Multi-Workflow QoS-Constrained Scheduling for Utility Computing

Hamid Arabnejad
Universidade do Porto, Faculdade de Engenharia,
Dep. de Engenharia Informática,
LIACC, Porto, Portugal
Email: hamid.arabnejad@fe.up.pt

Jorge G. Barbosa
Universidade do Porto, Faculdade de Engenharia,
Dep. de Engenharia Informática,
LIACC, Porto, Portugal
Email: jbarbosa@fe.up.pt

***Abstract*—In this paper, we introduce a utility driven strategy to schedule concurrent workflows constrained to user's QoS parameters, namely Deadline and Budget. The Multi-Workflow Deadline-Budget Scheduling algorithm (MW-DBS) can schedule multiple workflows that can arrive to the system at any instant of time, with the aim of satisfying individual QoS requirements. Common approaches optimize one factor, e.g. processing time, constrained to the other factor, e.g. cost. MW-DBS produces schedules without optimizing any of the parameters but guaranteeing that the deadline and budget defined for each workflow are not exceeded. We study the scalability of the algorithm with different types of workflows and service providers. Experimental results show that our strategy is able to increase the scheduling success rate of finding valid solutions.**

Keywords*—*concurrent workflows; grids; clouds; deadline; budget.

I. 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.

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 [8]. 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.

Recently, several algorithms have been proposed for concurrent workflow scheduling in order to improve the execution time of several applications in a Heterogeneous Computing System (HCS). However, most of these algorithms were designed for off-line scheduling or static scheduling, that is all the applications are known at the same time [22]. This approach although relevant, imposes limitations in the management of a dynamic system where users can submit jobs at any time. For this purpose there are a few algorithms that were designed to deal with dynamic application scheduling [1], [4], [10], [21]. These algorithms minimize one or two objectives, makespan and cost, but they do not guarantee a pre-defined deadline and budget constraint.

To address the problem mentioned above, we introduced a new scheduling strategy, Multi-Workflow Deadline-Budget scheduling algorithm (MW-DBS), for scheduling concurrent workflow applications constrained to the user's defined deadlines and budgets. The MW-DBS algorithm is a list scheduling approach and is, therefore, composed by two main steps: first, a ready task from each workflow is selected and a priority based on individual deadline is assigned, and secondly, a suitable resource to execute the current task that satisfies the QoS parameters of the workflow where the task belongs, is determined.

The remainder of the paper is organized as follows. In the next section we describe the related work. In section III, we present the problem definition. Section IV, presents the details of our MW-DBS scheduling algorithm. Section V present results and section VI concludes the paper.

II. Related Work

In general, scheduling algorithms on heterogeneous computing systems have focused on single workflow applications. In contrast, multiple workflow scheduling has not received much attention. In [24] it was proposed a scheduling algorithm in order to optimize performance and cost for Workflows in the Cloud. Bochenina in [6] introduced a strategy for mapping the tasks of multiple workflows with different deadlines on the static set of resources. Kumar et al. [13] proposed a time and cost optimization algorithm for hybrid clouds. Using a schedule map of a workflow, authors in [11] proposed a method to minimize the total execution time of a scheduling solution for concurrent workflows in HPC cloud. In [5] four strategies, which differ in the ordering of tasks, for scheduling multiple workflows on Grids are discussed. Zhao and Sakellariou in [22] proposed to merge multiple workflow application into a single DAG which can be scheduled by traditional DAG scheduling algorithms in order to minimize the overall makespan and achieve fairness, defined based on the slowdown experienced by each DAG due to competition for resources with other DAGs. But most of these strategies are designed for off-line workflow scheduling which imposes limitations on the management of a dynamic system where users can submit jobs at any moment in time, i.e, they scheduled only available and submitted workflows at this time and after a schedule is produced and initiated, no other workflow is considered.

There are few on-line algorithms proposed specifically to schedule concurrent workflows with the aim of improving individual QoS requirements. In [21] and [10] there were proposed two algorithms namely RANK_HYBD and OWM (Online Workflow Management), to schedule multiple online workflows and targeting to have lower average makespan and turnaround time for submitted workflows at different instants of time and by different users. In [1], [4] authors proposed the fairness dynamic workflow scheduling (FDWS) algorithm. The main objective of FDWS algorithm is to reduce the individual turnaround time for each workflow application in the system. The FDWS algorithm focuses on the QoS experienced by each application (or user) by minimizing turnaround time, while RANK_HYBD and OWM algorithms try to reduce the average completion time of all workflows.

Xu et al.[18] propose a multiple QoS constrained scheduling strategy of multi-workflows (MQMW) for cloud computing. The MQMW algorithm minimizes the makespan and cost of workflows which can be submitted and start at any time. In [3], authors proposed on-line strategies for concurrent workflows that extend the former concurrent on-line scheduling algorithms by considering fairness resource sharing constrained to the user defined budget and optimize the turnaround time, for each workflow application.

However, our approach here is different from the algorithms described above, in that we simultaneously consider time and cost as constraints of the scheduling problem and do not perform optimization. The scheduling algorithm proposed in this paper is the first multiple online workflow scheduling algorithm that simultaneously considers user's budget and deadline constraints for concurrent workflow scheduling in heterogeneous computing systems.

III. Scheduling System Model

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

A. System Model

The target utility computing platform is composed by a set of heterogeneous resources which provide services of different capabilities and costs [8]. Processors price is defined so that the most powerful processor has the highest cost and the less powerful processor has the lowest cost. In an utility grid the resource price is commonly defined and charged per time unit [16], [19], [23], so that if a task takes k time units to process in a resource that costs y euros per time unit, then 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, being a common practice charging per hour, such as in Amazon Elastic Compute Cloud (Amazon EC2) [1], and partial hours are rounded up. In our model, users share a set of resources to execute concurrently their applications, so that the cost impact of the granularity of reservation is not applicable individually. It is expected that the resources are reserved for some multiples of the minimum reservation period and the aim is to maximize the number of jobs that complete within the individually defined QoS parameters. Therefore, we consider that the heterogeneous system 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.

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

B. 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 tuple $G = < T, E >$. 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. The $\overline{C}_{(t_i \to t_j)}$ represents the average communication time between the tasks t_i and t_j which is calculated based on the average bandwidth and latency among all processor pair. 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 [17]. Due to heterogeneity, each task may have a different execution time on each processor.

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.

Makespan or Schedule length denotes the finish time of the last task of the workflow and is defined as:

$$makespan = FT(t_{exit}) \tag{1}$$

where $FT(t_i)$ denotes the Finish Time of task t_i on the processor assigned by the scheduling algorithm.

The cost of executing task t_i on processor p_k is defined as $Cost(t_i, p_k) = ET(t_i, p_k) \times Price(p_k)$ where $ET(t_i, p_k)$ represents the Execution Time to complete task t_i on processor p_k and $Price(p_k)$ denotes the price of processor p_k. $TotalCost$ is the overall cost for executing an application and is defined as:

$$TotalCost = \sum_{t_i \in T} AC(t_i) \tag{2}$$

where $AC(t_i)$ is defined as Assigned Cost of task t_i. After assigning the selected processor p_{sel} to execute task t_i, the assigned cost value is equal to $AC(t_i) = Cost(t_i, p_{sel})$. We considered zero monetary costs for communications between tasks because they occur inside a given site.

C. Scheduling Objectives

For a given scenario which includes a set of workflow applications that are submitted at different instants of time, by users with individual time and cost constraints, the objective is to find a schedule map for each workflow application that can meet its user-defined QoS constraints, i.e, the completion time of the workflow must not exceed the user defined time constraint (DEADLINE), and the total cost must not be higher than the user defined budget. Note that the completion time (or turnaround time) includes both the waiting time and execution time of a given workflow, extending the makespan definition for the single workflow scheduling problem [14]. These values of DEADLINE and BUDGET for each workflow application should be negotiated between the user and the provider in a range of feasibility values so that there are feasible scheduling solutions. Note that there is not any optimization function in the formulation, so that any scheduling solution that match the constraints, for a given workflow, is a plausible solution.

IV. The Proposed Algorithm

In this section, we present the Multi-Workflow Deadline-Budget scheduling algorithm (MW-DBS), which aims to find a feasible schedule within an individual BUDGET and DEADLINE constraints value for each submitted application. The MW-DBS algorithm is designed as a heuristic strategy that in each processor selection phase, for the selected task, obtains a schedule that always accomplishes the cost constraint (BUDGET) and that may accomplish the deadline constraint of the workflow application to which the task belongs to. If the time constraint (DEADLINE) of the workflow application is met, we have a successful schedule, otherwise we have a failure and the workflow application should be terminated. The algorithm is evaluated based on the success rate.

Generally, in most on-line scheduling algorithms, without in advance reservation, the scheduler is called when an executing task finishes and there is at least one free processor to execute new tasks. Like other online scheduler, MW-DBS algorithm consists of two main phases, namely a *task selection* phase and a *processor selection* phase as described next. For covering and meeting the QoS parameters in online scheduling strategy, the two critical keys are: 1) selection of the appropriate task in *task selection* phase among all ready tasks from each available workflow applications, and 2) to find a proper resource to execute the selected task. In both phases, the MW-DBS algorithm needs to consider user's QoS parameters in order to have solutions of higher quality and higher system performance.

A. Task Selection

The MW-DBS algorithm selects a suitable task to be executed among all ready-to-execute tasks from *ready tasks* pool which is filled by the ready tasks belonging to each submitted and unfinished workflow applications. In general, two methods are used to fill the *ready tasks* pool: a) collect a single ready task from each workflow [1], [3], [4], [10], or b) collect all ready tasks [21] belonging to each unfinished workflow application. Adding all ready tasks from each available workflow results in an unfair strategy because the high number of ready tasks may cause that some workflow applications do not participated in the current scheduling round. Therefore, in MW-DBS algorithm, to fill the *ready tasks* pool, a single ready task with highest upward rank [17] ($rank_u$) is selected and added into the pool. The key point in the task selection phase is to determine which task should be selected for scheduling among all ready-to-execute tasks. So, a priority assignment strategy based on the QoS parameters is needed to assign a secondary rank to each ready-to-execute task in the *ready tasks* pool.

The RANK_HYBD [21] and OWM [10] algorithms in order to reduce total execution time, use upward rank ($rank_u$) as secondary rank. The FDWS[1], [4] assigns a secondary priority rank ($rank_r$) and selects the task with highest $rank_r$ among all ready tasks. The $rank_r$ is calculated based on the number of remaining tasks and critical path length of the workflow to which the selected task belongs to. Therefore, because of lower expected finish time for wider workflows and workflows with low number of tasks, they have higher priority in FDWS algorithm. Also, by using the number of remaining tasks in the $rank_r$ priority, the FDWS algorithm considers the workflow history to make a scheduling decision. Because the time optimization is the objective in these algorithms, they use time measure in their secondary rank priority assignment strategy. For cost optimization, in [3], authors use $rank_B$ priority, which is calculated based on the fraction of unscheduled tasks and remaining budget of each workflow application in order to consider fairness resource sharing.

In this paper, because both time and cost factors are constraints, the secondary rank should consider both measures. Additionally, results in [1], [4] show that taking into account the workflow history of scheduled tasks leads us to a better performance. In this paper, we propose a new rank ($rank_D$) to assign a secondary priority to each task t_i belonging to workflow j in the *ready tasks* pool, defined by equation 3:

$$rank_D(t_i, j) = \frac{1}{DEADLINE_j} \times STR_j \tag{3}$$

where, $DEADLINE_j$ is the time constraint (DEADLINE) of workflow application j. The Scheduled Tasks Ratio STR_j is the ratio of the number of scheduled tasks to the total number of tasks in the workflow application j.

The $rank_D$ value is the product of two factors: a) the inverse of time limitation value (DEADLINE) for workflow j, giving higher priority to the submitted and unfinished workflow applications that have a lower deadline value; and b) the fraction of the workflow j that is scheduled in the system. The rational of the first factor is that the scheduler will consider first workflow applications that have lower time limitation before failing. And the rational of scheduled tasks ratio is to give higher priority to workflows that where submitted earlier, so that a longer workflow with several tasks already executed may have priority over a short and recent workflow. In the case where two workflows have the same deadline, the task belonging to the workflow that has higher ratio of scheduled tasks, will have higher priority than the other.

Finally, the task with highest $rank_D$ in *ready tasks* pool is selected to be schedule in the next phase.

B. Processor Selection

The Processor selection phase has responsibility for selecting an affordable resource for the current task. In this part, we propose a new strategy for processor selection based on QoS requirements. As every workflow is constrained to time and cost, both parameters should be considered in the processor selection phase. In order to control the consumed cost and time, a bound value for each factor is needed. In the following paragraphs, first bound values for cost and time are described and then our new strategy for processor selection is described.

Cost bound: there is a limitation on budget consumption by current task t_{curr} based on used budget by scheduled tasks and available budget. $Cost_{Bound}(t_{curr})$ is defined as the maximum available budget for the current task (t_{curr}) that can be consumed by its assignment. $Cost_{Bound}(t_{curr})$ is calculated as the minimum cost for the current task plus the spare available budget:

$$Cost_{Bound}(t_{curr}) = Cost_{min}(t_{curr}) + \Delta_j^{Cost} \tag{4}$$

where $Cost_{min}$ denotes the minimum execution cost of the current task among all processors and Δ_j^{Cost} represents the spare budget defined as the difference between unconsumed budget and cheapest cost assignment for unscheduled tasks for workflow j which task t_{curr} belongs to:

$$\Delta_j^{Cost} = RB_j - RCA_j \tag{5}$$

where RB_j is the Remaining unconsumed Budget of workflow j with initial value equals to its total available user's budget, and is updated at each step after selecting the processor for the current task t_{curr} as shown in Eq(6). Similarly, RCA_j is defined as Remain Cheapest Assignment of workflow j with an initial value equal to $\sum\limits_{t_i \in DAG_j} Cost_{min}(t_i)$ and updated by Eq(7).

$$RB_j = RB_j - AC(t_{curr}) \tag{6}$$
$$RCA_j = RCA_j - Cost_{min}(t_{curr}) \tag{7}$$

All free available processors are filtered by $Cost_{Bound}(t_{curr})$ to guarantee that the application can be executed without exceeding the budget constraint. The set of processors is named $P_{admissible}$. In the most restricted case, only the cheapest processors are considered. Otherwise, no feasible schedule exists under the user defined budget.

Time bound: based on the workflow deadline, a sub-DeadLine is assigned to each task of the workflow. We apply the common and direct project planning sub-deadline distribution strategy. The sub-deadline value (DL) 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(8):

$$\begin{aligned} DL(t_{curr}) \;&= \min_{t_{child} \in succ(t_{curr})} \Big[DL(t_{child}) - \\ &\overline{C}_{(t_{curr} \to t_{child})} - ET_{max}(t_{child}) \Big] \end{aligned} \tag{8}$$

where ET_{max} is defined as the maximum execution time of task t_{curr} among available processors. For the exit task, the sub-deadline is equal to the user defined deadline.

Unlike the cost bound ($Cost_{Bound}$), the sub-Deadline is a soft limit as in most deadline distribution strategies on HCS platforms with a fixed number of available resources [20]; if the scheduler cannot find a processor that satisfy 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, in order 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} belonging to workflow j on each admissible processor $p_k \in P_{admissible}$, shown in (9) and (10), respectively. Both quantities are normalized by their maximum values.

$$Time_Q(t_{curr}, p_k) = \begin{cases} \dfrac{DL(t_{curr}) - FT(t_{curr}, p_k)}{DL(t_{curr}) - FT_{min}(t_{curr})} & \text{if } FT(t_{curr}, p_k) \\ & < DL(t_{curr}) \\ \dfrac{FT(t_{curr}, p_k) - FT_{min}(t_{curr})}{FT_{max}(t_{curr}) - FT_{min}(t_{curr})} \times -1 & \text{otherwise} \end{cases} \tag{9}$$

$$Cost_Q(t_{curr}, p_k) = \begin{cases} \dfrac{Cost_{max}(t_{curr}) - Cost(t_{curr}, p_k)}{Cost_{max}(t_{curr}) - Cost_{min}(t_{curr})} & \text{if } FT(t_{curr}, p_k) \\ & < DL(t_{curr}) \\ 1 - \dfrac{Cost_{max}(t_{curr}) - Cost(t_{curr}, p_k)}{Cost_{max}(t_{curr}) - Cost_{min}(t_{curr})} & \text{otherwise} \end{cases} \tag{10}$$

where $FT_{min}(t_{curr})$ and $FT_{max}(t_{curr})$ denote the minimum and maximum finish time of current task among all available processors; $Cost_{min}(t_{curr})$ and $Cost_{max}(t_{curr})$ denote the minimum and maximum execution cost of the current task among all of the processors.

$Time_Q$ measures how much closer to the task sub-deadline (DL) the finish time of current task on processor p_k is. Processors with higher $Time_Q$ values have higher possibility to be selected. If the current task has higher finish time on processor p_k than its sub-deadline, $Time_Q$ uses an alternative strategy, i.e., assigns the task to the processor with earliest finish time. Similarly, $Cost_Q$ measures how much less the actual cost on p_k is than the maximum consumed cost ($Cost_{max}$).

In the case that any of the processors from $P_{admissible}$ can *not* guarantee t_{curr} sub-deadline, $Cost_Q$ and $Time_Q$ are a negative value that represents the relative finish time and consumed cost obtained with p_k.

Finally, to select the most suitable processor for the current task, the Quality measure (Q) for each processor $p_k \in P_{admissible}$ is computed as shown in Eq(11):

$$Q(t_{curr}, p_k) = Time_Q(t_{curr}, p_k) + Cost_Q(t_{curr}, p_k) \quad (11)$$

The MW-DBS algorithm is shown in Algorithm 1. First, all ready-to-execute tasks in *ready tasks* pool are ranked by $rank_D$ priority value (Eq.3). To fill *ready tasks* pool, the system collects a single ready-to-execute task with highest primary rank value $rank_u$ [17] from each submitted and unfinished workflow application. Until there is at least one ready and unscheduled task in *ready tasks* pool and free available processors, the current ready-to-execute task t_{curr} is selected and its quality measure Q (Eq.11) is calculated among all admissible processors ($P_{admissible} \subset P_{free}$). Then, the current task t_{curr} is assigned to the processor P_{sel} that has the highest quality measure. Then the Remain unconsumed Budget(RB) and Remain Cheapest Assignment (RCA) are updated for workflow j where task t_{curr} belongs to, Eq.6 and Eq.7 respectively.

Algorithm 1 MW-DBS algorithm

```
1:  procedure MW-DBS
2:      for $t_{i,j} \in Ready\ Tasks$ pool do
3:          Assign a priority $rank_D(t_{i,j})$
4:      end for
5:      $P_{free} \leftarrow$ free processors $p_k \in P$
6:      while ($Ready\ Tasks \neq \phi$ & $P_{free} \neq \phi$) do
7:          $t_{curr} \leftarrow$ task with highest priority
8:          for $p_k \in P_{admissible}$ do
9:              Calculate Quality Measure $Q(t_{curr}, p_k)$
10:         end for
11:         $P_{sel} \leftarrow$ Processor $p_k$ with highest $Q$
12:         Assign current task $t_{curr}$ to $P_{sel}$
13:         Update $RB_j$ and $RCA_j$
14:         $P_{free} \leftarrow P_{free} - P_{sel}$
15:         Remove Task $t_{curr}$ from $Ready\ Tasks$ pool
16:     end while
17: end procedure
```

In terms of time complexity, MW-DBS requires the computation of the upward rank ($rank_u$) and sub-DeadLines (DL) for each task that have a time complexity of $O(n.p)$, 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 find and assign a suitable processor for the current task, the complexity is $O(n.p)$ for calculating FT and $Cost$ for 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 $O(n^2.p)$.

V. Experimental Results

This section presents performance comparisons of the MW-DBS algorithm. We implemented modified versions of HEFT[17], FDWS[3], MIN-MIN and MAX-MIN. The MIN-MIN and MAX-MIN algorithms have been studied extensively in the literature [15], and therefore we implemented an online version of these algorithms for our problem. We use two version of the FDWS algorithm proposed in [3], namely FDWS2 and FDWS4. The difference between these two versions of FDWS is the ranking strategy for assigning priority to ready-to-execute tasks in *ready tasks* pool. FDWS2 uses $rank_r$ which is calculated based on critical path length and FDWS4 uses $rank_B$ that is calculated based on unconsumed budget. In the processor selection phase of these algorithms, cost is not taken into account and there is a possibility to have higher cost than the limited budget defined by the user. So, in modified versions, instead of considering all processors to compute the finish time of the current task, processors are filtered ($P_{admissible}$) based on the cost limitation value defined by Eq.4 and the processor that allows the lowest finish time among all affordable processors is selected.

Based on the results presented in [4], FDWS obtains better performances, in general, in comparison to both RANK_HYBD and OWM. Therefore, in this paper, we do not consider these algorithms for comparison.

This section is divided into five parts, namely, simulation platform, budget and deadline constraints, workflow structure description, the performance metric and finally results and discussion are presented.

A. 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] [9] 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.

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 have to share bandwidth. In this experiments, we connected all processors over one shared bandwidth.

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

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

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

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 I. DESCRIPTION OF THE GRID5000 CLUSTERS FROM WHICH THE PLATFORMS USED IN THE EXPERIMENTS WERE DERIVED

B. Budget and Deadline Constraints

To evaluate MW-DBS algorithm, in our simulation we need to define a value for time and cost as deadline and budget constraint parameters for each individual workflow application. For each tested site these parameters are computed independently of the number of CPUs on that site.

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.12 and Eq.13.

$$min_{time} = \sum_{t_i \in CP} \left(ET_{min}(t_i) + \overline{C}_{(t_{parent_{CP}} \to t_i)} \right) \quad (12)$$

$$max_{time} = \sum_{t_i \in CP} \left(ET_{max}(t_i) + \overline{C}_{(t_{parent_{CP}} \to t_i)} \right) \quad (13)$$

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. Please note that both of these values are defined as lowest and highest possible makespans based on 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, in order to specify a deadline based on the lower bound of the makespan.

In the same way, to specify a budget constrain, 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 absolute highest and lowest possible costs for executing the application, that are calculated by summing the maximum and the minimum execution costs for each task, respectively.

With these highest and lowest bound values, we define a DEADLINE and a BUDGET constraint for the current application as described in Eq(14) and Eq(15):

$$\text{DEADLINE}_{user} = min_{time} + \alpha_D \times (max_{time} - min_{time}) \quad (14)$$

$$\text{BUDGET}_{user} = min_{cost} + \alpha_B \times (max_{cost} - min_{cost}) \quad (15)$$

Because of high parallelism due to the number of concurrent workflow applications compared to total available processors in our simulation, the deadline parameter α_D was selected in the range of $[1 \ldots 2]$. But for budget parameter α_B, the range considered is $[0 \ldots 1]$.

C. Workflow Structure

In order to generate dynamic and concurrent workflows scenarios, we consider 50 workflow applications, in each scenario, that arrive with time intervals that range from 10%, 30% and 50% of completed tasks, i.e., a new workflow is inserted when the corresponding percentage of tasks from the last workflow currently in the system is completed or the last one was failed. The total number of scenarios is 5000 and each scenario is tested for 9 different combinations of deadline and budget constraints.

General 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 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.

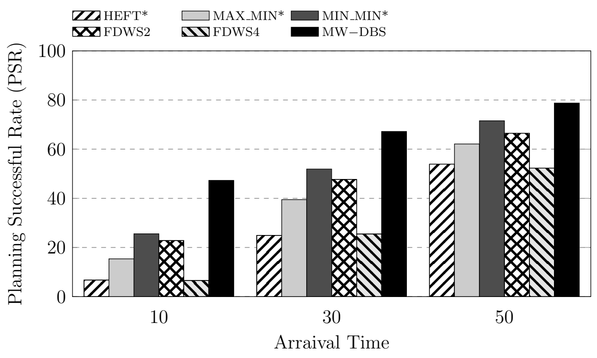
In our experiment, it was considered random DAGs with a number of tasks in the range $n = [10 \ldots 90]$, $jump = [1 \ldots 4]$. The fat, $regularity$, and $density$ are taken randomly from the range $[0.2 \ldots 0.5]$.

D. Performance Metric

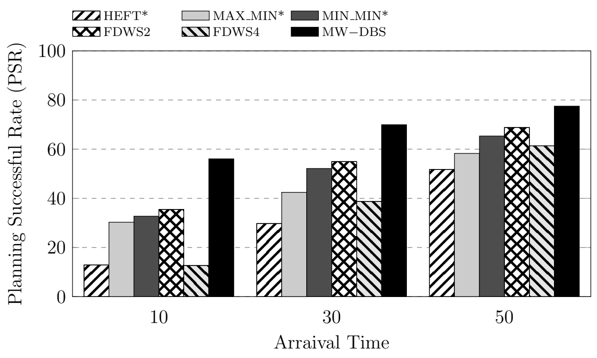
The metric to evaluate a dynamic scheduler of independent workflows, must represent the individual successful rate of finding a valid schedule map for each workflow application in the scenario in order to measure the QoS experienced. we consider the Planning Successful Rate (PSR), as expressed by Eq (16), for all workflow applications in total scenarios.

$$\text{PSR} = 100 \times \frac{\textit{Successful Planning}}{\textit{Total Number in experiment}} \quad (16)$$

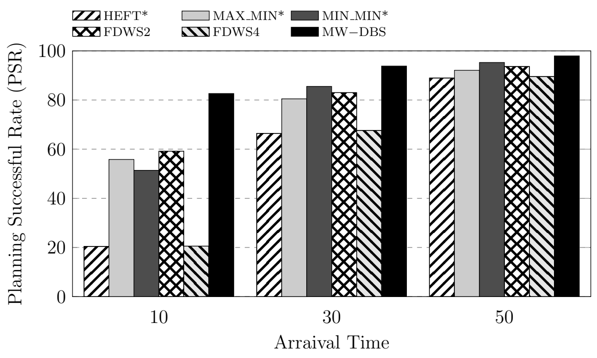
[3]https://github.com/frs69wq/daggen



(a) lille

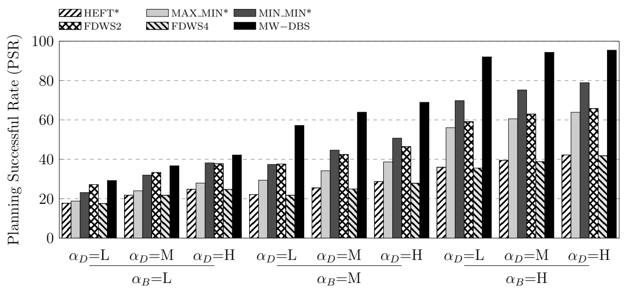


(b) rennes

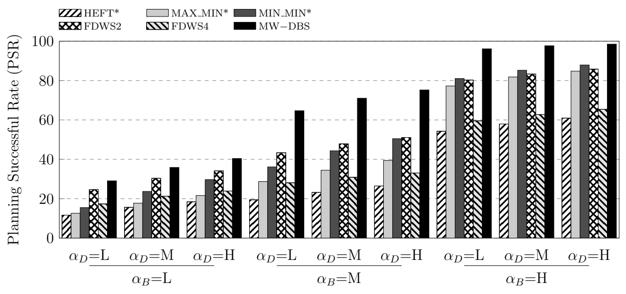


(c) sophia

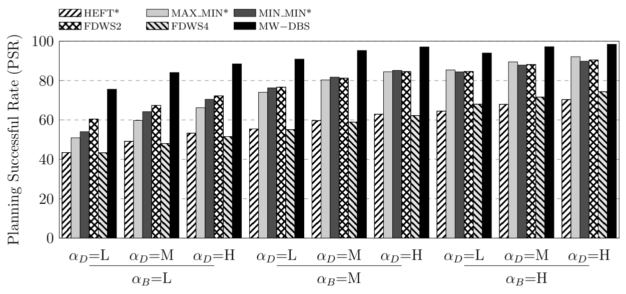
Fig. 1. PSR values of General workflow applications for different interval arrival time



(a) lille



(b) rennes



(c) sophia

Fig. 2. PSR values of General workflow applications for categorized deadline and budget constraints

E. Results and Discussion

In this section, we compare the MW-DBS algorithm with FDWS2 and FDWS4 [3] and the modified versions of HEFT [17], MIN-MIN and MAX-MIN , called HEFT*, MIN-MIN* and MAX-MIN* in shown figures. All algorithms are compared in terms of PSR (Planning Successful Rate) value. We consider a low number of processors compared to the number of DAGs to analyze the behavior of the algorithms with respect to the system load. The maximum load configuration is observed for 8 processors and 50 DAGs.

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 for different arrival time for general workflow applications.

In general, as we can see, the MW-DBS algorithm shows better performance in all sites. Also, the MAX-MIN* algorithm yielded poorer results than the MIN-MIN* algorithm in most cases. Comparing FDWS2 and FDWS4, it shows that the task selection strategy based on time minimization ($rank_r$ priority used in FDWS2) leads to better performance than cost parameter ($rank_B$ priority used in FDWS4). It also justifies the reason why we use the time factor ($rank_D$) for selecting the current task among all available ready tasks in the *ready tasks* pool.

The main advantage of MW-DBS algorithm occurred for low time intervals, that shows significant performance improvement rather than other algorithms. Increasing the time intervals between the DAGs arrival times, reduces the concurrency, and thus, the relative improvements decrease.

Figure 2 categorized the average Planning Successful Rate (PSR) for different budget (α_B) and deadline (α_D) parameters. We categorized the cost and time parameters with equal dividing the total range of each parameter in three classes namely L, M and H. As we can see, by increasing the budget value, the PSR value improves. The PSR has a softly increment with deadline growth inside of each budget category.

VI. Conclusion

In this paper, we have presented the Multi-Workflow Deadline-Budget scheduling algorithm (MW-DBS) for dynamic scheduling of concurrent workflows with two conflicting QoS requirements. To the best of our knowledge, there is no previous research that deal with multiple workflow scheduling that are submitted at different moments in time and that are based on the two conflicting QoS parameters, namely, time and cost constraints at the same time. One of the main advantages of the MW-DBS algorithm is its low complexity, that makes it suitable for usage in real infrastructures. In terms of result accuracy, we used a simulation with a realistic model of the computing platform and with shared links, as occurs in a common grid and cloud infrastructures.

The MW-DBS algorithm maps each workflow to a heterogeneous system and maximizes the global Planning Successful Rate, that is the number of workflows that can execute bounded by the user specified deadline and budget. The MW-DBS algorithm obtains better performances in all presented cases, specially for higher concurrent cases, i.e. higher arrival time.

In future work, we intend to extend the algorithm to consider more QoS parameters such as energy, reliability and fault tolerance in distributed environments.

VII. Acknowledgment

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

References

[1] H. Arabnejad and J.G. Barbosa. Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 633–639. IEEE, 2012.

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

[3] Hamid Arabnejad and JorgeG. Barbosa. Budget constrained scheduling strategies for on-line workflow applications. In *Computational Science and Its Applications ICCSA 2014*, Lecture Notes in Computer Science, pages 532–545. Springer International Publishing, 2014.

[4] Hamid Arabnejad, Jorge G. Barbosa, and Frdric Suter. *Fair Resource Sharing for Dynamic Scheduling of Workflows on Heterogeneous Systems*, chapter 9, pages 145–167. John Wiley & Sons, Inc. (Series on Parallel and Distributed Computing), 2014.

[5] L.F. Bittencourt and E. Madeira. Towards the scheduling of multiple workflows on computational grids. *Journal of Grid Computing*, 8:419–441, 2010.

[6] Klavdiya Bochenina. A comparative study of scheduling algorithms for the multiple deadline-constrained workflows in heterogeneous computing systems with time windows. *Procedia Computer Science*, 29:509–522, 2014.

[7] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[8] 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.

[9] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.

[10] Chih-Chiang Hsu, Kuo-Chan Huang, and Feng-Jian Wang. Online scheduling of workflow applications in grid environments. *Future Generation Computer Systems*, 27(6):860–870, 2011.

[11] He-Jhan Jiang, Kuo-Chan Huang, Hsi-Ya Chang, Di-Syuan Gu, and Po-Jen Shih. Scheduling concurrent workflows in hpc cloud through exploiting schedule gaps. In *Algorithms and Architectures for Parallel Processing*, pages 282–293. Springer, 2011.

[12] 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.

[13] B Arun Kumar and T Ravichandran. Time and cost optimization algorithm for scheduling multiple workflows in hybrid clouds. *European Journal of Scientific Research*, 89(2):265–275, 2012.

[14] Y.K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.

[15] Muthucumaru Maheswaran, Shoukat Ali, HJ Siegal, Debra Hensgen, and Richard F Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 30–44. IEEE, 1999.

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

[17] 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.

[18] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi. A multiple qos constrained scheduling strategy of multiple workflows for cloud computing. In *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on*, pages 629–634. IEEE, 2009.

[19] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3-4):217–230, 2006.

[20] 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.

[21] Z. Yu and W. Shi. A planner-guided scheduling strategy for multiple workflow applications. In *International Conference on Parallel Processing-Workshops (ICPP-W'08)*, pages 1–8. IEEE, 2008.

[22] H. Zhao and R. Sakellariou. Scheduling multiple DAGs onto heterogeneous systems. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–14. IEEE, 2006.

[23] 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.

[24] Amelie Chi Zhou and Bingsheng He. Transformation-based monetary cost optimizations for workflows in the cloud. *IEEE Transactions on Cloud Computing*, 2(1):85–98, 2014.