J Grid Computing (2014) 12:665–679
DOI 10.1007/s10723-014-9294-7

A Budget Constrained Scheduling Algorithm for Workflow Applications

Hamid Arabnejad · Jorge G. Barbosa

Received: 18 August 2013 / Accepted: 25 February 2014 / Published online: 23 March 2014

Abstract Service-oriented computing has enabled a new method of service provisioning based on utility computing models, in which users consume services based on their Quality of Service (QoS) requirements. In such pay-per-use models, users are charged for services based on their usage and on the fulfilment of QoS constraints; execution time and cost are two common QoS requirements. Therefore, to produce effective scheduling maps, service pricing must be considered while optimising execution performance. In this paper, we propose a Heterogeneous Budget Constrained Scheduling (HBCS) algorithm that guarantees an execution cost within the user's specified budget and that minimises the execution time of the user's application. The results presented show that our algorithm achieves lower makespans, with a guaranteed cost per application and with a lower time complexity than other budget-constrained state-of-the-art algorithms. The improvements are particularly high for more heterogeneous systems, in which a reduction of 30 % in execution time was achieved while maintaining the same budget level.

H. Arabnejad · J. G. Barbosa (✉)
LIACC, Departamento de Engenharia Informática,
Faculdade de Engenharia, Universidade do Porto,
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
e-mail: jbarbosa@fe.up.pt

H. Arabnejad
e-mail: hamid.arabnejad@fe.up.pt

Keywords Utility computing · Deadline · Quality of Service · Planning Success Rate

1 Introduction

Utility computing is a service provisioning model that provides computing resources and infrastructure management to customers as they need them, as well as a payment model that charges for usage. Recently, 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. Recently, 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 [3]. In this computing model, users consume services and resources when they need them and pay only for what they use. Cost and time have become the two most important 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 [8].

Many complex applications in e-science and e-business can be modelled as workflows [9]. A popular representation of a workflow application is the Directed Acyclic Graph (DAG), in which nodes represent individual application tasks, and the directed edges represent inter-task data dependencies. Many workflow scheduling algorithms have been developed to execute workflow applications. Some typical workflow scheduling algorithms were introduced in [24]. Most of these algorithms have a single objective, such as minimising execution time (makespan). However, additional objectives can be considered when scheduling workflows onto grids, based on the user's QoS requirements. If we consider multiple QoS parameters, such as budgets and deadlines, then the problem becomes more challenging.

The contributions of this paper are as follows: a) a new low time complexity algorithm that obtains higher performance than state-of-the-art algorithms of the same class for the two set-ups considered here, namely i) minimising the makespan for a given budget and ii) budget-deadline constrained scheduling; b) a realistic simulation that considers a bounded multi-port model in which bandwidth is shared by concurrent communications; and c) results for randomly generated graphs, as well as for real-world applications.

The remainder of the paper is organised as follows. Section 2 describes the system model, including the application model, the utility computing model, and the objective function. Section 3 discusses related work on budget-constrained workflow scheduling. The proposed scheduling algorithm (HBCS) is presented in Section 4. Experimental details and simulation results are presented in Section 5. Finally, Section 6 concludes the paper.

2 Problem Definition and System Model

A typical workflow application can be represented by a Directed Acyclic Graph (DAG), which is a directed graph with no cycles. In a DAG, an individual task and its dependencies are represented by a node and its edges, respectively. A dependency ensures that a child node cannot be executed before all of its parent tasks finish successfully and transfer the input data required by the child. The task computation times and communication times are modelled by assigning weights to nodes and edges respectively. A DAG can be modelled by a tuple $G(N, E)$, where N is the set of n nodes, each node $n_i \in N$ represents an application task, and E is the set of communication edges between tasks. Each edge $e(i, j) \in E$ represents a task-dependency constraint such that task n_i should complete its execution before task n_j can start.

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 n_{entry} and one exit task n_{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.

The target utility computing platform is composed of a set of clusters; each cluster has homogeneous processors that have a given capability and cost to execute tasks of a given application. The collection of clusters forms a heterogeneous system. Processors are priced, with the most powerful processor having the highest cost. To normalise diverse price units for the heterogeneous processors, as defined in [27], the price of a processor p_j 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. The price will be in the range of $]0 \ldots 1]$, where the fastest processors, with the highest power, have a price value equal to 1.

For each task n_i, $w_{i,j}$ gives the estimated time to execute task n_i on processor p_j, and $Cost(n_i, p_j) = w_{i,j}.Price(p_j)$ represents the cost of executing task n_i on processor p_j. After assigning a specific processor to execute the task n_i, $AC(n_i)$ is defined as *Assigned Cost* of task n_i. The overall cost for executing an application is defined as $TotalCost = \sum_{n_i \in N} AC(n_i)$.

The edges of the DAG represent a communication cost in terms of time, but they are considered to have zero monetary cost because they occur inside a given site. The schedule length of a DAG, also called Makespan, denotes the finish time of the last task in the scheduled DAG, and is defined by:

$$makespan = max\{AFT(n_{exit})\} \quad (1)$$

where $AFT(n_{exit})$ denotes the *Actual Finish Time* of the exit node. In cases in which there is more than one exit node, and no redundant node is added, the makespan is the maximum actual finish time of all of the exit tasks.

The *objective* of the scheduling problem is to determine an assignment of tasks of a given DAG to processors such that the Makespan is minimised, subject to the budget limitation imposed by the user, as expressed in (2):

$$\sum_{n_i \in N} AC(n_i) \leq BUDGET \tag{2}$$

The user specifies the budget within the range provided by the system, as shown by (3):

$$BUDGET = Cheapest_{Cost} + k_{Budget}\left(Highest_{Cost} - Cheapest_{Cost}\right) \tag{3}$$

where $Highest_{Cost}$ and $Cheapest_{Cost}$ are the costs of the assignments produced by an algorithm that guarantees the minimum processing time, such as HEFT [19], and the least expensive scheduling, respectively. The least expensive assignment is obtained by selecting the least expensive processor to execute each of the workflow tasks. The algorithm HEFT is used here as one algorithm that produces the minimum Makespans for a DAG in a heterogeneous system with complexity $O(v^2.p)$ [5]. Therefore, the lower bound of the execution cost is the minimum cost that can be achieved in the target platform, obtained by the cheapest assignment; the upper bound is the cost of the schedule that produces the minimum Makespan. Finally, the budget range feasible on the selected platform is presented to the user; he/she selects a budget inside that range, represented by k_{Budget} in the range of $[0 \ldots 1]$. This budget definition was first introduced by [15]. The least expensive assignment guarantees that it is always feasible to obtain valid mapping within the user budget, although without guaranteeing the minimisation of the makespan. If the user can afford to pay the highest cost or is limited to the least expensive cost, then the schedule is defined by the HEFT or by the least expensive assignment, respectively. Between these limits, the algorithm we propose here can be used to produce the assignment.

In conclusion, the scheduling problem described in this paper is a single objective function, in which only processing time is optimised, and cost is a problem constraint, the value of which must be guaranteed by the scheduler. This feature is very relevant for users within the context of the utility computing model, and it is a distinguishing feature compared to other algorithms that optimise cost without guaranteeing a user-defined upper bound, as described in the next section.

3 Related Work

Generally, the related research in this area can be classified into two main categories: *QoS optimisation* scheduling and *QoS constrained* scheduling. In the first category, the algorithm must find a schedule map that optimises all of the QoS parameters to provide a suitable balance between them for time and cost, as in [10, 16–18]. In the second category, the algorithm makes a scheduling decision to optimise for some QoS parameter while subjected to some user-specified constraint values. For example, considering budget and makespan as the QoS parameters, an algorithm in the first category attempts to find a task-to-processor map that best balances between budget and makespan, while an algorithm of the second category takes user-defined values for the budget as an upper bound and defines a mapping that optimises the makespan. The first category of algorithms can produce schedules with shorter makespans, but the costs of which cannot be limited by the user when submitting the work. Next, we present a review of the second class of algorithms.

The Hybrid Cloud Optimised Cost scheduling algorithm (HCOC), proposed in [2], and a cost-based workflow scheduling algorithm called Deadline-MDP (Markov Decision Process), proposed in [25], address the problem of minimising cost while constrained by a deadline. Although these models could have applicability in a utility computing paradigm, we do not consider such a paradigm in this paper.

An Ant Colony Optimisation (ACO) algorithm to schedule large-scale workflows with QoS parameters was proposed by [7]. Reliability, time, and cost are three different QoS parameters that are considered in the algorithm. Users are allowed to define QoS constraints to guarantee the quality of the schedule. In [14, 21], the Dynamic Constraint Algorithm (DCA) was proposed as an extension of the Multiple-Choice Knapsack Problem (MCKP), to optimise two independent generic criteria for workflows, e.g., execution time and cost. In [22], a budget constraint workflow scheduling approach was proposed that used genetic algorithms to optimise workflow execution time while meeting the users budget. This solution was extended

in [23] by introducing a genetic algorithm approach for constraint-based, two-criteria scheduling (deadline and budget). In [4], the Balanced Time Scheduling (BTS) algorithm was proposed, which estimates the minimum resource capacity needed to execute a workflow by a given deadline. The algorithm has some limitations, such as homogeneity in resource type and a fixed number of computing hosts.

All of the previous algorithms apply guided random searches or local search techniques, which require significantly higher planning costs and thus are naturally time-consuming. Next, we consider algorithms that were proposed for contexts similar to that considered here, which are heuristic-based and have lower time complexity than the algorithms referred to above and which are used in the Results section for comparison purposes.

In [15] LOSS and GAIN algorithms were proposed to construct a schedule optimising time and constraining cost. Both algorithms use initial assignments made by other heuristic algorithms to meet the time optimisation objective; a reassignment strategy is then implemented to reduce cost and meet the second objective, the users budget. In the reassignment step, LOSS attempts to reduce the cost, and GAIN attempts to achieve a lower makespan while attending to the user's budget limitations. In the initial assignment, LOSS has lower makespans with higher costs, and GAIN has higher makespans with lower costs. The authors proposed three versions of LOSS and GAIN that differ in the calculation of the tasks weights. The LOSS algorithms obtained better performance than the GAIN algorithms, and among the three different types of LOSS strategy, we used LOSS1 to compare to our proposed algorithm. All of the versions of the LOSS and GAIN algorithms use a search-based strategy for reassignments; to obtain their goals, the number of iterations needed tends to be high for lower budgets in LOSS strategies and for higher budgets in GAIN strategies.

The algorithms LOSS and GAIN differ from our approach because they start with a schedule, and then changes are made iteratively to the schedule until the user budget is guaranteed. We do not consider any initial schedule, and in contrast to those algorithms, ours is not iterative; the time to produce a schedule is constant for a given workflow and platform.

A budget-constrained scheduling heuristic called *greedy time-cost distribution* (GreedyTimeCD) was proposed by [26]. The algorithm distributes the overall user-defined budget to the tasks, based on the estimated tasks average execution costs. The actual costs of allocated tasks and their planned costs are also computed successively at runtime. This is a different approach, which optimises task scheduling individually. First, a maximum allowed budget is specified for each task, and a processor is then selected that minimises time within the task budget.

In [27, 28] the *Budget-constrained Heterogeneous Earliest Finish Time* (BHEFT) was proposed, which is an extension of the HEFT algorithm [19]. The context of execution is an environment of multiple and heterogeneous service providers; BHEFT defines a suitable plan by minimising the makespan so that the user's budget and deadline constraints are met, while accounting for the load on each provider. An adequate solution is one that satisfies both constraints (i.e., budget and deadline); if no plan can be defined, it is considered a mapping failure. Therefore, the metric used by the authors was the planning success rate: the percentage of problems for which a plan was found. The BHEFT approach consists of minimising the execution time of a workflow (HEFT based) but within the budget constraints. Our approach is also one of minimising execution time while being constrained to a user-defined budget; therefore, we compare our proposed algorithm to BHEFT in terms of execution time versus budget. Additionally, we compare them in terms of plan success rate, where a deadline is specified for that purpose, similar to [27, 28].

Our algorithm differs from BHEFT in two important aspects: first, we allow more processors to be considered as affordable and, therefore, selected; and second, we do not necessarily select the processor that guarantees the earliest finish time, as BHEFT does. Instead, we compute a worthiness value, proposed in this paper, which combines the time and cost factors to decide on the processor for the current task.

GreedyTimeCD and BHEFT have time complexity of $O(v^2.p)$ for a workflow of v nodes and a platform with p processors. Next, we present our proposed scheduling algorithm, which minimises processing time while constrained to a user-defined budget and with low time complexity and which obtains better schedules then other state-of-the-art algorithms, namely LOSS1, GreedyTimeCD, and BHEFT.

4 Proposed Budget Constrained Scheduling Algorithm

In this section, we present the Heterogeneous Budget Constrained Scheduling (HBCS) algorithm, which minimises execution time while constrained to a user-defined budget. The algorithm starts by computing two schedules for the DAG: a schedule that corresponds to the minimum execution time that the scheduler can offer (e.g., produced with HEFT) and the highest cost; and another schedule that corresponds to the least expensive schedule cost on the target machines ($Cheapest_{Cost}$), as explained in Section 2. With the least expensive assignment, the user knows the minimum cost and corresponding deadline to execute the job; with the highest cost assignment, the user knows the minimum deadline that can be expected for the job and the maximum cost that should be spent to run the job. With this information, the user is able to verify whether the platform can execute the job before the required deadline and within the associated cost range. If these parameters satisfy the users expectations, he/she specifies the required budget according to (3). HBCS is shown in Algorithm 1.

Like most list-based algorithms [12], HBCS consists of two phases, namely a *task selection* phase and a *processor selection* phase.

4.1 Task Selection

Tasks are selected according to their priorities. To assign a priority to a task in the DAG, the upward rank ($rank_u$) [19] is computed. This rank represents the length of the longest path from task n_i to the exit node, including the computational time of n_i, and it is given by (4):

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{\overline{c_{i,j}} + rank_u(n_j)\} \quad (4)$$

where $\overline{w_i}$ is the average execution time of task n_i over all of the machines, $\overline{c_{i,j}}$ is the average communication time of an edge $e(i, j)$ that connects task n_i to n_j over all types of network links in the network, and $succ(n_i)$ is the set of immediate successor tasks to task n_i. To prioritise tasks, it is common to consider average values because they must be assigned a priority before the location where they will run is known.

4.2 Processor Selection

The processor selection phase is guided by the following quantities related to cost. We define the Remaining Cheapest Budget (RCB) as the remaining $Cheapest_{Cost}$ for unscheduled tasks, excluding the current task, and the Remaining Budget (RB) as the actual remaining budget. RCB is updated at each step before executing the processor selection block for the current task, using (5) (line 9), which represents the lowest possible cost of the remaining tasks:

Algorithm 1 HBCS algorithm

Require: DAG and user defined BUDGET

1: Schedule DAG with HEFT and Cheapest algorithm
2: Set task priorities with $rank_u$
3: **if** $HEFT_{cost} < BUDGET$
4: **return** Schedule Map assignment by HEFT
5: **end if**
6: $RB = BUDGET$ and $RCB = Cheapest_{Cost}$
7: **while** there is an unscheduled task **do**
8: n_i = the next ready task with highest $rank_u$ value
9: Update the Remaining Cheapest Budget (RCB) as defined in Eq.5
10: **for all** Processor $p_i \in P$ **do**
11: calculate $FT(n_i, p_j)$ and $Cost(n_i, p_j)$
12: **end for**
13: Compute $Cost_{Coeff}$ as defined in Eq.7
14: **for all** Processor $p_i \in P$ **do**
15: calculate $worthiness(n_i, p_i)$ as defined in Eq.10
16: **end for**
17: P_{sel} = Processor p_i with highest $worthiness$ value
18: Assign Task n_i to Processor P_{sel}
19: Update the Remaining Budget (RB) as defined in Eq.6
20: **end while**
21: **return** Schedule Map

$$RCB = RCB - Cost_{lowest} \quad (5)$$

where $Cost_{lowest}$ is the lowest cost for the current task among all of the processors. The initial value for the Remaining Budget is the user budget ($RB =$

$BUDGET$), and it is updated with (6) after the processor selection phase for the current task (line 19). RB represents the budget available for the remaining unscheduled tasks:

$$RB = RB - Cost(n_i, P_{sel}) \tag{6}$$

where P_{sel} is the processor selected to run the current task (line 17 of the algorithm), and $Cost(n_i, P_{sel})$ is the cost of running task n_i on processor P_{sel}.

The quantity Cost Coefficient ($Cost_{Coeff}$), defined by (7), is the ratio between RCB and RB, and it provides a measurement of the least expensive assignment cost relative to the remaining budget available. If $Cost_{Coeff}$ is near one, it means that the available budget only allows for selecting the least expensive processors:

$$Cost_{Coeff} = \frac{RCB}{RB} \tag{7}$$

HBCS minimises execution time. Therefore, the finish time of the current task (n_i) is computed for all of the processors ($FT(n_i, p_j)$) at lines 10 and 11, and they constitute one set of factors for processor selection. The other set of factors consists of the costs of executing the task on each processor, that is $Cost(n_i, p_j)$. The variables p_{best} and p_{worst} are defined as the processors with shortest and longest finish times for the current task, respectively.

Processor selection is based on the combination of two factors: time and cost. Therefore, we define two relative quantities, namely Time rate ($Time_r$) and Cost rate ($Cost_r$), for the current task n_i on each processor $p_j \in P$, shown in (8) and (9), respectively:

$$Time_r(n_i, p_j) = \frac{FT_{worst} - FT(n_i, p_j)}{FT_{worst} - FT_{best}} \tag{8}$$

$$Cost_r(n_i, p_j) = \frac{Cost_{best} - Cost(n_i, p_j)}{Cost_{highest} - Cost_{lowest}} \tag{9}$$

where FT_{best} and $Cost_{best}$ are the finish time and cost of the current task on processor p_{best}, respectively. FT_{worst} is the finish time on processor p_{worst}, and $Cost_{highest}$ and $Cost_{lowest}$ are the highest and the lowest cost assignments for the current task among all of the available processors, respectively. $Time_r$ measures how much shorter than the worst finish time (FT_{worst}) the finish time is of the current task on processor p_j. Similarly, $Cost_r$ measures how much less the actual cost on p_j is than the cost on the processor that results in the earliest finish time. Both variables are normalised to their highest ranges.

Finally, to select the processor for the current task n_i, the worthiness value for each processor $p_j \in P$ is computed as shown in (10):

$$worthiness(n_i, p_i) = \begin{cases} -\infty & \text{if } Cost(n_i, p_j) > Cost_{best} \\ -\infty & \text{if } Cost(n_i, p_j) > RB - RCB \\ Cost_r(n_i, p_j) & \\ .Cost_{Coeff} & \\ +Time_r(n_i, p_j) & \text{otherwise} \end{cases} \tag{10}$$

The first two statements guarantee that if the cost of task n_i on processor p_j is higher than the cost on the processor that gives the minimum FT and if that cost is higher than the available budget for task n_i, then processor p_j cannot be selected. With these statements, the resulting schedule does not exceed the user budget and is guaranteed to be valid. In the third statement, the worthiness value depends on the available budget and on the time during which a processor can finish the task. If the remaining budget (RB) is high, then $Time_r$ has more influence, and a processor with the greater difference in FT compared to the worst processor will have higher worthiness. In contrast, if the remaining budget is smaller, then the cost factor will increase the worthiness of the processors with lower cost to run task n_i. After testing all of the processors, the one with highest worthiness value is selected, and the remaining budget is updated according to (6).

The two phases, task selection and processor selection, are repeated until there are no more tasks remaining to schedule. In terms of time complexity, the HBCS requires the computation of HEFT and the least expensive schedule, which are used as pre-requisites to calculate several variables in the HBCS algorithm; these are $O(v^2.p)$ and $O(v^2.p^*)$, respectively, where p^* is the number of cheapest processors. In our platform with heterogeneous clusters of homogeneous processors, there is more than one processor with the cheapest cost. The least expensive strategy attempts to assign each task to the least expensive processor; considering insertion policy to calculate the Earliest Finish Time (EFT) among all of the cheapest processors, the complexity of the cheapest algorithm is $O(v^2.p^*)$. The complexities of the two phases of HBCS are of the same order as the HEFT algorithm: $O(v^2.p)$. In conclusion, the total time is $O(v^2.p + v^2.p^* + v^2.p)$, which results in time complexity of the order $O(v^2.p)$.

5 Experimental Results

This section presents performance comparisons of the HBCS algorithm with the LOSS1 [15], GreedyTime-CD [26], BHEFT [28], and Cheapest scheduling algorithms. We consider synthetic randomly generated and Real Application workflows to evaluate a broader range of loads. The results presented were produced with SimGrid [6], which is one of the best-known simulators for distributed computing and which 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 LIGO and Epigenomics [1]. The randomly generated workflows were created by the synthetic DAG generation program.[1] The computational complexity of a task was modelled as one of the three following forms, which are representative of many common applications: $a.d$ (e.g., image processing of a $\sqrt{d}.\sqrt{d}$ image), $a.d\log d$ (e.g., sorting an array of d elements), and $d^{3/2}$ (e.g., multiplication of $\sqrt{d}.\sqrt{d}$ matrices), where a is chosen 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; 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 indicates that the levels contain very dissimilar numbers of tasks, whereas a high value indicates that all of the levels contain similar numbers 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, we used as the number of tasks $n = [10...60]$, $jump = [1, 2, 3]$, $regularity = [0.2, 0.4, 0.8]$, $fat = [0.2, 0.4, 0.8]$, and $density = [0.2, 0.4, 0.8]$. With these parameters, each DAG was created by choosing the value for each parameter randomly from the parameter data set. The total number of DAGs generated in our simulation was 1000.

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[2] [6] 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. In many research papers on scheduling, the authors have assumed a contention-free network model, in which processors

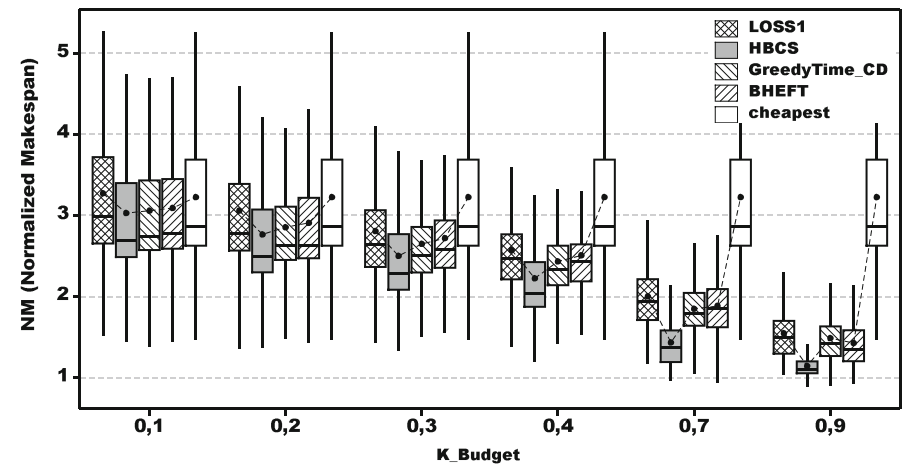
[1] https://github.com/frs69wq/daggen

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

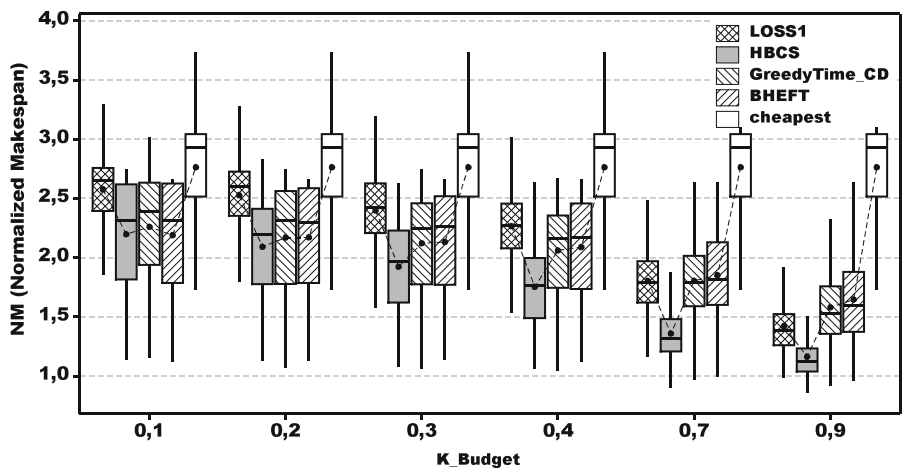
Table 1 Description of the Grid5000 clusters from which the platforms used in the experiments were derived

Site	Cluster	$\#CPU_{Total}$	$\#CPU_{used}$			Power(GFlop/s)	σ
Lille	chicon	26	2	4	9	8.9618	
	chimint	20	2	4	7	23.531	0.69
	chinqchint	46	4	8	16	22.270	
Sophia	helios	56	3	6	12	7.7318	
	sol	50	3	5	10	8.9388	0.90
	suno	45	2	5	10	23.530	

can simultaneously send data to or receive data from as many processors as possible, without experiencing any performance degradation. Unfortunately, that model, the *multi-port* model, is not representative of actual network infrastructures. Conversely, the network model provided by SimGrid corresponds to a theoretical *bounded multi-port* model. In this model, a processor can communicate with several other processors simultaneously, but each communication flow is limited by the bandwidth of the traversed route, and communications using a common network link must share bandwidth. This scheme corresponds well to the behaviour of TCP connections on a LAN. The validity of this network model was demonstrated by [20].



(a) $\mathrm{Cluster}_{name}$ = Lille



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

Fig. 1 Normalized Makespan for Random workflows with CPU_{used}=8

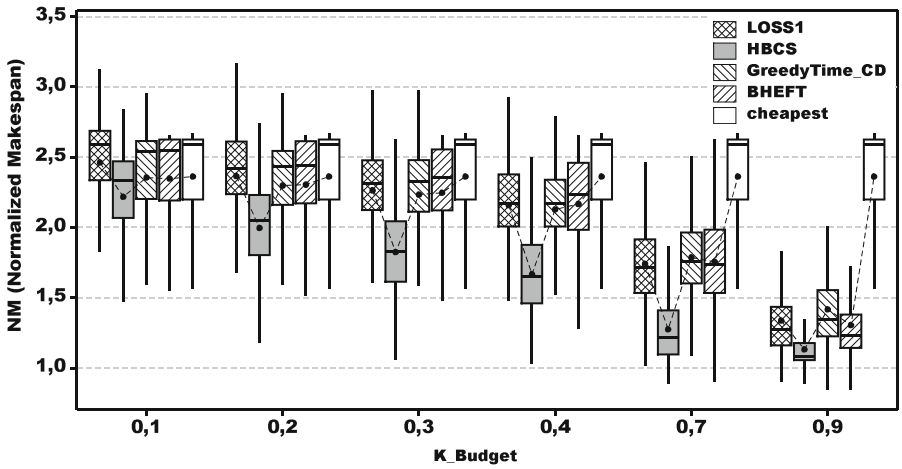
We consider two sites that comprise multiple clusters. Table 1 provides the name of each cluster, along with its number of processors, processing speed expressed in GFlop/s, and the standard deviation (σ) of CPU capacity as a heterogeneity factor. Column $\#CPU_{used}$ shows the number of processors used from each cluster for 8-, 16- and 32-processor configurations.

5.3 Budget-Deadline Constrained Scheduling

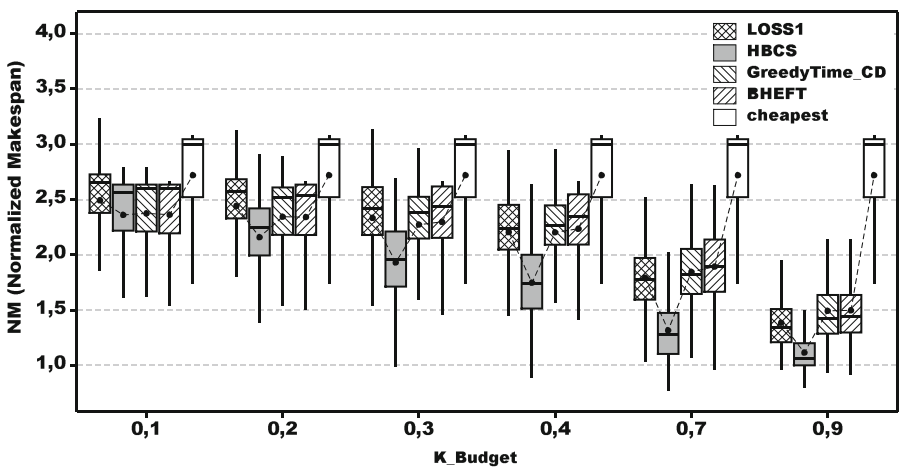
Budget-deadline constrained scheduling was introduced in [27, 28], in which the BHEFT algorithm was presented. Its aim is to produce mappings of tasks to resources that meet the deadline and budget constraints imposed by the user. Although BHEFT considers the load of the providers, the algorithm can be evaluated by attending exclusively to the deadline and budget constraints and ignoring the providers loads, as also presented in [27, 28]. This process allows for evaluating and comparing the algorithms, without the additional random variables that are the providers loads. The budget factor is considered here, as defined by (3). The deadline constraint is defined by (11), which specifies a deadline value for a given workflow, based on the mapping obtained by HEFT:

$$DeadLine = LB_{deadline} + k_{Deadline}\left(UB_{deadline} - LB_{deadline}\right) \quad (11)$$

where $LB_{deadline}$ is equal to $HEFT_{Makespan}$, the makespan obtained with HEFT, and $UB_{deadline}$ is



(a) Cluster_{name} = Lille



(b) Cluster_{name} = Sophia

Fig. 2 Normalized Makespan for Random workflows with CPU_{used}=32

3.$HEFT_{Makespan}$. The range of values for $k_{Deadline}$ is [0 . . . 1]. $HEFT_{Makespan}$ is the minimum time that the user is allowed to choose to obtain a valid schedule. We restrict the upper bound deadline as the cube of $HEFT_{Makespan}$ to evince the quality of the algorithms in generating valid mappings.

5.4 Performance Metrics

To evaluate and compare our algorithm with other approaches, we consider the metric Normalised Makespan (NM), defined by (12). NM normalises the makespan of a given workflow to the lower bound, which is the workflow execution time obtained with HEFT:

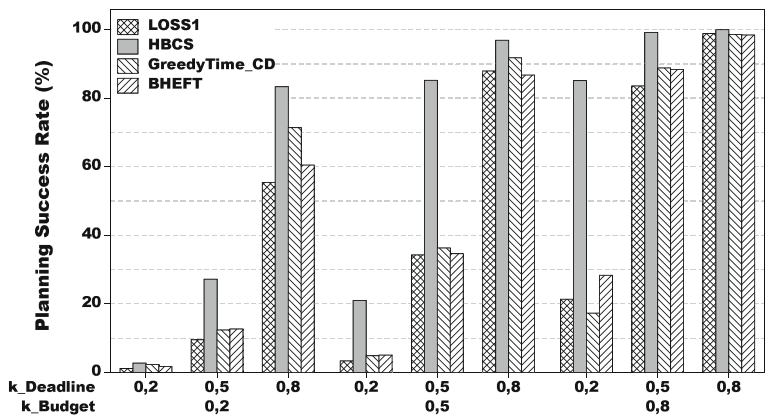
$$\text{NM} = \frac{\text{schedule makespan}}{HEFT_{Makespan}} \tag{12}$$

To evaluate the algorithms using the budget-deadline constrained methodology, we consider the Planning Success Rate (PSR), as expressed by (13) and defined in [27, 28]. This metric provides the percentage of valid schedules obtained in a given experiment.

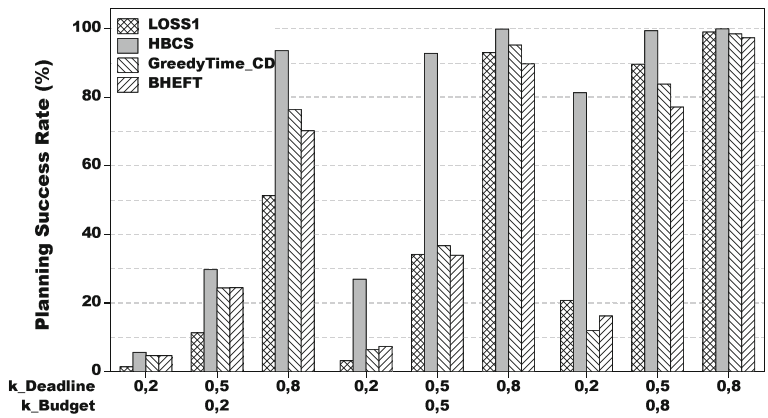
$$\text{PSR} = 100 \times \frac{\text{Successful Planning}}{\text{Total Number in experiment}} \tag{13}$$

5.5 Results and Discussion

Among all of the algorithms mentioned above in the related work section, we selected LOSS1, GreedyTimeCD, BHEFT, and Cheapest scheduling algorithms; these match our goal and conditions, but we have made some modifications in their original



(a) Cluster_{name} = Lille



(b) Cluster_{name} = Sophia

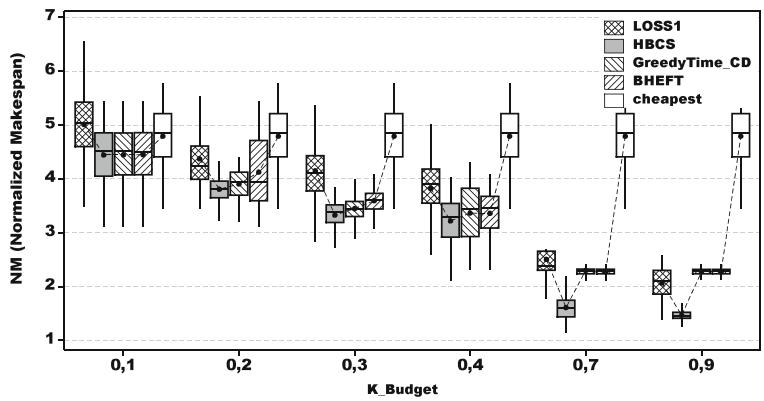
Fig. 3 Planning Success Rate for Random workflows

strategies. The original implementation of the LOSS 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 grid environment, each cluster was homogeneous and was based only on the cost parameter, so there could be more than one processor candidate. In this case, we tested all of the possible processors and selected that which achieved the smallest makespan. The same procedure was applied to the least expensive scheduling strategy, which attempts to schedule each task on the service with the lower execution cost.

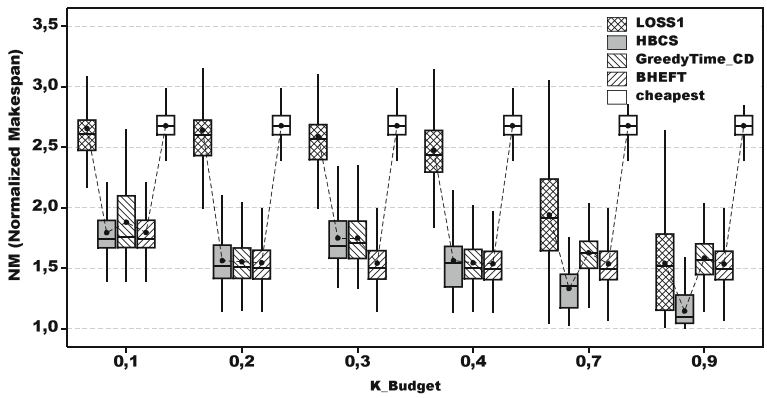
To evaluate the performance of each algorithm, we computed the makespan for each DAG and normalised it using (12). The results presented next were obtained by SimGrid with the bounded multi-port model, and they are presented for two data sets: randomly generated workflows and real applications.

5.5.1 Results for Randomly Generated Workflows

For random DAG generation, as stated previously, we model the computational complexity of common applications tasks, such as image processing, array sorting, and matrix multiplication. Figures 1 and 2 show the Normalised Makespan values obtained on two CPU configurations per site, namely, 8 and 32 processors. We can see that the HBCS algorithm obtains significant performance improvements over the other algorithms for most of the user-defined budget values and configurations. For 8 processors on the Lille site, the improvement of HBCS over BHEFT is 2.0 % for $k = 0.1$, and it increases to 19.7 % for $k = 0.9$. On the Sophia site, the improvement is 0 % for $k = 0.1$, and it increases to 29.4 % for $k = 0.9$. For $k \geq 0.7$, the second-best algorithm is LOSS1, and the improvements of HBCS over LOSS1 are 24.7 % and 18.2 % for $k = 0.7$ and $k = 0.9$, respectively.

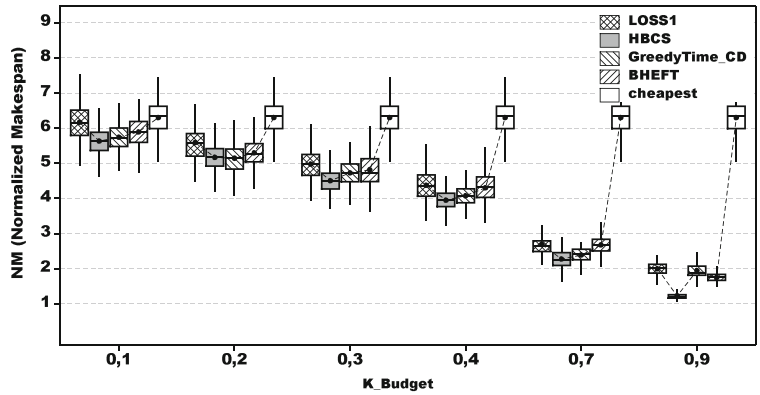


(a) Cluster_{name} = Lille , CPU_{used}=8

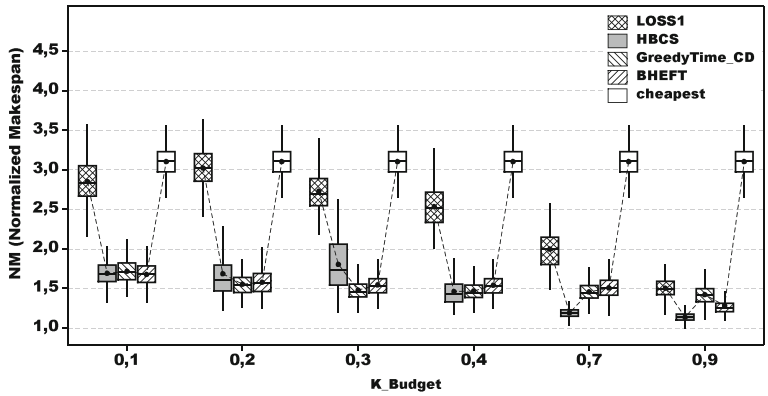


(b) Cluster_{name} = Sophia , CPU_{used}=8

Fig. 4 Normalized Makespan for Epigenomics workflows on GRID5000



(a) Cluster_{name} = Lille , CPU_{used}=8



(b) Cluster_{name} = Sophia , CPU_{used}=8

Fig. 5 Normalized Makespan for LIGO workflows on GRID5000

For 32 processors, the same pattern of results is obtained on the Lille and Sophia sites but with greater differences for lower k values. On the Lille site, the improvement of HBCS over BHEFT is 5.4 %, 13.4 % and 18.8 % for k values of 0.1, 0.2, and 0.3, respectively. The performance of BHEFT and GreedyTime-CD are very similar for all of the ranges of k on both sites. For $k \geq 0.7$, the second best algorithm is again LOSS1.

In conclusion, HBCS outperforms all of the other algorithms, indicating that HBCS can achieve lower makespans for any given budget.

The results of the evaluation made with the budget-deadline constrained methodology are shown in Fig. 3, where the deadline and budget factors assume the values of 0.2, 0.5, and 0.8. The figures present the average values for 8, 16 and 32 processors for each site. HBCS yielded higher PSR values for all of the cases. The PSR obtained by HBCS for shorter deadline factors is significantly higher than the PSR values obtained with the other algorithms. On both sites, for deadline and budget factors of 0.2 and 0.8, respectively, the HBCS PSR is greater than 80 %, while for the remaining algorithms, the PSR is less than 30 %. We can conclude that HBCS obtains the highest Planning Success Rates for the two levels of heterogeneity considered here.

5.5.2 Results for Real World Applications

To evaluate the algorithms on a standard and real set of workflow applications, synthetic workflows were generated using the code developed in [13]. Two well-known structures were chosen [11], namely Epigenomics,[3] for mapping the epigenetic state of human DNA, and LIGO.[4] The Epigenomics workflow is a highly pipelined application with multiple pipelines operating on independent chunks of data in parallel. In

[3] USC Epigenome Center, http://epigenome.usc.edu

[4] http://www.advancedligo.mit.edu

these two real applications, most of the jobs have high CPU utilisation and relatively low I/O, so they can be classified as CPU-bound workflows.

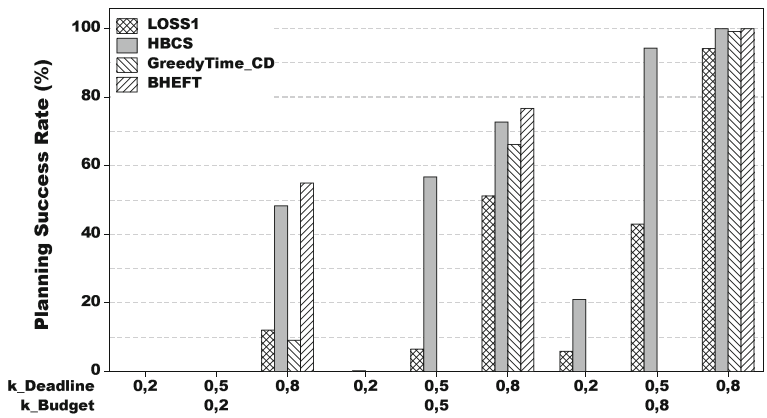
Workflows with 24 and 30 tasks were created for both Epigenomics and LIGO. For each workflow size, 300 different workflow instances were generated, obtaining a total collection of 600 workflows for each type of application.

For Normalised Makespan, Figs. 4 and 5 show that HBCS achieves better performance for most of the budget factor values. The improvements increase with the budget factor and the site heterogeneity. Similar behaviour is obtained for the LIGO application.

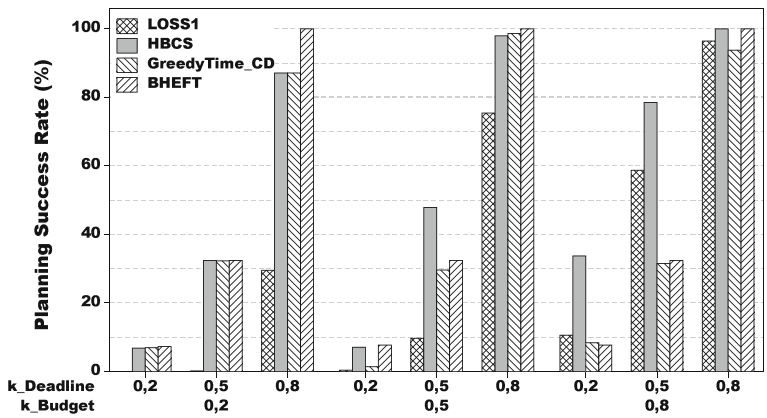
Figures 6 and 7 show the Planning Success Rate for the Epigenomics and LIGO workflows. For each site, the average PSRs for 8, 16, and 32 CPUs are presented. These results are consistent with those obtained for the randomly generated workflows. It is also observed that for some cases, particularly for Epigenomics, only HBCS obtained valid schedules and significant success rates.

6 Conclusions and Future Work

In this paper, we have presented the Heterogeneous Budget Constrained Scheduling algorithm, which maps a workflow to a heterogeneous system and minimises the execution time constrained to a user-defined budget. The algorithm was compared with other state-of-the-art algorithms and was shown to achieve lower makespans for all of the budget factors in higher heterogeneity platforms; that is, HBCS can produce shorter makespans for the same budget. A reduction of up to 30 % in execution time was achieved while maintaining the same budget level. Considering Budget-Deadline constrained scheduling, HBCS achieves a higher planning success rate for

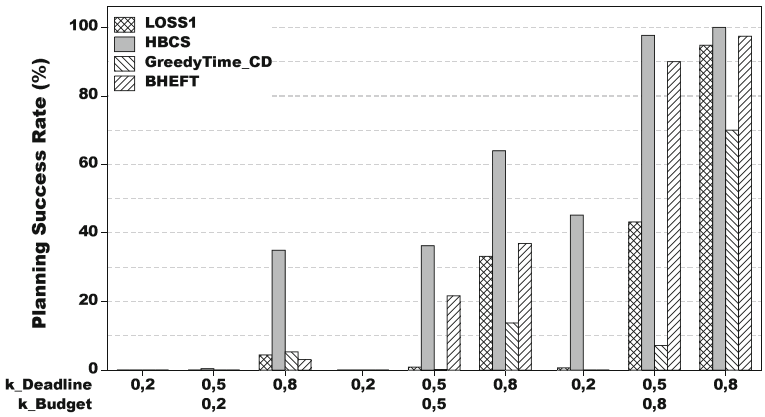


(a) Cluster_{name} = Lille

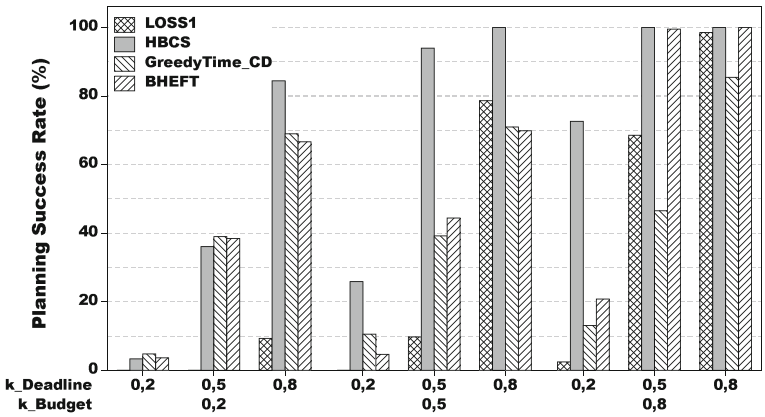


(b) Cluster_{name} = Sophia

Fig. 6 Planning Success Rate for Epigenomics workflows



(a) Cluster_{name} = Lille



(b) Cluster_{name} = Sophia

Fig. 7 Planning Success Rate for LIGO workflows

more heterogeneous systems. In some particular cases, the PSR obtained with HBCS is considerably higher than the PSRs obtained with the other algorithms. For example, for the budget and deadline factors of 0.8 and 0.2, respectively, on the Lille and Sophia sites, the PSR of HBCS is greater than 80 %, while all of the other algorithms achieve PSRs less than 30 %. On these sites, HBCS performs significantly better than the other algorithms for the more restricted deadline factors of 0.2 and 0.5.

The results achieved for two real applications, namely LIGO and Epigenomics, are consistent with the randomly generated workflows for both metrics.

Concerning time complexity, HBCS has the same complexity as GreedyTimeCD and BHEFT, having a constant running time for all ranges of budget factors for a given workflow and platform, because it has bounded complexity. In contrast, the LOSS algorithms have a search-based step, and their running time therefore depends on the number of iterations to find a solution, where for lower budgets, the number of iterations is significantly greater.

In conclusion, we have presented the HBCS algorithm for budget constrained scheduling, which has proved to achieve better performance with lower time complexity than other state-of-the-art algorithms. The results were obtained in a simulation with a realistic model of the computing platform and with shared links, as occurs in a common grid infrastructure.

In future work, we intend to extend the algorithm to consider the dynamic concurrent DAG scheduling problem. This consideration will allow users to execute concurrent workflows that might not be able to start together but that can share resources so that the total cost for the user can be minimised.

Acknowledgments This work was supported in part by the Fundação para a Ciência e Tecnologia, PhD Grant FCT - DFRH - SFRH/BD/80061/2011.

References

1. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: Third Workshop on Workflows in Support of Large-Scale Science, 2008. WORKS 2008, pp. 1–10. IEEE (2008)
2. Bittencourt, L.F., Madeira, E.R.M.: Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. J. Internet Serv. Appl. **2**(3), 207–227 (2011)
3. Broberg, J., Venugopal, S., Buyya, R.: Market-oriented grids and utility computing: The state-of-the-art and future directions. J. Grid Comput. **6**(3), 255–276 (2008)
4. Byun, E.-K., Kee, Y.-S., Kim, J.-S., Deelman, E., Maeng, S.: Bts: Resource capacity estimate for time-targeted science workflows. J. Parallel Dist. Comput. **71**(6), 848–862 (2011)
5. Canon, L.C., Jeannot, E., Sakellariou, R., Zheng, W.: Comparative evaluation of the robustness of dag scheduling heuristics. In: Grid Computing, pp. 73–84. Springer (2008)
6. Casanova, H., Legrand, A., Quinson, M.: Simgrid: a generic framework for large-scale distributed experiments. In: Proceedings of the Tenth International Conference on Computer Modeling and Simulation, UKSIM '08, pp. 126–131. IEEE Computer Society, Washington (2008)
7. Chen, W.-N., Zhang, J.: An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **39**(1), 29–43 (2009)
8. Coffman, E.G., Bruno, J.L.: Computer and Job-shop Scheduling Theory. Wiley (1976)
9. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., et al: Mapping abstract complex workflows onto grid environments. J. Grid Comput. **1**(1), 25–39 (2003)
10. Dŏgan, A., Özguner, F.: Bi-objective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems. Comput. J. **48**(3), 300–314 (2005)
11. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Futur. Gener. Comput. Syst. **29**(3), 682–692 (2013)
12. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999)
13. Pegasus. Pegasus workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator (2013)
14. Prodan, R., Wieczorek, M.: Bi-criteria scheduling of scientific grid workflows. IEEE Trans. Autom. Sci. Eng. **7**(2), 364–376 (2010)
15. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.: Scheduling workflows with budget constraints. Integr. Res. Grid Comput., 189–202 (2007)
16. Singh, G., Kesselman, C., Deelman, E.: 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, pp. 117–126. ACM (2007)
17. Sen, S., Li, J., Qingjia, H., Shuang, K., Wang, J.: Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput. **39**, 177–188 (2013)
18. Szabo, C., Kroeger, T.: Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. In: WCCI IEEE World Congress on Computational Intelligence, pp. 1–8. IEEE (2012)
19. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
20. Velho, P., Legrand, A.: Accuracy study and improvement of network simulation in the simGrid framework. In: Proccedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools). Rome, Italy (2009)
21. Wieczorek, M., Podlipnig, S., Prodan, R., Fahringer, T.: Bi-criteria scheduling of scientific workflows for the grid. In: 8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID'08, pp. 9–16. IEEE (2008)
22. Yu, J., Buyya, R.: A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In: Workshop on Workflows in Support of Large-Scale Science, 2006. WORKS'06, pp. 1–10. IEEE (2006)
23. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program. **14**(3), 217–230 (2006)
24. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow scheduling algorithms for grid computing. Metaheuristics Sched. Distrib. Comput. Environ., 173–214 (2008)
25. Yu, J., Buyya, R., Tham, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. In: First International Conference on e-Science and Grid Computing, 2005, pp. 8–pp. IEEE (2005)
26. Jia, Y., Ramamohanarao, K., Buyya, R.: Deadline/budget-based scheduling of workflows on utility grids. Market-Oriented Grid Util. Comput., 427–450 (2009)
27. Zheng, W., Sakellariou, R.: Budget-deadline constrained workflow planning for admission control in market-oriented environments. In: Economics of Grids, Clouds, Systems, and Services, pp. 105–119. Springer (2012)
28. Zheng, W., Sakellariou, R.: Budget-deadline constrained workflow planning for admission control. J. Grid Comput., 1–19 (2013)