reason that this rank is called upward rank is that the ranking process starts from the $exit\ task$ and ranks are calculated recursively by traversing the DAG to the $entry\ task$. The *upward rank* value for the t_{exit} is:

$$rank_u(t_{exit}) = \overline{w_{exit}} \quad (10)$$

The *downward rank* starts from the $entry\ task$ and is computed recursively by traversing the DAG to the $exit\ task$.

$$rank_d(t_i) = \max_{t_k \in \prec(t_i)} (\overline{w_k} + \overline{c_{k,i}} + rank_d(t_k)) \quad (11)$$

The *downward rank* for the t_{entry} is equal to zero. $rank_d(t_i)$ is the longest distance from the $entry\ node$ to task t_i, excluding the computation cost of task itself, where $rank_u(t_i)$ is the length of the critical path from task t_i to the $exit\ node$, including the computation cost of the task itself [16]. The *upward rank* and *downward rank* is calculated once based on the different instance types available. Therefore, rank calculation does not increase the time complexity of the algorithm.

In our proposed algorithm, we apply a new ranking method as defined in the following:

modified upward rank :

$$Mrank_u(t_i) = \overline{w_i} + \sum_{t_j \in \succ(t_i)} (\overline{c_{i,j}}) + \max_{t_j \in \succ(t_i)} (rank_u(t_j)) \quad (12)$$

modified downward rank :

$$Mrank_d(t_i) = \sum_{t_k \in \prec(t_i)} (\overline{c_{k,i}}) + \max_{t_k \in \prec(t_i)} (\overline{w_k} + rank_d(t_k)) \quad (13)$$

The difference between our modified rank from standard rank is that the modified rank aggregates a task's predecessors' or successors' communication time instead of selecting the maximum. With the modified rank, those tasks with higher out-degree or in-degree have higher priorities. As a result, they have a greater chance to execute first and more tasks on the next CCP can be considered as ready tasks.

In this paper, we use the sum rank to find all CPs [16]:

$$rank_{sum} = rank_u + rank_d \quad (14)$$

All tasks are first sorted based on their $rank_{sum}$ values and those tasks with the highest values are selected as the first CP. All tasks in the first CP are labeled as visited tasks. Proceeding in the same way, all CPs in a workflow can be found. Our algorithm for finding CPs is given in algorithm 1.

3) An illustrative example: We now present an example that demonstrates how we find CCPs and the differences between standard and our modified rank. We consider a sample DAG that contains 11 tasks as shown in Fig. 1. The numbers above each edge shows the data transfer time between tasks. Average execution time ($\overline{w_i}$) of each task is displayed in table I. The first CP is obtained based on the highest sum rank which is the aggregation of *upward rank* and *downward rank* (0→1→4→9→11). Regardless of any previously visited tasks, proceeding in the same way, other CPs are found as displayed in table II. The next step is traversal of CPs to find CCPs in a round-robin order. The first CCP contains 0→1 as other tasks in the first CP are not yet ready. For example, consider task 4 which is in the first CP, this task can not be added to the CCP as one of its parents, task 2, has not yet been added to any CCPs. As no more ready tasks can be found in the first CP, a second CP is considered to build a new CCP. In the next CP we have task 2 which is a ready task, as its only parent has already been included in a previous CCP. The second CCP consists of three tasks (2→5→8) having excluded task 10 from the second CP. Similarly, other CCPs are generated by using the remaining CPs. The different CCPs calculated by our modified rank approach are presented in Table III.

Algorithm 1 Find Critical Paths

```
procedure FIND CP(DAG (G))
    for all task t_i ∈ DAG do
        calculate the rank_u, rank_d and rank_sum
    end for
    CPlist⟵null
    while there is an unvisited task in G do
        t_i ⟵ biggest rank_sum
        CP⟵ null
        while t_i is not null do
            Add t_i to CP
            t_i ⟵ max_{t_j ∈ ≺(t_i)} (rank_sum t_j)
        end while
        Add CP to CPlist
    end while
end procedure
```

TABLE I: Ranks values

		Standard Rank			Modified Rank		
Task	$\overline{w_i}$	$rank_u$	$rank_d$	$rank_s$	$rank_u$	$rank_d$	$rank_s$
0	22	190	0	190	284	0	284
1	29	142	48	190	142	48	190
2	22	150	38	188	203	38	241
3	20	96	39	135	96	39	135
4	27	84	106	190	84	115	199
5	21	110	78	188	140	78	218
6	9	65	74	139	65	85	150
7	14	70	115	185	89	115	204
8	12	75	113	188	75	113	188
9	11	41	149	190	41	173	214
10	21	44	144	188	44	156	200
11	10	10	180	190	10	236	246

TABLE II: CPs and CCPs based on standard ranks

Critical Path	Constrained Critical Path
0→1→4→9→11	0→1
2→5→8→10	2→5→8
3→6	3→6
7	7
	4→9
	10
	11

B. Task assignment

One of the main advantages of cloud infrastructures is the ability to provision instances on-demand. Unlike grid and cluster systems that offer limited and typically static

Fig. 1: A sample DAG with 11 tasks

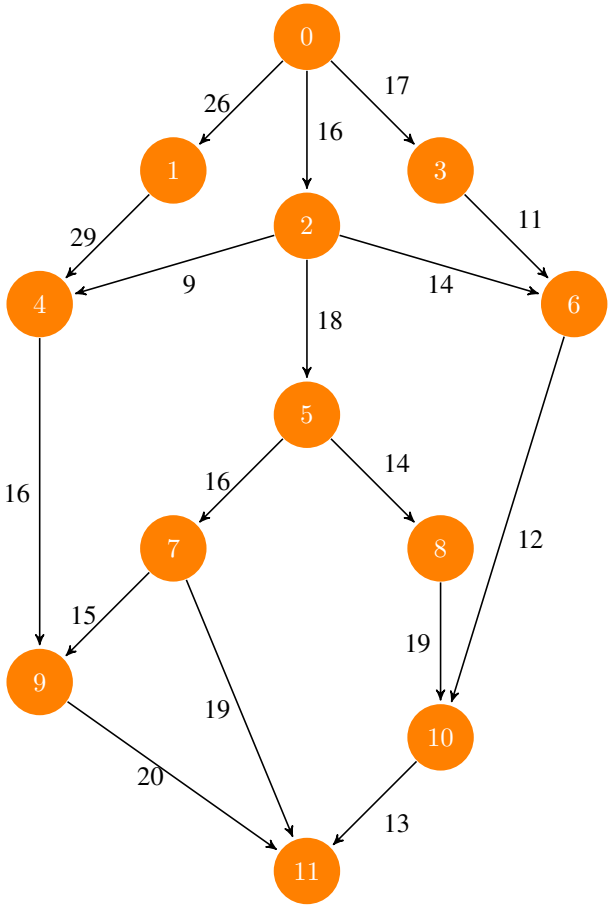

TABLE III: CPs and CCPs based on modified ranks

Critical Path	Constrained Critical Path
0→2→5→7→11 1→4→9	0→2→5→7 1→4→9
3→6→10 8	3→6 8 10 11

resources to executing workflow applications, clouds offer flexible capacity in which different instances can be launched, used and shut down on request.

In the task assignment phase we aim to identify the most appropriate instance to execute CCPs. Note: all tasks in a CCP are executed in the same instance with the goal of avoiding communication cost between them. The task assignment decision aims to minimize the total cost of workflow execution while also attempting to meet the CCP's sub-deadline. The time needed for the current CCP, (CCP_i), to execute on the instance p_j is calculated by $\mathrm{ECT}(CCP_i, p_j)$. Work in scheduling generally assumes such an estimate can be calculated. In practice this is difficult, however work is underway to profile workflow tools and underlying cloud systems to provide usable estimates for use in production systems [33]. The ECT is the earliest time that a CCP can complete execution on a instance (as defined in equation 6 for a single task).

The differences between the estimated level deadline and earliest completion time of the current CCP on the instance p_j is determined by:

$$\mathrm{Time}\ {}^{p_j}_{CCP_i} = Level^{\ell_{t_i}}_{deadline} - \mathrm{ECT}(CCP_i, p_j) \qquad (15)$$

where $Level_{deadline}$ is the deadline that is assigned to the level which contains the last task t_i on the current CCP. There is a possibility that this value may be negative which means the current CCP exceeds the level deadline ($\mathrm{ECT}(CCP_i, p_j) > Level^{\ell_{t_i}}_{deadline}$). $Cost_{CCP_i,p_j}$ is the cost of executing all tasks on current CCP on instance p_j.

Algorithm 2 Select an instance

```
    procedure INSTANCE SELECTION(CCP_i)
2:      F ⟵ find all instances that have zero cost for CCP_i
        M ⟵ find all instances that can meet sub-deadline
              for CCP_i
4:      if (F ∩ M) then
            SelectedInstance ⟵ minECT(F ∩ M)
6:      else if (M) then
            SelectedInstance ⟵ minCost(M)
8:      else
            SelectedInstance ⟵ minCost(All instances)
10:     end if
    end procedure
```

The pseudo code presented in algorithm (2) considers three different scenarios to find the most appropriate instance:

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 a CCP as well as ensuring that the earliest completion time does not exceed the level deadline, as it is shown in algorithm (2). The instance with minimum ECT is then selected (the fastest one).
2) If no instances can be found in the previous step, our algorithm provisions a new instance. For example, at the beginning of the scheduling process when an instance is assigned to the first CCP. For this purpose, DCCP searches among instances that can meet the level deadline and selects the cheapest one.
3) In tight deadlines, there is a possibility that none of the instances can meet the task level's sub-deadline (i.e.,

when $\text{Time}_{CCP_i}^{p_j}$ is negative). If this condition for a CCP is met, it does not mean that its impossible to meet the overall user defined deadline. Rather, it means that the sub-deadline will be violated. In this case we select the best available instance - as overall the schedule may still be met.

V. Evaluation

In this section we compare the performance of DCCP with another well known algorithm, IC-PCP [1]. We present two versions of our algorithm, DCCP(SR) and DCCP(MR) which are based on the Standard Rank (SR) and Modified Rank (MR), respectively. In the DCCP(MR) algorithm we also use the level by level method to find critical paths. In this method, the highest rank among all tasks in the first level is selected. Then, among the selected tasks' successors in the second level, the task with highest rank is chosen. Repeating this approach, all other critical tasks are selected until the last level is reached. The second CP is found by restarting the selection from the first level.

A. CloudSIM

We apply a simulation-based approach to compare the performance of the proposed algorithm. Specifically, we use CloudSim [34], 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 IV. The average bandwidth between instances is fixed to 20 MBps, based on the average bandwidth provided by AWS [35]. The processing capacity of an EC2 unit is estimated at one Million Floating Point Operations Per Second (MFLOPS) [36]. 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 [37]. Therefore, in our simulation, we adopted a 97-second boot time based on previous measurements of EC2 [37].

In order to evaluate the performance of our algorithms with a realistic load, we use four common scientific workflows: Cybershake, Montage, LIGO and SIPHT. The characteristics of which have been analyzed in related work [3], [38]:

To evaluate the performance sensitivity of algorithms, we explore different deadlines chosen from and interval from tight to relaxed. To derive these intervals we calculate two baseline schedules, fastest and slowest.

- Fastest Schedule (FS): If all tasks on the main CP of a workflow are executed on the fastest instance type, the fastest schedule will be reached.

$$FS = \sum_{t_i \in CP} (w_i^j) \tag{16}$$

where w_i^j is the computation cost of task t_i on the fastest instance p_j.

TABLE IV: Instance Types

Type	ECU	Memory(GB)	Cost($)
m3.medium	3	3.75	0.067
m4.large	6.5	8	0.126
m3.xlarge	13	15	0.266
m4.2xlarge	26	32	0.504
m4.4xlarge	53.5	64	1.008
m4.10xlarge	124.5	160	2.520

- Slowest Schedule (SS): If all tasks on the CP of a workflow are executed on the slowest instance type, the slowest schedule will be reached.

$$SS = \sum_{t_i \in CP} (w_i^k) \tag{17}$$

where w_i^j is computation cost of task t_i on slowest instance p_k.

Based upon these schedules we calculate deadlines as follows:

$$\text{deadline} = FS + \alpha * (SS - FS) \tag{18}$$

The deadline factor $\alpha \in [0.1, 1]$ starts from 0.1 to consider very tight deadlines (near the fastest schedule) with increasing step length of 0.1. The interval ends at 1 which results in a deadline equivalent to the slowest schedule.

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 we use a time interval of 60 minutes in our simulations. We also ran the simulations with intervals of 5 and 30 minutes; however these results were not significantly different and are therefore omitted for brevity.

To compare performance with respect to workflow size we evaluated workflows with 50, 100 and 200 tasks. However, as these results did not vary significantly we present here only workflows with 100 tasks. We used the Pegasus workflow generator [38] 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 deadline factor, 50 distinct Pegasus generated workflows were scheduled in CloudSIM and these results are detailed in the following section.

B. Results

In this section we compare the performance of both versions of our DCCP algorithm (SR and MR) with IC-PCP [1]. To compare the monetary cost between the algorithms, we consider the cost of failure in meeting a deadline. A failure is when an algorithm cannot meet the required 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:

$$Weighted\ Cost = \frac{\sum_k Cost\,(k)}{R_s}, \tag{19}$$

where $Cost\,(k)$ the cost for experiments that meet the deadline (returned by the minimization in (1)). R_s is the success rate of each algorithm, calculated as the ratio between the number of simulation runs that successfully met the scheduling deadline and the total number of simulation runs (denoted by n_{Tot}), defined as:

$$R_s = \frac{n\,(k)}{n_{Tot}}, \tag{20}$$

where $n(k)$ is the cardinality of the set k and $n_{Tot} = 50$. We consider the cheapest schedule as the scheduling of all tasks on the cheapest instance [1]. For cost normalization, the obtained weight of each algorithm is divided by cheapest schedule.

In terms of cost comparison, Fig. 2 shows that both DCCP algorithms, in most cases, have a lower cost when compared with IC-PCP. With the tightest deadline, when the deadline factor $\alpha = 0.1$, IC-PCP has a lower cost in SIPHT and Cybershake, yet exhibits a 100% failure in LIGO and, in Montage, almost twice the cost. For the LIGO and Montage workflows – most of the time the DCCP algorithms can schedule workflows at nearly half the cost achieved by IC-PCP. In all scientific datasets DCCP(MR) has slightly better performance than DCCP(SR), except for the tightest deadline in Cybershake.

The success rate of each algorithm for each dataset is shown in Fig.3. In general, the results show that the DCCP algorithms can meet almost all deadlines successfully except in Cybershake. The success of IC-PCP with very tight deadlines, when $\alpha = 0.1$, is generally poor, as shown in Fig.3 with a 100% failure rate over 50 samples for the LIGO workflow. In general, the data indicates that DCCP is far less sensitive to deadline than IC-PCP. In production systems consistent behavior is of great importance. The inconsistency observed in IC-PCP is due in part to the high failure rate [8], [27], which we attribute to how IC-PCP selects its instances.

VI. Conclusion

In this paper we have presented the Deadline Constrained Critical Path (DCCP) algorithm for scheduling workflows on dynamically provisioned cloud resources. Our approach focuses on addressing the unique characteristics of workflow execution on cloud platforms, such as on-demand provisioning and instance heterogeneity, while minimizing cost and meeting user-defined deadlines. The DCCP algorithm builds upon the concept of Constrained Critical Paths (CCP) to execute a set of tasks on the same instance with the goal of reducing communication cost between instances. The DCCP algorithm first calculates CCPs based on two different ranking strategies (standard and our modified ranking) before assigning tasks to cloud instances. We evaluated the normalized cost and success rate of DCCP with both ranking strategies and compared these results with IC-PCP. Overall, DCCP schedules with lower cost and higher success rate to meet the workflow deadline. In a few cases where the deadline was particularly tight, IC-PCP was able to schedule workflows at a lower cost. In future work we aim to extend the DCCP algorithm to consider other QoS parameters such as budget and reliability, we would also like to better understand how the structure of a workflow impacts scheduling.

References

[1] 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.

[2] 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.

[3] 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.

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

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

[8] 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.

[9] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," ACM Computing Surveys (CSUR), vol. 31, no. 4, pp. 406–471, 1999.

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

[15] G. Sih and E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," Parallel and Distributed Systems, IEEE Transactions on, vol. 4, no. 2, pp. 175–187, Feb 1993.

[16] 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.

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

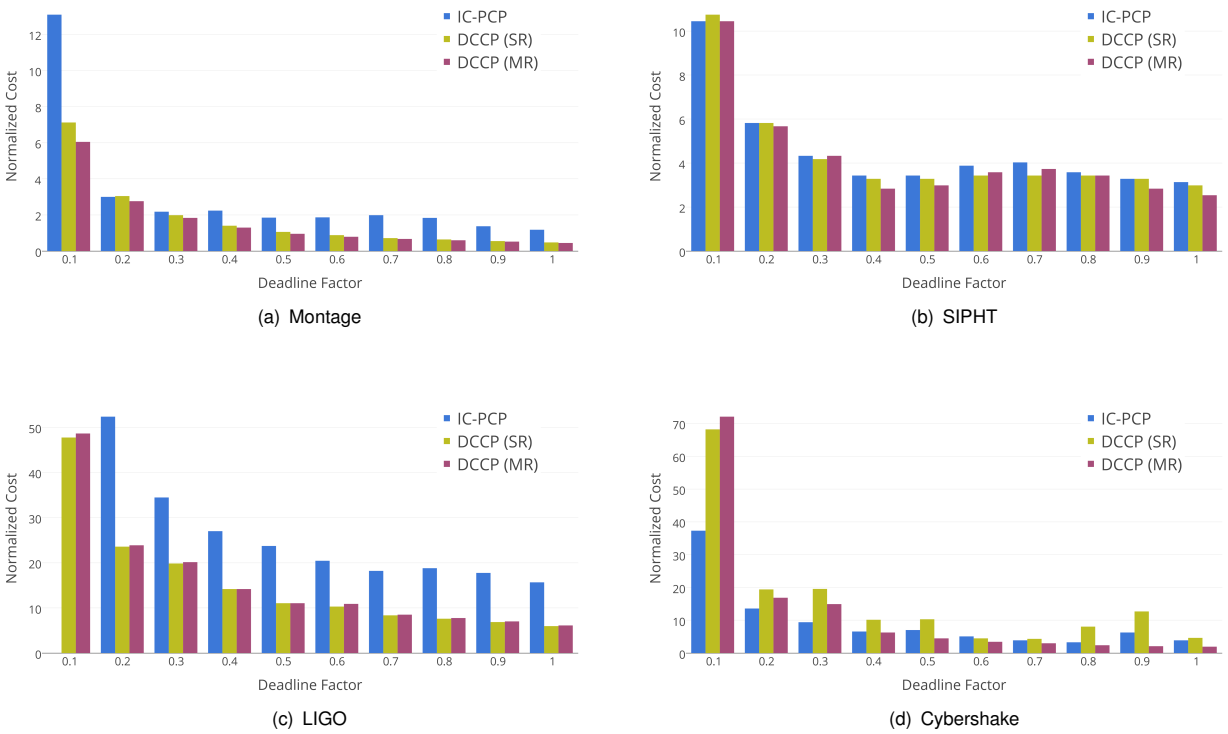

Fig. 2: Normalized Cost vs. deadline for four different datasets.

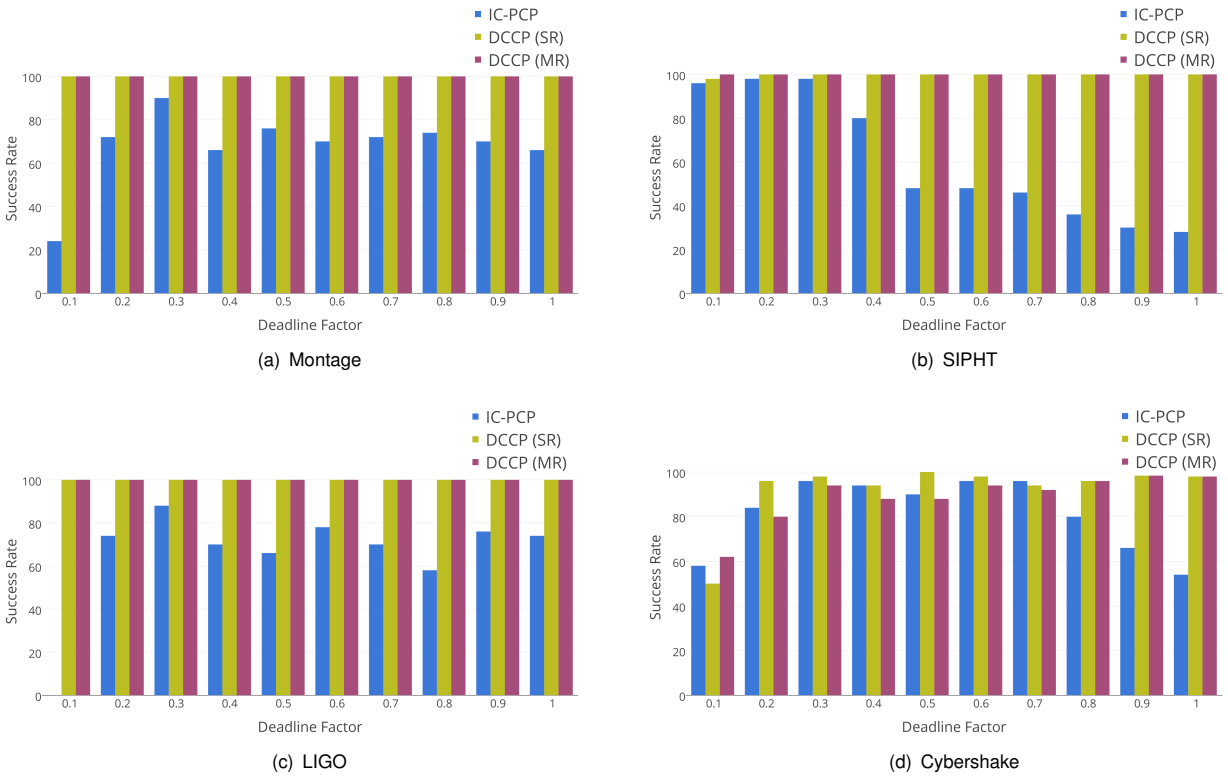

Fig. 3: Comparison of Success Rates for four different datasets.

[18] L. Bittencourt, R. Sakellariou, and E. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on, Feb 2010, pp. 27–34.

[19] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in Workflows in Support of Large-Scale Science, 2006. WORKS'06. Workshop on. IEEE, 2006, pp. 1–10.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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, ser. GRID '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 10–17.

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] R. Chard, K. Bubendorfer, and B. Ng, "Network health and e-science in commercial clouds," Accepted: Future Generation Computer Systems, June 2015.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.