Low-Time Complexity Budget-Deadline Constrained Workflow Scheduling on heterogeneous resources

Hamid Arabnejad[a], Jorge G. Barbosa[a,*], Radu Prodan[b]

[a]*LIACC, Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto, Portugal*
[b]*University of Innsbruck, Institute of Computer Science, Innsbruck, Austria*

Abstract

The execution of scientific applications, under the utility computing model, is constrained to Quality of Service (QoS) parameters. Commonly, applications have time and cost constraints such that all tasks of an application need to be finished within a user-specified Deadline and Budget. Several algorithms have been proposed for multiple QoS workflow scheduling, but most of them use search-based strategies that generally have a high time complexity, making them less useful in realistic scenarios. In this paper, we present a heuristic scheduling algorithm with quadratic time complexity that considers two important constraints for QoS-based workflow scheduling, time and cost, named Deadline-Budget Constrained Scheduling (DBCS). From the deadline and budget defined by the user, the DBCS algorithm finds a feasible solution that accomplishes both constraints with a success rate similar to other state-of-the-art search-based algorithms in terms of the successful rate of feasible solutions, consuming in the worst case only approximately 4% of the time. The DBCS algorithm has a low-time complexity of $O(n^2.p)$ for n tasks and p processors.

Keywords: Quality of Service, grids, clouds, list scheduling, search-based algorithms

*Corresponding author
Email addresses: `hamid.arabnejad@fe.up.pt` (Hamid Arabnejad), `jbarbosa@fe.up.pt` (Jorge G. Barbosa), `radu@dps.uibk.ac.at` (Radu Prodan)

1. Introduction

Utility computing is a service provisioning model that provides computing resources and infrastructure management to consumers as they need them, as well as a payment model that charges for usage. Service-oriented grid and cloud computing, which supply frameworks that allow users to consume utility services in a secure, shared, scalable, and standard network environment, have become the basis for providing these services.

Computational grids have been used by researchers from various areas of science to execute complex scientific applications. Utility computing has been rapidly moving towards a pay-as-you-go model, in which computational resources or services have different prices with different performance and Quality of Service (QoS) levels [6]. In this computing model, users consume services and resources when they need them and pay only for what they use. In this context, cost and time become two of the most relevant user concerns. Thus, the cost/time trade-off problem for scheduling workflow applications has become challenging. Scheduling consists of defining an assignment and mapping of the workflow tasks onto resources. In general, the scheduling problem belongs to a class of problems known as NP-complete [10].

Most research on workflow QoS aware scheduling considers the optimization of one QoS parameter, such as time, constrained to another QoS parameter, such as cost [2, 32]. Other approaches consider a bi-objective approach that consists in optimizing two QoS parameters, such as time and cost simultaneously [17, 19, 20], the constraints being mainly related to processor availability and load. Other combination of QoS parameters may be considered, such as time and reliability [11].

Workflow scheduling to satisfy multiple QoS parameters is becoming an active research area in the context of utility computing. Many algorithms have been proposed for multi-objective scheduling, but in most of them, meta-heuristic methods or search-based strategies have been used to achieve good solutions. However, these methods based on meta-heuristics or search-based strategies usually need significantly high planning costs in terms of the time consumed to produce good results, which makes them less useful in real platforms that need to obtain map decisions on the fly. In this paper, a low-time complexity heuristic, named Deadline-Budget Constrained Scheduling (DBCS), is proposed to schedule workflow applications on computational heterogeneous infrastructures constrained to two QoS parameters. In our model, the QoS parameters are time and cost. The objective of the

proposed DBCS algorithm is to find a feasible schedule map that satisfies the user defined deadline and budget constraint values. To fulfill this objective, DBCS implements a mechanism to control the time and cost consumption by each task when producing a schedule solution. To the best of our knowledge, the algorithm proposed here is the first low-time complexity heuristic for a bounded number of heterogeneous resources addressing two QoS parameters that obtains similar performances to higher-time complexity scheduling algorithms in a small fraction of the scheduling time.
The contributions of this paper are:

- a review of multiple QoS parameter workflow scheduling on heterogeneous resources;
- a new heuristic algorithm with quadratic complexity for workflow application scheduling, constrained to time and cost, on a bounded set of heterogeneous resources;
- similar performances of search-based state-of-the-art algorithms in a small fraction of the time that ranges from 0.004% to 4%;
- a realistic simulation considering a bounded multi-port model in which bandwidth is shared by concurrent communications;
- extensive evaluation with results for randomly generated graphs as well as for real-world applications.

The remainder of the paper is organized as follows. Section 2 describes related work. Section 3 defines the scheduling problem and describes the system model. Section 4 presents the proposed scheduling algorithm. Section 5 presents results, and Section 6 concludes the paper.

2. Related Work

Workflow scheduling has been extensively investigated. The scheduling strategies can be classified into two main categories: single and multiple QoS parameters.

On the single QoS parameter, the execution time of a workflow application, also called *makespan*, has been the major concern in most of the scheduling strategies. In [3, 7], a wide study on list scheduling algorithms is presented for *makespan* optimization.

The problem becomes more challenging when two or more QoS parameters are considered in the scheduling problem. Time, cost, energy and reliability are common QoS parameters considered in recent research work in this area. Many algorithms consider time and cost in their formulation but most of them perform: a) optimization of one parameter constrained to the other; b) optimization of both parameters in a bi-objective formulation; and c) a consideration of an unlimited number of resources, in particular for cloud platforms, where the strategy to accomplish time constraints is by allocating new computational instances.

Concerning the optimization of one parameter constrained to the other, Mao et al. [16] proposed an auto-scaling mechanism that automatically scales computing instances based on workload information to minimize the cost of the scheduling map while meeting application deadlines on cloud environments. Zeng et al. [30] proposed *ScaleStar*, a budget-conscious scheduling algorithm to minimize the execution time of large-scale many-task workflows in Clouds with monetary costs. Yu et al. [26] proposed a QoS-based workflow scheduling algorithm utilizing a Markov Decision Process approach for the service Grid that minimizes the total cost of the application while meeting the deadline constraints imposed by the user. Their algorithm first categorizes tasks into two classes: synchronization tasks (the nodes that have more than one parent or child) and simple tasks. Then, the original workflow is partitioned into sub-workflows, and based on the two classes of tasks, sub-deadlines are assigned to each partition. Finally, the cost optimized mapping for each partition is obtained, guaranteeing the application deadline. In [28], Yuan et al.proposed a time-cost tradeoff dynamic heuristic scheduling strategy to optimize the cost and time of the whole workflow. In addition, [29] presented a heuristic scheduling algorithm called DET (Deadline Early Tree), which minimizes cost with a deadline constraint. The communication time between tasks is not considered in their model. Sakellariou et al. [18] presented the LOSS1 algorithm to construct schedules that optimize time constrained to a cost. The algorithm uses initial assignments made by other heuristic algorithms to meet the time optimization objective; then, a reassignment strategy is implemented to reduce cost and meet the cost constraint that is specified by the user budget. Zheng et al., in [31] and [32], proposed the algorithm Budget-constrained Heterogeneous Earliest Finish Time (BHEFT), which optimizes the execution time of a workflow application constrained to a budget. Arabnejad et at.[2] proposed a Heterogeneous Budget Constrained Scheduling (HBCS) algorithm that guarantees an execution cost within the users specified budget and that minimizes the application execution time similarly to BHEFT. The results presented show that the HBCS algorithm achieves lower makespans, with a guaranteed cost per application. The algorithm proposed in this paper extends the HBCS algorithm to consider deadline (time) and budget (cost) as constraints. Similar to HBCS, in this paper, we propose a quality measure for each processor that combines time and cost constraints, which is used for processor selection and may not necessarily select the processor that guarantees the earliest finish time. BHEFT uses a different formulation that, in two steps, selects the set of affordable processors, the cost factor, and then selects the processor that minimizes the processing time. LOSS1 and BHEFT are also selected for comparison to the algorithm proposed in this paper, DBCS, as an alternative approach that optimizes time constrained to a budget.

The following algorithms use a bi-objective formulation. Talukder et al. [21] proposed a workflow execution planning approach using Multi-objective Differential Evolution (MODE) to satisfy the user time and cost constraint parameters. Chen et al. [9] proposed an ant colony optimization (ACO) to schedule large-scale workflows with various QoS parameters such as reliability, time, and cost in computational grids. Garg et al. [12] proposed a multi-objective non-dominated sort particle swarm optimization (NSPSO) approach to find schedule maps minimizing the makespan and total cost under the specified deadline and budget constraints. In [24], Yu et al. proposed a genetic algorithm (GA) approach for scheduling workflow applications constrained to budget and deadline, on heterogeneous environments. Two fitness functions are used to encourage the formation of individuals who satisfy the deadline and budget constraints. Prodan et al. [17] proposed a general bi-criteria scheduling heuristic called the Dynamic Constraint Algorithm (DCA) based on dynamic programming to optimize two independent generic criteria for workflows, e.g., execution time and cost. The DCA scheduling algorithm has two main phases: The first selects one criterion as primary and optimizes it and, in the second phase, optimizes the secondary criteria while keeping the primary criteria within a defined sliding constraint. GA and DCA are considered in this paper for comparison to the DBCS algorithm as state-of-the-art search-based algorithms with higher-time complexity. In particular, DCA performs a full domain search in the second phase of the algorithm.

Other algorithms were proposed considering the cloud on-demand resource allocation. In [1], two scheduling algorithms for cost minimization constrained to a deadline for IaaS Cloud environments were proposed. Due to on-demand resource provisioning, the time constraint can always be accomplished as long as the cloud provides an unlimited number of computational resources. This model corresponds to an unbounded set of resources, which differs from our context that considers a bounded set of processors. Other approaches, such as Malawski et. al [15], proposed three scheduling algorithms for scientific workflow ensembles on clouds to complete workflows from an ensemble under budget and deadline constraints. Our problem differs from such a model because we did not consider the composition of several interrelated workflows grouped into ensembles. A comprehensive survey about grid and cloud workflow scheduling is presented in [23].

Among all of these previous works, we select a few that are closer to our context. Most of the works consider a scheduling decision to optimize for some QoS parameter while being subjected to some user-specified constraint or optimize all of the QoS parameters to provide a suitable balance between them. In this study, our goal is not the optimization of QoS parameters; instead, we consider budget and deadline constraints rather than cost and time optimization as goals similarly to [24].

In terms of time complexity, most of the scheduling strategies mentioned above are search based and usually require long execution time before producing good results, making them less useful in a realistic computational infrastructure.

In this paper, our goal is to propose a novel low-time complexity workflow scheduling algorithm that addresses the budget and deadline constraints. To evaluate the performance of our scheduling strategy, we select four well-known algorithms, namely, DCA [17], Genetic Algorithm (GA) [24], LOSS1 [18] and BHEFT [31, 32]. The DCA and GA scheduling algorithms are search-based methods that obtain near-optimal schedule maps among the set of solutions. The LOSS1 algorithm tries to accomplish the objective function by using a task reassignment (rescheduling) strategy. The BHEFT is a low-time complexity scheduling strategy that minimizes time constrained to a budget, but it is included here as a low-time complexity alternative. In addition to these algorithms, we include a *RANDOM* strategy for task assignment, i.e., the scheduler randomly selects a resource without taking into account the execution time. This is the lower-time complexity algorithm, and it is also the base line for comparison.

3. Problem Definition

This section presents the system model, the application model and the scheduling objectives.

3.1. System Model

The target utility computing platform is composed of a set of heterogeneous resources that provide services of different capabilities and costs [6]. Processor price is defined so that the most powerful processor has the highest cost and the less powerful processor has the lowest cost. In a utility grid, the resource price is commonly defined and charged per time unit [17, 24, 31], so that if a task takes k time units to process in a resource that costs y euros per time unit, the cost of executing the task in that resource is $k \times y$ euros. In a cloud environment, the granularity of charging resource usage varies, charging per hour being a common practice, such as in Amazon Elastic Compute Cloud (Amazon EC2) [1], and partial hours are rounded up. This approach has the disadvantage that there is no cost benefit of adding resources for workflows that run for less than an hour [14]. It also makes the effective evaluation of scheduling algorithms and solutions difficult. Therefore, in this paper, we consider that a heterogeneous resource has a set of processors available and that each one has a price per time unit, so that the cost imputed to the workflow execution is only the effective used time. Although this assumption may seem unrealistic for the hour pricing model, it is only true if we consider the current dominant policy and individual user contracts with the provider. However, other models may be explored such as, for example, a group of users that rent a set of resources so that resources are shared and probably cost less than individual contracts. This is a feasible approach for a research group. Other players are more flexible and charge per minute, with a minimum of 10 minutes, such as the Google Compute Engine [2]. Additionally, there is an emergent market of cloud providers that put in the market the spare resources of their private clouds, so that a wider set of pricing models may be available in the near future. Therefore, for the sake of an unambiguous comparison of the results produced by the scheduling algorithms, as in [17, 24, 31], we consider the second as the time unit, as well as the unit for processor charging.

[1]http://aws.amazon. com/ec2

[2]http://cloud.google.com/compute/

3.2. Application Model

A typical workflow application can be represented by a Directed Acyclic Graph (DAG), a directed graph with no cycles. A DAG can be modeled by a three-tuple $G =< T, E, Data >$. Let n be the number of tasks in the workflow. The set of nodes $T = \{t_1, t_2, \cdots, t_n\}$ corresponds to the tasks of the workflow. The set of edges E represent their data dependencies. A dependency ensures that a child node cannot be executed before all its parent tasks finish successfully and transfer the required child input data. $Data$ is a $n \times n$ matrix of communication data, where $data_{i,j}$ is the amount of data that must be transferred from task t_i to task t_j. The average communication time between the tasks t_i and t_j is defined as:

$$\overline{C}_{(t_i \to t_j)} = \overline{L} + \frac{data_{i,j}}{\overline{B}} \tag{1}$$

where $\overline{B}$ is the average bandwidth among all processor pairs and $\overline{L}$ is the average latency. This simplification is commonly considered to label the edges of the graph to allow for the computation of a priority rank before assigning tasks to processors [22].

Due to heterogeneity, each task may have a different execution time on each processor. Then, $ET(t_i, p_j)$ represents the Execution Time to complete task t_i on processor p_j in available processors set P. The average execution time of task t_i is defined as:

$$\overline{ET}(t_i) = \frac{\sum_{p_j \in P} ET(t_i, p_j)}{|P|} \tag{2}$$

where $|P|$ denotes the number of resources in processor set P.

In a given DAG, a task with no predecessors is called an *entry task* and a task with no successors is called an *exit task*. We assume that the DAG has exactly one entry task t_{entry} and one exit task t_{exit}. If a DAG has multiple entry or exit tasks, a dummy entry or exit task with zero weight and zero communication edges is added to the graph.

In addition to these definitions, we next present some of the common attributes used in task scheduling, which will be used in the following sections.

$pred(t_i)$ and $succ(t_i)$ denote the set of immediate predecessors and immediate successors of task t_i, respectively. $FT(t_i)$ is defined as the Finish Time of task t_i on the processor assigned by the scheduling algorithm.

Schedule length or *makespan* denotes the finish time of the last task of the workflow and is defined as $makespan = FT(t_{exit})$.

$EST(t_i, p_j)$ and $EFT(t_i, p_j)$: denotes Earliest Start Time (EST) and the Earliest Finish Time (EFT) of a task t_i on processor p_j, respectively, and are defined as:

$$EST(t_i, p_j) = \max\Big\{T_{Available}(p_j), \max_{t_{parent} \in pred(t_i)} \{AFT(t_{parent}) + C_{(t_{parent} \to t_i)}\}\Big\} \tag{3}$$

$$EFT(t_i, p_j) = EST(t_i, p_j) + ET(t_i, p_j) \tag{4}$$

where $T_{Available}(p_j)$ is the earliest time at which processor p_j is ready. The inner max block in the EST equation is the time at which all data needed by t_i arrive at the processor p_j. The communication time $C_{(t_{parent} \to t_i)}$ is zero if the predecessor node t_{parent} is assigned to processor p_j. For the entry task, $EST(t_{entry}, p_j) = 0$. Then, to calculate EFT, the execution time of task t_i on processor p_j (ET) is added to its Earliest Start Time.

The financial cost $Cost(t_i, p_j)$ of executing task t_i on specific processor p_j during the time span of $ET(t_i, p_j)$ is the sum of three cost components:

$$Cost(t_i, p_j) = EC(t_i, p_j) + TC(t_i) + SC(t_i) \tag{5}$$

where $EC(t_i, p_j)$ denotes the cost of running task t_i on processor p_j and is defined as $EC(t_i, p_j) = ET(t_i, p_j) \times Price(p_j)$ where $Price(p_j)$ denotes the processor price per time unit. $TC(t_i)$ denotes the cost of transferring the data required for task t_i. In addition, $SC(t_i)$ denotes the data storage cost of task t_i. These cost components are determined by the target platform infrastructure.

$TotalCost$ is the overall cost for executing an application and is defined as:

$$TotalCost = \sum_{t_i \in T} AC(t_i) \tag{6}$$

where $AC(t_i)$ is defined as Assigned Cost of task t_i. After assigning a processor p_{sel} to execute task t_i, the assigned cost value is equal to $AC(t_i) = Cost(t_i, p_{sel})$. In the case of an intra-cluster data transfer, zero monetary costs for communications between tasks are considered, i.e $TC(t_i) = 0$. We also considered zero cost for task storage usage, $SC(t_i) = 0$, as this factor is common to all algorithms and does not influence the comparison of results.

3.3. Scheduling problem

For a given workflow G, a scheduling is defined by a function $sched_G : T \rightarrow P$ which assigns to each task $t_i \in T$ a processor $p_j \in P$, subject to:

- Processor constraint so that no processor executes more than one task at the same time;
- Precedence constraint represented by the edges of the workflow G;
- Budget constraint defined as:

$$\sum_{t_i \in T} AC(t_i) \leq BUDGET \tag{7}$$

- Deadline constraint defined as:

$$Makespan_G \leq DEADLINE \tag{8}$$

The scheduling consists in finding a schedule map between tasks and resources that can meet user-defined QoS constraints, i.e., the total execution time of the workflow must not be larger than the user defined deadline, and the total cost must not be larger than the user defined budget. These values of $DEADLINE$ and $BUDGET$ should be negotiated between users and providers in a range of feasibility values so that there are feasible scheduling solutions. Note that there is no optimization function in the formulation, so that any scheduling solution that matches the constraints is a plausible solution.

4. Proposed Deadline-Budget Constrained Scheduling Algorithm

In this section, we present the Deadline-Budget Constrained Scheduling algorithm (DBCS), which aims to find a feasible schedule within a budget and deadline constraints. The DBCS algorithm is a heuristic strategy that in a single step obtains a schedule that always accomplishes the budget constraint and that may or may not accomplish the deadline constraint. If the time constraint is met, we have a successful schedule; otherwise, we have a failure, and no schedule is produced. The algorithm is evaluated based on the success rate. Before the description of the DBCS algorithm, we next present the attributes used in the algorithm:

- t_{curr} denotes the current task to be scheduled, selected on the task selection phase among all ready tasks;
- $FT_{min}(t_{curr})$ and $FT_{max}(t_{curr})$ denote the minimum and maximum finish time of current tasks among all available processors. Idle slots in the processor schedule, which can accommodate the current task, are also considered;
- $Cost_{min}(t_{curr})$ and $Cost_{max}(t_{curr})$ denote the minimum and maximum execution cost of the current task among all available processors on the target platform;
- $Cost_{best}(t_{curr})$ is the execution cost of the current task on the processor that obtains the lowest finish time among all available processors;
- Δ_{Cost} represents the spare budget defined as the difference between unconsumed budget and cheapest cost assignment for unscheduled tasks. The initial value is $\Delta_{Cost} = BUDGET_{user} - Cost_{cheapest}$ where $BUDGET_{user}$ is the user defined budget as maximum allowed cost and $Cost_{cheapest}$, defined as $Cost_{cheapest} = \sum_{t_i \in T} Cost_{min}(t_i)$, is the cost of the cheapest assignment and represents the cost lower bound for executing the application. Δ_{Cost} is updated at each step after selecting the processor for the current task t_{curr}, as shown in eq (9):

$$\Delta_{Cost} = \Delta_{Cost} - \Big[AC(t_{curr}) - Cost_{min}(t_{curr})\Big] \tag{9}$$

The DBCS, as a list scheduling algorithm, consists of two phases, namely, a *task selection* phase and a *processor selection* phase as described next.

4.1. Task Selection

Tasks are selected according to their priorities. To assign a priority to a task in the DAG, the upward rank ($rank_u$) [22] is computed. This rank represents, for a task t_i, the length of the longest path from task t_i to the exit node(t_{exit}), including the computational time of t_i, and it is given by Eq.10:

$$rank_u(t_i) = \overline{ET}(t_i) + \max_{t_{child} \in succ(t_i)} \Big\{ \overline{C}_{t_i \to t_{child}} + rank_u(t_{child}) \Big\} \tag{10}$$

where $\overline{ET}(t_i)$ is the average execution time of task t_i over all resources, $\overline{C}_{t_i \to t_{child}}$ is the average communication time between two tasks t_i and t_{child}, and $succ(t_i)$ are the set of immediate successor tasks of task t_i. To prioritize tasks, it is common to consider average values because they have to be prioritized before knowing the location where they will run. For the exit node, $rank_u(t_{exit}) = \overline{ET}(t_{exit})$.

4.2. Processor Selection

The processor to be selected to execute the current task is guided by the following quantities related to cost and time. To control the consumed cost and time, a limit value for each factor is needed. We define two variables, CL and DL as limits for cost and time. To select the best suitable processor, a trade-off between these two variables is evaluated. In the following paragraphs, we describe these two variables in detail.

$CL(t_{curr})$ is the maximum available budget for the current task t_{curr} that can be consumed by its assignment, and it is defined as the minimum cost for t_{curr} plus the spare budget available:

$$CL(t_{curr}) = Cost_{min}(t_{curr}) + \Delta_{Cost} \tag{11}$$

All available processors are filtered by $CL(t_{curr})$ to guarantee that the application can be executed without exceeding the budget constraint. We defined this filtered processor set as admissible processors, $P_{admissible}$. In the most restricted case, only the cheapest processors are considered. Otherwise, no feasible schedule exists under the user defined budget.

$DL(t_{curr})$ is defined as the sub-DeadLine that is assigned to each task based on the total application deadline. There are some studies that proposed different strategies to distribute workflow deadlines among tasks. In [25], tasks are grouped in different levels based on their depth in the graph, and then the final deadline is divided into levels in such a way that all tasks belonging to the same level have the same sub-deadline. In [26], first the original workflow is partitioned into sub-workflows, and then the total deadline is divided among partitions. In this paper, we apply the common and direct project planning sub-deadline distribution strategy. The sub-deadline value for each task t_i is computed recursively by traversing the task graph upwards, starting from the exit task. Due to heterogeneity, sub-Deadline can be defined in several different forms. Here, we consider the minimum execution time of the current task, as shown by Eq.12:

$$DL(t_{curr}) = \min_{t_{child} \in succ(t_{curr})} \left[DL(t_{child}) - \overline{C}_{(t_{curr} \rightarrow t_{child})} - ET_{min}(t_{child}) \right] \tag{12}$$

where ET_{min} is defined as the minimum execution time of task t_{curr} among available processors. For the exit task, the sub-deadline is equal to the user defined deadline, $DL(t_{exit}) = DEADLINE$.

Unlike the cost limit, the sub-Deadline is a soft limit as in most deadline distribution strategies on grid platforms with a fixed number of available resources [27]; if the scheduler cannot find a processor that satisfies the sub-deadline for the current task, the processor that can finish the current task at the earliest time is selected.

The processor selection phase is based on the combination of the two QoS factors, time and cost, to obtain the best balance between time and cost minimum values. We define two relative quantities, namely, Time Quality ($Time_Q$) and Cost Quality ($Cost_Q$), for current task t_{curr} on each admissible processor $p_j \in P_{admissible}$, shown in (13) and (14), respectively. Both quantities are normalized by their maximum values.

$$Time_Q(t_{curr}, p_j) = \frac{\Omega \times DL(t_{curr}) - FT(t_{curr}, p_j)}{FT_{max}(t_{curr}) - FT_{min}(t_{curr})} \tag{13}$$

$$Cost_Q(t_{curr}, p_j) = \frac{Cost_{best}(t_{curr}) - Cost(t_{curr}, p_j)}{Cost_{max}(t_{curr}) - Cost_{min}(t_{curr})} \times \Omega \tag{14}$$

where

$$\Omega = \begin{cases} 1 & \text{if} \quad FT(t_{curr}, p_j) < DL(t_{curr}) \\ 0 & otherwise \end{cases} \tag{15}$$

$Time_Q$ measures how much closer to the task sub-deadline (DL) the finish time of the current task on processor p_j is. Processors with higher $Time_Q$ values have a greater possibility of being selected. If the current task has a higher finish time on processor p_j than its sub-deadline, $Time_Q$ assumes a negative value for p_j, reducing the possibility of this processor being selected.

Similarly, $Cost_Q$ measures how much less the actual cost on p_j is than the cost on the processor that results in the earliest finish time ($Cost_{best}$). Although CL, eq(11), is the maximum allowed cost for the current task, here, $Cost_{best}$ is used to avoid selecting a processor that performs worse and costs more than the processor that guarantees the earliest finish time.

In the case that none of the processors from $P_{admissible}$ can guarantee t_{curr} sub-deadline, $Cost_Q$ is zero for all of them, and $Time_Q$ for each processor p_j is a negative value that represents the relative finish time obtained with

p_j. The processor from $P_{admissible}$ with higher $Time_Q$, i.e., closer to zero, would be selected. Note that, in any case, cost will be lower than CL, the maximum available budget for the current task.

Finally, to select the most suitable processor for the current task, the Quality measure (Q) for each processor $p_j \in P_{admissible}$ is computed as shown in Eq(16):

$$\begin{aligned} Q(t_{curr}, p_j) = & Time_Q(t_{curr}, p_j) + \\ & Cost_Q(t_{curr}, p_j) \times \frac{Cost_{Cheapest}}{Budget_{Unconsumed}} \end{aligned} \tag{16}$$

where the cost quality factor is weighted by the ratio of the cheapest cost execution for unscheduled tasks over the unconsumed budget, so that the effectiveness of the cost quality factor can be controlled. A higher value of the fraction means that the unconsumed budget is close to the cheapest cost execution for unscheduled tasks, so that the cost factor is more predominant in the processor Quality measure. In the same way, a lower value means a higher difference between unconsumed budget and cheapest cost execution for unscheduled tasks, so that the cost factor is less influential, allowing the selection of more expensive processors that guarantee a lower processing time for t_{curr}.

The DBCS algorithm is shown in Algorithm 1. First, the possibility of finding a schedule map under a user defined budget is checked in lines 1-3. After some initializations in lines 4-5, the algorithm starts to map all tasks of the application (while looping in lines 6-14). At each step, on line 7, among all ready tasks, the task with highest priority ($rank_u$) is selected as the current task (t_{curr}). Then, in lines 8-10, the Quality measure for assigning t_{curr} to processor p_j ($Q(t_{curr}, p_j)$) is calculated. Note that, first, the finish time (FT) and execution cost of the current task is calculated and then the quality measure for all *admissible* processors is calculated. Next, the processor with the highest quality measure among all processors is selected (line 11-12). Finally, after assigning the processor to the current task, the Δ_{Cost} variable is updated using Eq.9 (line 13).

In terms of time complexity, DBCS requires the computation of the upward rank ($rank_u$) and sub-DeadLines (DL) for each task that have complexity $O(n.p)$, where p is the number of available resources and n is the number of tasks in the workflow application. In the processor selection phase, to

Algorithm 1 DBCS algorithm

```
Require: a DAG and user's QoS Parameters values for time ($DEADLINE_{user}$) and cost ($BUDGET_{user}$)
 1: if $BUDGET_{user} < Cost_{cheapest}$ then
 2:     return no possible schedule map
 3: end if
 4: Initialize $\Delta_{Cost} = BUDGET_{user} - Cost_{cheapest}$
 5: Compute the upward rank ($rank_u$) and sub-DeadLine value ($DL$) for each task
 6: while there is an unscheduled task do
 7:     $t_{curr}$ = the next ready task with highest $rank_u$ value
 8:     for all $p_j \in P_{admissible}$ do
 9:         calculate Quality measure $Q(t_{curr}, p_j)$ using Eq.16
10:     end for
11:     $P_{sel}$ = Processor $p_j$ with highest Quality measure ($Q$)
12:     Assign current task $t_{curr}$ to Processor $P_{sel}$
13:     Update $\Delta_{Cost}$ using Eq.(9)
14: end while
15: return Schedule Map
```

find and assign a suitable processor for the current task, the complexity is $O(n.p)$ for calculating FT and $Cost$ for the current task among all processors, plus $O(p)$ for calculating the Quality measure. The total time is $O(n.p + n(n.p + p))$, where the total algorithm complexity is of the order $O(n^2.p)$.

5. Experimental Results

This section presents performance comparisons of the DBCS algorithm with DCA [17], LOSS1 [18], GA [24], BHEFT [31, 32] and RANDOM scheduling algorithms.

We consider synthetic randomly generated and Real Application workflows to evaluate a wider range of loads. The results presented were produced with SimGrid [8], which is one of the simulators for distributed computing and allows for a realistic description of the infrastructure parameters.

5.1. Workflow Structure

To evaluate the relative performances of the algorithms, both the randomly generated and real-world application workflows were used; namely, MONTAGE and EPIGENOMICS [4] are used. The randomly generated workflows were created by the synthetic DAG generation program[3]. The computational complexity of a task is modelled as one of the three following forms, which are representative of many common applications: $a.d$ (e.g., image processing of a $\sqrt{d} \times \sqrt{d}$ image), $a.d\log d$ (e.g., sorting an array of d elements), $d^{3/2}$ (e.g., multiplication of $\sqrt{d} \times \sqrt{d}$ matrices) where a is picked randomly between 2^6 and 2^9. As a result, different tasks exhibit different communication/computation ratios.

The DAG generator program defines the DAG shape based on four parameters: *width*, *regularity*, *density*, and *jumps*. The width determines the maximum number of tasks that can be executed concurrently. A small value will lead to a thin DAG, similar to a chain, with low task parallelism, and a large value induces a fat DAG, similar to a fork-join, with a high degree of parallelism. The regularity indicates the uniformity of the number of tasks in each level. A low value means that the levels contain very dissimilar numbers of tasks, whereas a high value means that all levels contain similar numbers

[3]https://github.com/frs69wq/daggen

of tasks. The density denotes the number of edges between two levels of the DAG, where a low value indicates few edges and a large value indicates many edges. A jump indicates that an edge can go from level l to level $l + jump$. A jump of one is an ordinary connection between two consecutive levels.

In our experiment, for random DAG generation, it was considered as the number of tasks $n = [30, 70, 90]$, $jump = [1, 2, 3]$, $fat = [0.2, 0.4, 0.8]$, $regularity = [0.2, 0.8]$, and $density = [0.2, 0.8]$. With these parameters, each DAG is created by picking the value for each parameter randomly from the parameter data set. The total number of DAGs generated in our simulation is 10000.

5.2. Simulation Platform

We resorted to simulation to evaluate the algorithms discussed in the previous sections. Simulation allows us to perform a statistically significant number of experiments for a wide range of application configurations in a reasonable amount of time. We used the SimGrid toolkit[4] [8] as the basis for our simulator. SimGrid provides the required fundamental abstractions for the discrete-event simulation of parallel applications in distributed environments. It was specifically designed for the evaluation of scheduling algorithms. Relying on a well-established simulation toolkit allows us to leverage sound models of a heterogeneous computing system, such as the grid platform considered in this work.

We consider three sites that comprise multiple clusters and have different CPU power compositions. The Rennes site has normal distribution of CPU power among its clusters, and the Sophia site contains a higher number of low-speed processors, while the Lille site has a higher number of fast processors.

To normalize diverse price units for the heterogeneous processors, as defined in [31], the price of a processor $p_j \in P$ is assumed to be $Price(p_j) = \alpha_{p_j}(1 + \alpha_{p_j})/2$, where α_{p_j} is the ratio of p_j processing capacity to that of the fastest processor of the set P. The price will be in the range of $]0 \cdots 1]$, where the fastest processor, with the highest power, has a price value equal to 1.

Table 1 provides the name of each site, along with the set of clusters that compose the site. For each cluster, it presents the total number of processors ($\#CPU_{Total}$), processing speed expressed in GFlop/s and processor cost.

[4]http://simgrid.gforge.inria.fr

$\#CPU_{used}$ shows the number of processors used from each cluster for 8-, 16- and 32-processor configurations.

Site	Cluster	#CPU$_{Total}$	#CPU$_{used}$			Power(GFlop/s)	Cost($)
`rennes`	`paradent`	64	3	7	13	21.496e9	0.61$
	`paramount`	33	2	3	6	12.910e9	0.31$
	`parapide`	25	1	2	4	30.130e9	1.00$
	`parapluie`	40	2	4	9	27.391e9	0.87$
`sophia`	`helios`	56	3	6	12	7.7318E9	0.16$
	`sol`	50	3	5	10	8.9388e9	0.19$
	`suno`	45	2	5	10	23.530e9	0.70$
`lille`	`chicon`	26	2	4	9	8.9618e9	0.19$
	`chimint`	20	2	4	7	23.531e9	0.70$
	`chinqchint`	46	4	8	16	22.270e9	0.64$

Table 1: Description of the Grid5000 clusters from which the platforms used in the experiments were derived

5.3. Budget and Deadline parameters

To evaluate the DBCS algorithm, in our simulation, we need to define a value for time and cost as DEADLINE and BUDGET constraint parameters. For each site, these parameters are computed independently of the number of CPUs on that site. Additionally, to have better performance analysis, the DEADLINE and BUDGET values are calculated among all possible sites in our tested platform, i.e., all resources in the three possible sites `rennes`, `sophia` and `lille` are considered regardless of the site at which the DAG is to be executed.

To specify the deadline parameter, we define the min_{time} and max_{time} as the lowest and highest execution time of the application as shown in Eq.17 and Eq.18.

$$min_{time} = \sum_{t_i \in CP} \left(ET^*_{min}(t_i) + \overline{C}_{(t_{parent_{CP}} \to t_i)} \right) \tag{17}$$

$$max_{time} = \sum_{t_i \in CP} \left(ET^*_{max}(t_i) + \overline{C}_{(t_{parent_{CP}} \to t_i)} \right) \tag{18}$$

where CP is the set of tasks belonging to the critical path, $t_{parent_{CP}}$ is the critical parent of task t_i and $\overline{C}$ is the average communication time between task t_i and its critical parent. $ET^*_{min}(t_i)$ and $ET^*_{max}(t_i)$ are defined as the minimum and the maximum execution time for task t_i on the fastest and the slowest processor among all sites. In our tested sites, the slowest and fastest processors, $P_{fastest}$ and $P_{slowest}$, belong to cluster `parapide` and `helios`, respectively. Please note that both of these values are defined as the lowest and highest possible makespans based on an infinite number of CPUs. For a bounded number of resources, the minimum and maximum execution time may be very optimistic and not reachable. The processing time range is defined based on the critical path to specify a deadline based on the lower bound of the makespan.

Similarly, to specify a budget constraint, we need to estimate the maximum and the minimum cost to obtain a range of feasible budgets to execute the application. We defined the max_{cost} and min_{cost} as the absolute highest and lowest possible costs for executing the application, which are calculated by summing the maximum and the minimum execution costs for each task, respectively, among all resources in all sites.

With these highest and lowest bound values, we define for the current application a unique DEADLINE and BUDGET constraint, independently of the execution site, as described by Eq (19) and Eq 20):

$$DEADLINE_{user} = min_{time} + \alpha_D \times (max_{time} - min_{time}) \tag{19}$$

$$BUDGET_{user} = min_{cost} + \alpha_B \times (max_{cost} - min_{cost}) \tag{20}$$

where the deadline parameter α_D and budget parameter α_B can be selected in the range of $[0 \dots 1]$.

5.4. Performance Metric

To evaluate and compare our algorithm with other approaches, we consider the Planning Successful Rate (PSR), as expressed by Eq (21). This metric provides the percentage of valid schedules obtained in a given experiment.

$$\text{PSR} = 100 \times \frac{\textit{Successful Planning}}{\textit{Total Number in experiment}} \tag{21}$$

5.5. Results and Discussion

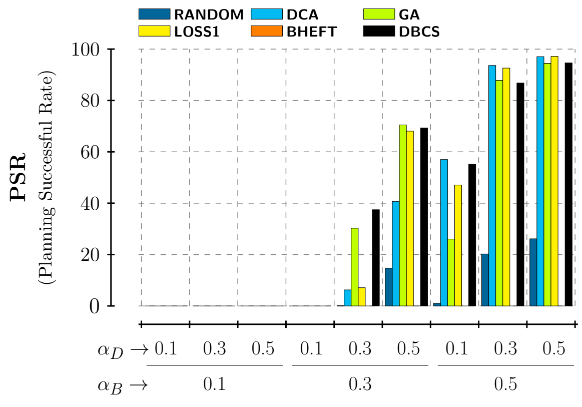
As we selected GA, DCA, LOSS1 and BHEFT algorithms for comparison, we first describe the adaptations considered from their original strategies. A RANDOM scheduling algorithm is also considered.

The original implementation of the LOSS1 scheduling algorithm assumed that all of the processors had different costs, and therefore, there was no conflict in selecting a processor based on the cost parameter. In our heterogeneous environment, each cluster is homogeneous, so there could be more than one processor candidate. In this case, we test all of the possible processors and selected the one that achieves the earliest finish time. Also, to apply the time and cost constraint parameters in the algorithm, the loop exit point to test and generate a schedule map changed to cover QoS parameters time and cost. For the GA scheduling algorithm, the default configuration used for producing results is: population size equals 300, swapping mutation, replacing mutation probability equal to 0.5 and a generation limit of 100. For the DCA algorithm, we considered 100 cells for the memorization table and a maximum of 10 intermediate solutions per cell. In both algorithms, as stated before, we stop the solution search after obtaining the first feasible solution for the scheduling problem in our implementation. Undoubtedly, increasing the configuration parameters for the GA and DCA algorithms, we would be able to achieve higher successful percentage rates, but it would increase the execution time of the algorithms exponentially. Here, we use the same configurations as in their original papers. For BHEFT, we stop the makespan optimization constrained to the defined BUDGET, as the current makespan is lower than the defined DEADLINE. The RANDOM scheduling strategy uses the task selection phase, as described in 4.1, to select the current task but selects the processor randomly for each task.

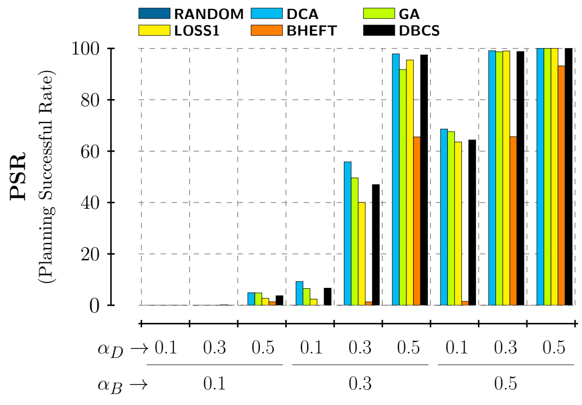
5.5.1. Results for Randomly Generated Workflows

For randomly generated workflows, we model the computational complexity of common application tasks, such as image processing, array sorting, and matrix multiplication. To observe the ability of finding valid schedule maps, we selected a low set of values, $\{0.1, 0.3, 0.5\}$, for time and cost parameters (α_D and α_B) to test the performance of each algorithm on harder conditions.

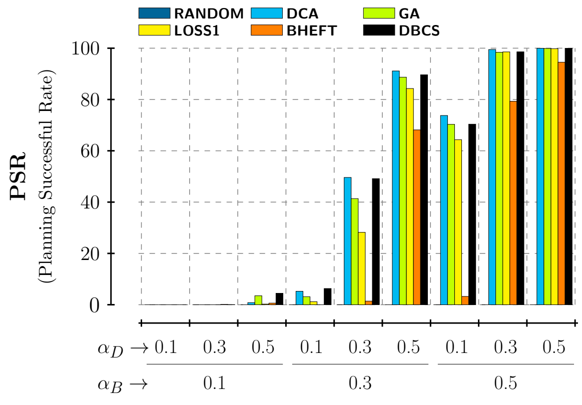
Figure 1 shows the average Planning Successful Rate (PSR) obtained on three different sites with CPU configurations per site equal to 8, 16 and 32 processors. The main result is that the algorithm DBCS obtains similar performance to other state-of-the-art search-based algorithms for the range of budget and deadline values considered here. For the Sophia and Lille sites, DBCS is very close to DCA performance and better than GA and LOSS1 for most of the cases. In addition, DBCS always obtains better results than BHEFT, which presents very low PSR values for the most restricted cases.



(a) Cluster$_{name}$ = Rennes



(b) Cluster$_{name}$ = Sophia



(c) Cluster$_{name}$ = Lille

Figure 1: Planning Success Rate for Random workflows

On the contrary, even on the most restricted cases, DBCS always obtains PSR values close to the DCA results. Due to the definition of the budget that is based on the cheapest processor across all sites, the lower budget values are too restrictive on the `Rennes` site, as its cheapest processor cost about twice the $P_{slowest}$. Therefore, the PSR values in `Rennes` are lower than in the other two sites. DBCS achieves similar PSR values as the best higher-time complexity algorithms, which in some cases is DCA and in other cases is GA. On the `Rennes` site, BHEFT did not produce valid schedules. The RANDOM scheduling strategy gives the lowest PSR values, which were near zero.

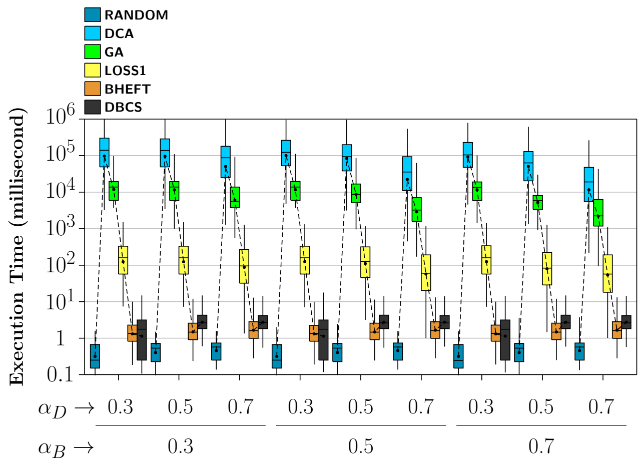


Figure 2: Execution time for each algorithm to find a valid solution (log scale)

As a heuristic algorithm, the main advantage of the DBCS consists in having an execution time in the range of the heuristic algorithms, such as BHEFT and RANDOM, but a planning success rate similar to the higher-time complexity search-based algorithms. Figure 2 shows, in logarithmic scale, the processing time of each of the algorithms to find a valid solution. We can observe that the algorithm DBCS requires an average of 4 milliseconds to obtain a valid solution, whereas other algorithms take substantially more time on average, DCA being the most expensive, with an average value of approximately 100 seconds, followed by GA with approximately 10 seconds and LOSS1 with approximately 100 milliseconds. Compared to those algorithms, DBCS produces schedules in a fraction of the time, which ranges from 0.004% to 4%. The three heuristic algorithms, DBCS, BHEFT and

RANDOM, have a time complexity of $O(n^2.p)$ for n nodes and p processors and, therefore, obtain an execution time in the same range, below 10 milliseconds. However, DBCS achieves higher planning success rates than both of them, as shown above.

5.5.2. Results for Real World Applications

To evaluate the algorithms on a standard and real set of workflow applications, a set of workflows were generated using the code developed in Pegasus toolkit[5]. Two well-known structures were chosen [13], namely: MONTAGE and EPIGENOMICS. MONTAGE is an I/O-intensive workflow, and EPIGENOMICS is a compute-intensive workflow. For each of the MONTAGE and EPIGENOMICS workflows, we generated 300 DAGs with a number of tasks equal to 46 and 50, respectively. Additionally, we consider the application Wien2K [5], which is a material science workflow for performing electronic structure calculations of solids that contains two parallel sections with sequential synchronization activities in between. In this experiment, we use Wien2k workflows with 30 tasks that were obtained from trace data collected from historical executions conducted in the Austrian Grid[6]. To complement the characterization of these real world applications, Figure 3 shows the Communication to Computation Ratio (CCR) associated with each application. Once a deadline is specified based on the workflow critical path, if the workflow has higher CCR distribution or higher average CCR value, it will decrease the success rate of that workflow.

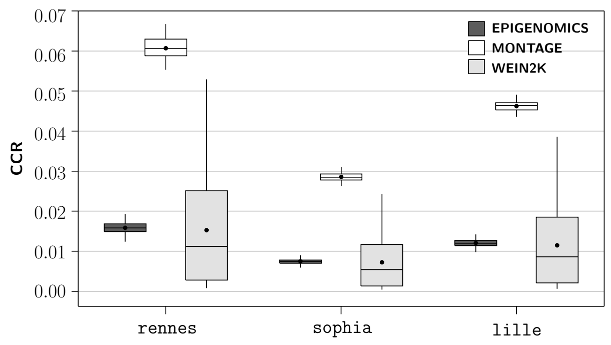
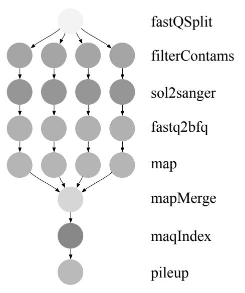


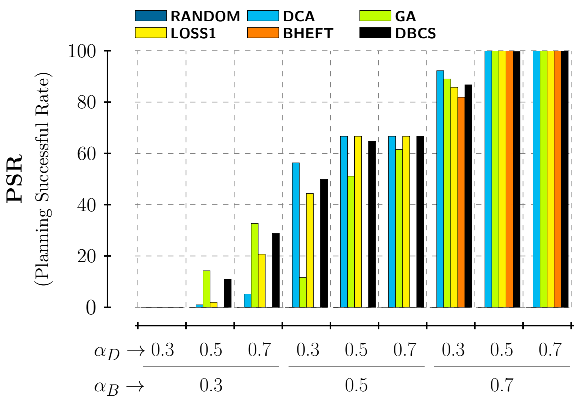
Figure 3: CCR values for each workflow on different sites

[5]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
[6]http://www.austriangrid.at

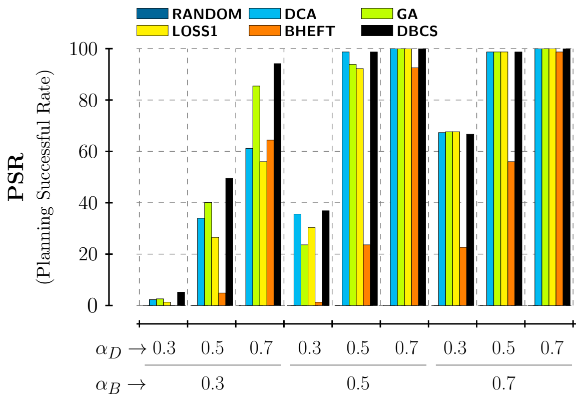
We consider the following values for time and cost parameters: $\alpha_D, \alpha_B \in \{0.3, 0.5, 0.7\}$. Figure 4, 5 and 6 shows the PSR for each real application and for three different sites.



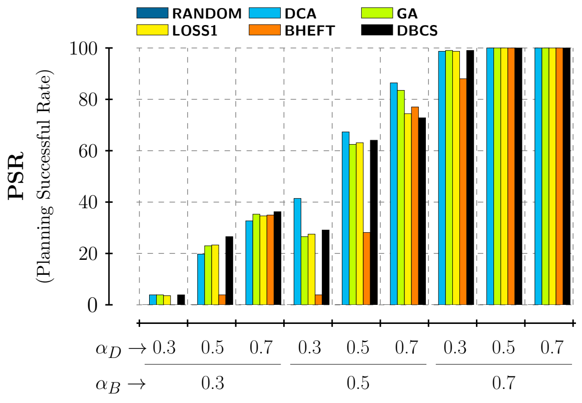
(a) EPIGENOMICS workflow



(b) EPIGENOMICS, Cluster$_{name}$ = Rennes

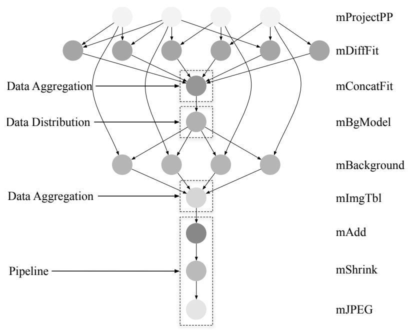


(c) EPIGENOMICS, Cluster$_{name}$ = Sophia

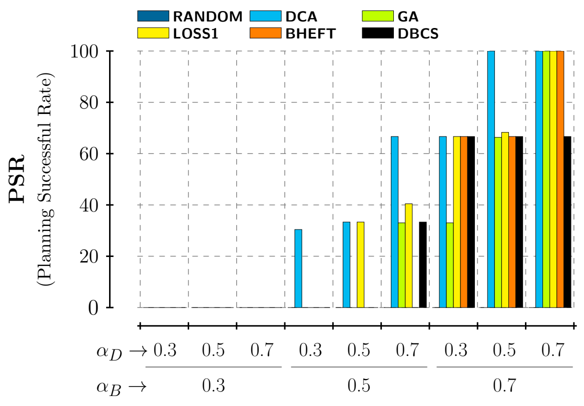


(d) EPIGENOMICS, Cluster$_{name}$ = Lille

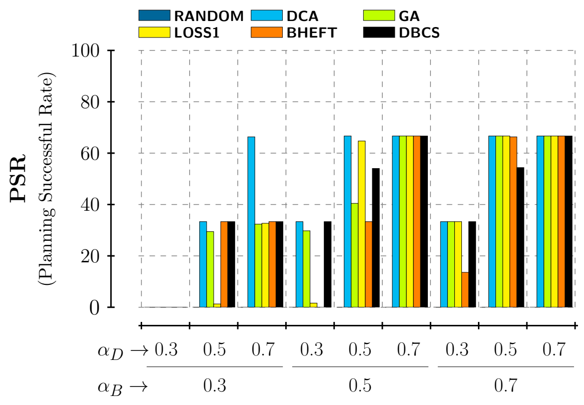
Figure 4: Normalized Makespan for EPIGENOMICS workflows on GRID5000



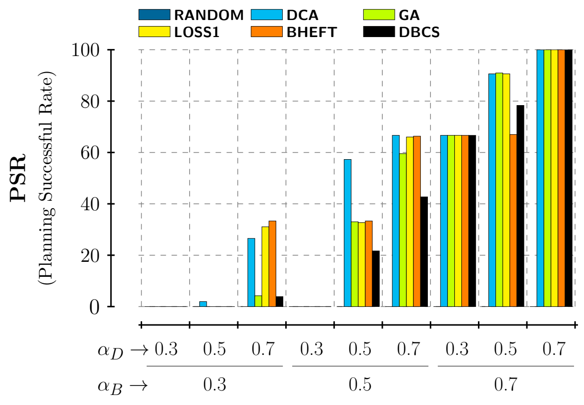
(a) MONTAGE workflow



(b) MONTAGE, Cluster$_{name}$ = Rennes

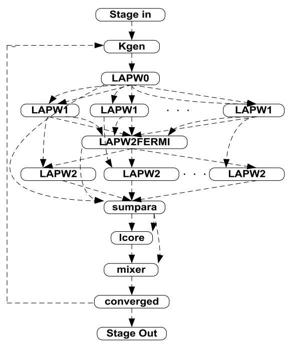


(c) MONTAGE, Cluster$_{name}$ = Sophia

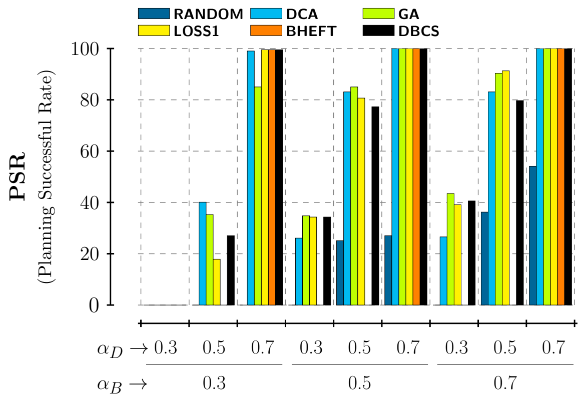


(d) MONTAGE, Cluster$_{name}$ = Lille

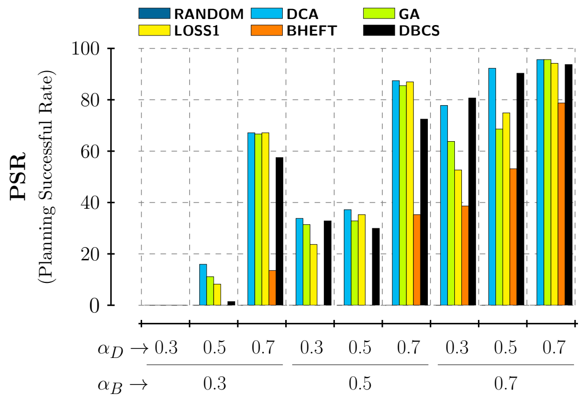
Figure 5: Normalized Makespan for MONTAGE workflows on GRID5000



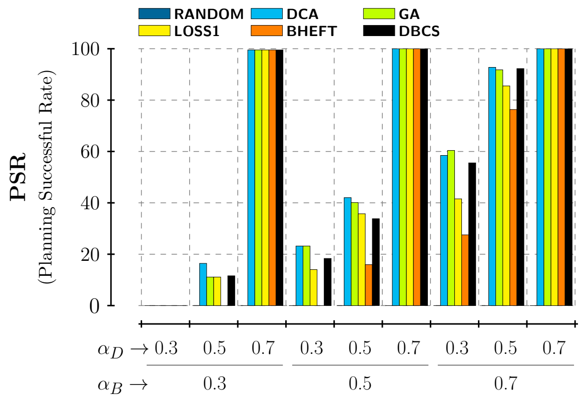
(a) Wien2k workflow



(b) Wien2k, Cluster_{name} = Rennes



(c) Wien2k, Cluster_{name} = Sophia



(d) Wien2k, Cluster_{name} = Lille

Figure 6: Normalized Makespan for Wien2K workflows on GRID5000

As the results for random graph applications indicate, none of the algorithms always performs better for all sites and applications. For the EPIGENOMICS workflow, Figures 4.b, 4.c and 4.d, DBCS obtained similar performances to other higher complexity algorithms, always achieving the best result for the `Sophia` site. For the `Lille` site, the results are similar to `Sophia`, DBCS being the best algorithm for the lower budget ($\alpha_B = 0.3$) and producing slightly lower performance than DCA for medium and larger budgets. DBCS is always better than BHEFT, except for a single case in `Lille`, with significantly better performance in most configurations of the deadline and budget. For the MONTAGE workflow, Figures 5.b, 5.c and 5.d, the DCA algorithm consistently obtained higher PSR values, but DBCS obtained comparable PSR values for higher budgets. This is explained by the fact that MONTAGE has a higher average CCR; thus, the deadline imposed, based on the critical path, is too restrictive, in particular on `Rennes`. This only affects DBCS, BHEFT and RANDOM because due to their heuristic nature, a moving forward strategy, they do not roll back and change the task assignment. The other algorithms that are search-based strategies may change the task assignment during the solution space search. BHEFT also obtains good performances, comparable to DBCS, in particular for `Sophia` and `Lille`. For the Wien2K algorithm, Figures 6.b, 6.c and 6.d, DCA presents generally better performances, but DBCS achieves the same value for many cases and a PSR value very close to the best algorithm that alternates from DCA, GA and LOSS1. In `Rennes` site, BHEFT produces results only for the higher deadline ($\alpha_D = 0.7$), while DBCS produces very good PSR values for all cases as well as DCA, GA and LOSS1. In `Sophia`, the results are similar to `Rennes` but with lower PSR values. BHEFT produces results for more cases but significantly lower than DBCS. In `Lille`, the results follow the same pattern as in `Rennes` with a PSR value of 100% for all algorithms, except RANDOM in the highest deadline. In the other cases, DBCS is close to the best algorithms, which are DCA and GA, and significantly better than BHEFT.

EPIGENOMICS workflow has the highest rate of successful schedules due to its lower average CCR as well as lower CCR distribution values, which makes it more feasible to find schedule maps for the range of budgets and deadlines. The RANDOM algorithm only produces valid schedules for Wien2K in `Rennes`, with a much lower performance than DBCS, showing that the slightly higher processing time of DBCS compensates. The same conclusion can be drawn in relation to BHEFT, where with a similar scheduling time, DBCS obtains globally better performances.

6. Conclusions and Future Work

In this paper, we presented the Deadline-Budget Constrained Scheduling algorithm (DBCS), which maps a workflow to a heterogeneous system constrained to user-defined deadline and budget values. The algorithm was compared with other state-of-the-art algorithms. In terms of time complexity, which is a critical factor for effective usage on real platforms, our algorithm has the lowest time complexity (quadratic time complexity), while other algorithms mostly have cubic or polynomial time complexities. In terms of the quality of results, DBCS achieves rates of successful schedules similar to higher-time complexity algorithms for both random and real application workflows on diverse platforms and also for the range of values of deadline and budget constraints considered in this paper. Compared to other low-time complexity algorithms, namely, BHEFT and RANDOM, DBCS performs, in general, significantly better for the workflows and platforms considered.

In conclusion, we have presented the DBCS algorithm for budget and deadline constrained scheduling, which has proved to achieve a similar performance to higher-time complexity algorithms, namely, DCA, GA and LOSS1, but with a time complexity of the heuristic algorithms, namely, BHEFT and RANDOM, of the order $O(n^2.p)$ for n tasks and p processors.

In future work, we intend to extend the algorithm to consider the dynamic concurrent DAG scheduling problem. This model will allow users to execute concurrent workflows that might not be able to start together but that can share resources so that the total time and cost for the user can be minimized to meet their deadlines and budgets.

Acknowledgments

This work is partially supported by Fundação para a Ciência e Tecnologia, PhD Grant FCT - DFRH - SFRH/BD/80061/2011, and by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

[1] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[2] H. Arabnejad and J.G. Barbosa. A budget constrained scheduling algorithm for workflow applications. *Journal of Grid Computing*, 2014.

[3] H. Arabnejad and J.G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):682–694, March 2014.

[4] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10. IEEE, 2008.

[5] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. Wien2k: An augmented plane wave plus local orbitals program for calculating crystal properties. Technical report, Institute of Physical and Theoretical Chemistry, TU Vienna, 2001.

[6] J. Broberg, S. Venugopal, and R. Buyya. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[7] Louis-Claude Canon, Emmanuel Jeannot, Rizos Sakellariou, and Wei Zheng. Comparative evaluation of the robustness of dag scheduling heuristics. In *Grid Computing*, pages 73–84. Springer, 2008.

[8] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131. IEEE, 2008.

[9] Wei-Neng Chen and Jun 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*, 39(1):29–43, 2009.

[10] E.G. Coffman and J.L. Bruno. *Computer and job-shop scheduling theory*. John Wiley & Sons, 1976.

[11] A. Doğan and F. Özgüner. Bi-objective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems. *The Computer Journal*, 48(3):300–314, 2005.

[12] Ritu Garg and Awadhesh Kumar Singh. Multi-objective workflow grid scheduling based on discrete particle swarm optimization. In *Swarm, Evolutionary, and Memetic Computing*, pages 183–190. Springer, 2011.

[13] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[14] Gideon Juve, Ewa Deelman, G. Bruce Berriman, Benjamin P. Berman, and Philip Maechling. An evaluation of the cost and performance of scientific workflows on amazon ec2. *Journal of grid computing*, 10(5):5–21, 2012.

[15] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. *Future Generation Computer Systems*, 48:1–18, 2015.

[16] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 49. ACM, 2011.

[17] Radu Prodan and Marek Wieczorek. Bi-criteria scheduling of scientific grid workflows. *Automation Science and Engineering, IEEE Transactions on*, 7(2):364–376, 2010.

[18] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D Dikaiakos. Scheduling workflows with budget constraints. In *Integrated Research in GRID Computing*, pages 189–202. Springer, 2007.

[19] Gurmeet Singh, Carl Kesselman, and Ewa Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *Proceedings of the 16th international symposium on High performance distributed computing*, pages 117–126. ACM, 2007.

[20] Sen Su, Jian Li, Huang Qingjia, Kai Shuang, and Jie Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39:177–188, 2013.

[21] AKM Talukder, Michael Kirley, and Rajkumar Buyya. Multiobjective differential evolution for scheduling workflow applications on global grids. *Concurrency and Computation: Practice and Experience*, 21(13):1742–1756, 2009.

[22] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[23] Fuhui Wu, Qingbo Wu, and Yusong Tan. Workflow scheduling in cloud: a survey. *The Journal of Supercomputing*, pages 1–46, 2015.

[24] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3):217–230, 2006.

[25] Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005.

[26] Jia Yu, Rajkumar Buyya, Chen Khong Tham, et al. Qos-based scheduling of workflow applications on service grids. In *Proc. of 1st IEEE International Conference on e-Science and Grid Computing*, 2005.

[27] Jia Yu, Kotagiri Ramamohanarao, and Rajkumar Buyya. Deadline/budget-based scheduling of workflows on utility grids. *Market-Oriented Grid and Utility Computing*, pages 427–450, 2009.

[28] Yingchun Yuan, XiaoPing Li, and Qian Wang. Dynamic heuristics for time and cost reduction in grid workflows. In *Computer Supported Cooperative Work in Design III*, pages 499–508. Springer, 2007.

[29] Yingchun Yuan, Xiaoping Li, Qian Wang, and Xia Zhu. Deadline division-based heuristic for cost optimization in workflow scheduling. *Information Sciences*, 179(15):2562–2575, 2009.

[30] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li. Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In *Advanced Information Networking and Appli-*

cations (AINA), 2012 IEEE 26th International Conference on, pages 534–541. IEEE, 2012.

[31] Wei Zheng and Rizos Sakellariou. Budget-deadline constrained workflow planning for admission control in market-oriented environments. In *Economics of Grids, Clouds, Systems, and Services*, pages 105–119. Springer, 2012.

[32] Wei Zheng and Rizos Sakellariou. Budget-deadline constrained workflow planning for admission control. *Journal of grid computing*, 11(4):633–651, 2013.

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.