Budget-Deadline Constrained Workflow Planning for Admission Control in Market-Oriented Environments

Wei Zheng[1] and Rizos Sakellariou[2]

[1] School of Information Science and Technology, Xiamen University, China
[2] School of Computer Science, University of Manchester, UK

Abstract. There is an increasing interest for distributed computing technologies to be delivered through a market-based paradigm, which allows consumers to make use of and pay for services that meet certain Quality of Service requirements. In turn, providers receive income for successful provision of these services. In this paper, we assume an environment with multiple, heterogeneous resources, which provide services of different capabilities and of a different cost. Users want to make use of these services to execute a workflow application, within a certain deadline and budget. The problem considered in this paper is to find a plan for admission control. This allows providers to agree on constraints set by the user and allocate services for the execution of a workflow so that both deadline and budget constraints are met while account is also taken of the existing load (confirmed reservations) in the environment and the planning costs. A novel heuristic is proposed and evaluated using simulation with four different real-world workflow applications.

Keywords: Market-Oriented Computing, Workflow Execution, Workflow Planning, SLA.

1 Introduction

In market-oriented environments, such as grid or cloud platforms where resource owners provide services of different capacities and of different prices [15], users may want to use these services to execute complex applications, such as workflows [3]. Typically, a user may require his/her workflow application to complete within a certain deadline and budget; such requirements are generally recognised as Quality of Service (QoS) requirements. In analogy to markets in the real world, a Service Level Agreement (SLA) [10], which acts as a bilateral contract between a user and a service provider, is usually specified to capture the user's QoS requirements and act as a guarantee of the expected QoS. However, to establish an SLA, the service provider must have a way of determining in advance if it is feasible to fulfil a user's request. From the service provider's point of view, this implies that there is a need to find a plan for the execution of every new workflow to see if both the budget and deadline constraints requested by

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 105–119, 2012.

the user can be met according to the current load of the service provider's resources. Such a plan is called a Budget-Deadline Constrained plan, or, in short, '*BDC-plan*'. The planning procedure is called '*BDC-planning*'. BDC-planning should be part of the *admission control* of a workflow request: if a BDC-plan is found, a user's request can be accepted and a relevant SLA can be agreed; otherwise, the user's request should be rejected.

BDC-planning is an important but also remarkably challenging problem for market-oriented environments. First, such a planning problem is NP-complete [12]. Second, the non-dedicated nature of resources imposes more difficulties as the contention for shared resources (due to other, already agreed workloads) needs to be considered during planning. This suggests that the planner may have to somehow query resources for their runtime information (e.g., the existing load) to make informed decisions. Moreover, at the same time, BDC-planning should be performed in *short time*, because: (i) users may require a real-time response, and (ii) the (runtime) information, on which a planning decision has been made, varies over time and, thus, a planning decision made using out-of-date information may not be valid any more.

Essentially, the problem considered in this paper boils down to bi-criteria DAG planning, as we assume that every workflow application is represented by a Directed Acyclic Graph (DAG). This problem involves the planning process to optimize two metrics at the same time to meet the specified constraints (budget and deadline). There have been quite a few bi-criteria DAG planning heuristics in the literature [12,25,26,16,5,20,19]. However, some of them do not take the existing load of resources into account (or adopting them to do so could be too costly). Moreover, most of these heuristics have sophisticated designs, such as guided random research or local search, which usually require considerably high planning costs. Such features do not make existing heuristics particularly suitable for the BDC-planning problem discussed above (as opposed to the problem of scheduling a workflow already admitted, in which case high-cost approaches could be justified). This motivates the work presented in this paper.

In this paper, a new BDC-planning heuristic is proposed with the objective to simultaneously provide effective BDC-planning and fast planning time. The proposed heuristic is based on the Heterogeneous Earliest Finish Time (HEFT) algorithm [21], which is a well-known list scheduling heuristic aiming at minimizing the overall execution time of a DAG application in a heterogeneous environment. While being powerful at optimizing makespan, the HEFT algorithm does not consider the monetary cost and budget constraint when making scheduling decisions. In this paper, the HEFT algorithm is extended in order to resolve the BDC-planning problem and the new algorithm is named the *Budget-constrained Heterogeneous Earliest Finish Time (BHEFT)*. In the experimental section of the paper, it is demonstrated that, for the BDC-planning problem, the proposed heuristic addresses well the aforementioned challenges. In addition, it performs at least as effectively as sophisticated heuristics, but costs much less in terms of computation and communication overheads.

In the rest of this paper, related work is reviewed in Section 2. The model assumed and a problem definition are presented in Section 3. A novel BDC-planning heuristic (BHEFT) is described in Section 4. Experimental details and simulation results are discussed in Section 5. The paper is concluded in Section 6.

2 Related Work

Admission control problems have been studied in various computing platforms where QoS is considered. Yeo and Buyya [23] investigated the advance impact of inaccurate runtime estimates for deadline constrained job admission control in clusters. Yin *et al.* [24] proposed a predictive admission control algorithm to support advance reservation in equipment grids. Admission control issues were also studied as a subproblem of resource management in grids which support SLAs [9,1]. Nevertheless, none of these works takes budget requirements from users into account; moreover, their targeted applications are not workflows.

Admission control for workflows in market-oriented grids requires bi-criteria DAG planning techniques. A grid capacity planning approach is presented in [17], which aims at producing a plan for a workflow without reservation conflicts to optimize resource utilization and multiple QoS constraints. However, this paper mainly focused on a 3-layer negotiation mechanism rather than a planning heuristic itself. The studies in [14,13] proposed mapping heuristics to meet deadline constraints, at the same time minimizing the reservation cost of workflows, but they regarded workflow tasks as being multiprogramming, something not commonly adopted in workflow scheduling studies [22]. Based on the model of Utility Grids, the time-cost constrained optimization has been studied for meta-scheduling [8,6,7] in which planning is considered at application-level, but applications are assumed to be independent rather than task-based and bounded by dependencies as is the case in workflow DAGs. Therefore, although they consider both time and cost constraints in planning, these techniques are not really applicable for admission control for workflows.

To resolve the multi-objective (time and cost, commonly) DAG planning problem, evolutionary techniques (e.g., genetic algorithms) have been widely used. Examples can be found in [25,26,20,19]. Although algorithms based on evolutionary techniques normally perform well on optimization, they also require significantly high planning costs and thus are naturally too time-consuming for BDC-planning.

There are also bi-criteria scheduling heuristics for workflow applications derived from local search and list scheduling techniques. Wieczorek *et al.* [12] propose a two-phase algorithm (DCA) to address the optimization problem with two independent generic criteria for workflows in Grid environments. The algorithm optimizes the primary criterion in the first phase, then optimizes the secondary criterion while keeping the primary one within the defined sliding constraint. In [16], two scheduling heuristics based on guided local optimization, LOSS and GAIN, were developed to adjust a schedule, and these may be generated by a time-optimized heuristic or a cost-optimized heuristic, to meet

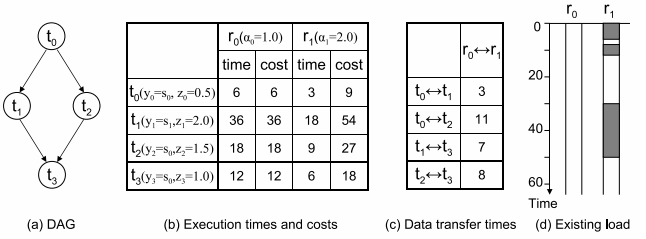
users' budget constraints. As an extension to the DLS algorithm [18], BDLS is presented in [5] which focuses on developing bi-criteria scheduling algorithms to achieve a trade-off between execution time and reliability. Based on local search, DCA and LOSS require a considerable number of repetitions to obtain a final result. As a list scheduling heuristic, BDLS may have low complexity. The main planning costs of BDLS arise from the computation of dynamic priorities when making scheduling decisions. The main issue with these heuristics is that they do not consider the existing load of resources in their assumptions, and thus tend to produce plans which may lead to reservation conflicts, i.e., given that one resource can only execute one task at a time, the planned task may overlap with the tasks of other workflows which have already been reserved. However, with an added communication phase between the planner and service providers (as mentioned in Section 3) and a slight change of algorithm design, these heuristics may be able to produce BDC-plans without reservation conflicts. In Section 5, these slightly-changed heuristics will be compared with BHEFT in terms of planning performance and overheads.

To the best of our knowledge, there is no previous study which attempts to address equally all four key elements of the problem at the same time, that is: (i) workflow planning for (ii) admission control of (iii) market-oriented environments while (iv) considering dynamically existing loads in non-dedicated resources. Unlike the aforementioned works which exhibit drawbacks according to the BDC-planning challenges mentioned in Section 1, BHEFT is a novel bi-criteria DAG planning heuristic proposed to address these challenges. By applying BHEFT, the planner of a market-oriented environment is enabled to effectively determine whether a workflow request should be accepted or not in a real-time manner so that the establishment of an SLA can be facilitated.

3 Problem Description

A workflow is modelled as a DAG consisting of a group of nodes and a set of directed edges. A node denotes a task $t_i, (1 \leq i \leq n)$, where n is the number of tasks. An edge represents a task dependency $t_i \rightarrow t_j$, where t_i is called a parent task of t_j and t_j a child task of t_i. A child task cannot be executed until all the input data depending on parent tasks have been received. Information associated with each task (t_i) are: the service type the task wants to use (y_i) and the task size (z_i). A workflow request is submitted with a budget B and a deadline D.

There is a group of service types $\mathcal{Y} = \{s_0, s_1, \cdots\}$, and a set of heterogeneous resources which are fully interconnected. A resource r_p may provide a set of service types $\mathcal{Y}_p \subseteq \mathcal{Y}$. Service instance $s_{x,p}$ exists if $s_x \in \mathcal{Y}_p$. Mapping a task to $s_{x,p}$ means allocating the task to resource r_p. Task t_i can be mapped to r_p for execution if and only if $y_i \in \mathcal{Y}_p$. Different service types may have different capacities with a different executing cost. For each service type s_x, a parameter β_x is given to depict its standard execution time, which is one of the factors to estimate the execution time of a task which uses this service type. Similarly,



(a) DAG

	$r_0(\alpha_0=1.0)$		$r_1(\alpha_1=2.0)$	
	time	cost	time	cost
$t_0(y_0=s_0, z_0=0.5)$	6	6	3	9
$t_1(y_1=s_1, z_1=2.0)$	36	36	18	54
$t_2(y_2=s_0, z_2=1.5)$	18	18	9	27
$t_3(y_3=s_0, z_3=1.0)$	12	12	6	18

(b) Execution times and costs

	$r_0 \leftrightarrow r_1$
$t_0 \leftrightarrow t_1$	3
$t_0 \leftrightarrow t_2$	11
$t_1 \leftrightarrow t_3$	7
$t_2 \leftrightarrow t_3$	8

(c) Data transfer times

(d) Existing load

Fig. 1. An example of program input

different resources may have different powers with different prices. For each resource r_p, a power ratio α_p is given to depict its power. The larger α_p is, the more powerful the resource will be. Thereby, for t_i, the execution time on r_p (i.e., $et_{i,p}$) is defined as $et_{i,p} = z_i \times \beta_x/\alpha_p$, and the execution cost $ec_{i,p} = \mu_p \times et_{i,p}$, where μ_p is the price unit for resource r_p; it is assumed $\mu_p = \alpha_p(1+\alpha_p)/2$. Also, the time to transmit data between two dependent tasks which are respectively mapped to specific (different) resources is given. Moreover, in a resource, confirmed reservations may exist. This is regarded as existing load denoted by the set of pairs $\mathcal{L} = \{(st_0, ft_0), \cdots (st_k, ft_k), \cdots\}$, where st means the start time of a reservation and ft the finish time. Here, it is assumed that only one service can run at a time on a resource. Thus, each reservation reserves a certain period of a whole resource for a task which wants to use a service instance provided by the resource.

All the above-mentioned types of input are illustrated with an example shown in Fig. 1, which includes a 4-node DAG and two resources. Every resource implements two service types s_0 and s_1, of which the standard execution time is given by $\beta_0 = 12$ and $\beta_1 = 18$. In Fig. 1(b), the parameters associated with each task and each resource are presented and used to compute execution time and cost on different resources. The data transfer times and existing load are respecitively depicted in Fig. 1(c) and Fig. 1(d).

The BDC-planning problem is to map every task onto a suitable service instance (i.e., a resource) and specify an appropriate start time for each mapped task so that the execution time and overall cost of the workflow is within D and B, respectively, and the produced plan does not overlap with existing reservations. With the same input, different heuristics may differ at deciding whether a BDC-plan can be obtained. The objective of a BDC-planning heuristic is to maximize the likelihood that a BDC-plan can be successfully found for a given workflow request.

It is worth mentioning that the planner has to communicate with resource owners to produce a plan without reservation conflict. We assume that the planner has to send a *Time Slot Query* (TSQ), i.e., ask for a certain length of time slot on a specific resource, and then the resource owner responds with the earliest availability. Here, the alternative of allowing the planner to retrieve all free time slots of resources is not considered, since the service providers may not want their workload, which may be commercially sensitive, to be exposed.

Input: DAG G with Budget B and Deadline D;
Output: A BDC-plan

1. Compute *rank* (as defined in Eq.(1)) for all tasks.
2. Sort all tasks in a planning list in the non-ascending order of *rank*.
3. **for** $k := 0$ to n **do** (where n is the number of tasks)
4. Select the kth task from the planning list.
5. Compute the Spare Application Budget for task k (as defined in Eq.(2)).
6. Compute the Current Task Budget for task k (as defined in Eq.(3)).
7. Construct the set of Affordable Services (as defined in Eq.(4)) for task k.
8. **for** each service which can be used by task k **do**
9. Compute the earliest finish time of mapping task k to the service using TSQ as described in Section 3.
10. **endfor**
11. Select a service for task k according to the defined selection rules.
12. **endfor**

Fig. 2. The BHEFT Heuristic

Let $\mathcal{L}_p$ be the existing load of resource r_p, we define TSQ in the form of $f_Q(t_i, r_p, dat_{i,p}, dur) = \min\{(a,b)|(a,b) \cap \mathcal{L}_p = \emptyset, a \geq dat_{i,p}, b = a + dur\}$, where $dat_{i,p}$ means the time all required data is available for task t_i on resource r_p, and dur denotes the required duration which is considered to be equal to the estimated execution time $et_{i,p}$. According to Fig. 1, $\mathcal{L}_1 = \{(0,6),(8,12),(30,50)\}$ and for task 0, $dat_{0,1} = 0$ and $et_{0,1} = 3$, then it holds that $f_Q(0,1,0,3) = (12,16)$.

With TSQ, there are two ways for a planning heuristic to avoid reservation conflicts. One is invoking TSQ every time when computing the estimated earliest finish time for a task on a resource. A planning heuristic, such as DCA [12] or LOSS [16], normally involves lots of such estimates, and thus may introduce heavy communication costs if using this approach. The other way is producing an initial plan without considering the existing reservations and then using TSQ to reallocate the time slot for each mapped tasks in the order that tasks are initially scheduled. In this case, the communication costs may be small but the performance of the heuristic may degrade.

4 The Proposed Heuristic

This section describes the details of BHEFT, of which the outline is shown in Figure 2. Similar to the original HEFT algorithm, the BHEFT also has two major phases: *task prioritizing* and *service selection*.

In the task prioritizing phase, the priorities of all tasks are computed using upward ranking which is the same as defined in HEFT. The rank of a task i is recursively defined by

$$rank_i = \overline{et}_i + \max_{j \in Succ(i)} \{\overline{dt}_{i,j} + rank_j\} \quad (1)$$

where $Succ(i)$ is the set of the child tasks of task i, $\overline{et}_i$ is the average execution time of task t_i, $\overline{dt}_{i,j}$ is the average data transfer time of edge $t_i \rightarrow t_j$. In the case of childless nodes, the rank equals to the average execution time.

In the service selection phase, the tasks are selected in order of priority. Each selected task is allocated to its "best possible" service, of which the metric may change according to an assessment of the spare budget which varies as planning proceeds. For this assessment, two variables are used: *Spare Application Budget* (SAB) and *Current Task Budget* (CTB). Suppose that the kth task is being allocated, SAB_k and CTB_k are respectively computed by

$$SAB_k = B - \sum_{i=0}^{k-1} c_i - \sum_{j=k}^{n-1} \overline{c}_j \tag{2}$$

$$CTB_k = \begin{cases} \overline{c}_k + SAB_k \times \overline{c}_k / \sum_{i=k}^{n-1} \overline{c}_i & : \quad SAB_k \geq 0 \\ \overline{c}_k & : \quad SAB_k < 0 \end{cases} \tag{3}$$

where B is the given budget, c_i is the reservation cost of the allocated task i, $\overline{c}_j$ is the average reservation cost of the unallocated task j over different resource mappings, n is the number of tasks. Provided that task t_k uses service type s_x, a set $\mathcal{S}_k^*$ is constructed consisting of an *affordable service* for task k, i.e.,

$$\mathcal{S}_k^* = \{s_{x,p} | \exists s_{x,p}, c_{k,p} \leq CTB_k\} \tag{4}$$

Then the "best possible" service is selected by the selection rules as follows:

1. If $\mathcal{S}_k^* \neq \emptyset$, the affordable service with the earliest finish time is selected;
2. If $\mathcal{S}_k^* = \emptyset$ and $SBA \geq 0$, the service with the earliest finish time selected;
3. If $\mathcal{S}_k^* = \emptyset$ and $SBA < 0$, the cheapest service is selected;

Using the example presented in Fig. 1 and assuming a budget $B = 89$ and a deadline $D = 60$, the planning results derived by using the above-described heuristic are shown in Fig. 3, where tasks are planned in the order t_0, t_1, t_2, t_3 and a BDC-plan is successfully obtained.

5 Performance Evaluation

5.1 Experimental Setting

To run the experiments, a job planner (broker) and a set of resources were simulated by java programs distributed on computing nodes with 3.0 Ghz CPU, 1 GB memory and connection through Gigabit Ethernet. The communication between the broker and the service providers was implemented by socket programming. The existing load of resources was also randomly generated for simulation. Given a specific period between time a and b, the existing load of each resource p (i.e., $\mathcal{L}_p$) is parameterized by two pre-specified values: Utilization Rate (UR) and Average Task Load (ATL). The former is the ratio of the total reserved time to the whole period, and the latter is the ratio of the number of tasks appearing during a certain period to the length of this period. Then, the average duration

	Avg. Time $\overline{et_i}$	Avg. Cost $\overline{c_i}$	Rank	SAB	CTB	S*	Selected Service	Start Time	Finish Time	Actual Cost c_i
t_0	4.5	7.5	50.5	-1	7.5	ϕ	$\{s_{0,0}\}$	0	6	6
t_1	27	45	43	0.5	45.27	$\{s_{1,0}\}$	$\{s_{1,0}\}$	6	42	36
t_2	13.5	22.5	30.5	9.5	28.2	$\{s_{0,0}, s_{0,1}\}$	$\{s_{0,1}\}$	17	26	27
t_3	9	15	9	5	20	$\{s_{1,0}, s_{1,1}\}$	$\{s_{1,0}\}$	42	54	12
The given budget is 89, and Sum =										81

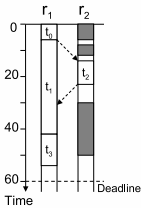


Fig. 3. Planning derived using BHEFT (Avg.=Average)

of a reservation slot is $\overline{RD} = UR/ATL$ and the average duration of an idle slot is $\overline{ID} = (1 - UR)/ATL$.

The following procedure describes how the existing load of resource p ($\mathcal{L}_p$) was constructed: (1) Set $\mathcal{L}_p = \emptyset$ and current time $CT = a$. (2) Randomly determine current state among RESERVED and IDLE. (3) If RESERVED: (3.a) randomly generate reserved duration RD by normal distribution with mean $\overline{RD}$ and standard deviation $\overline{RD}/2$ whereas only $RD > 0$ is adopted; (3.b) set $\mathcal{L}_p = \mathcal{L}_p \cup (CT, CT + RD)$; (3.c) set $CT = CT + RD$; (4.d) switch current state to IDLE. (4) If IDLE: (4.a) randomly generate idle duration ID by normal distribution with mean $\overline{ID}$ and standard deviation $\overline{ID}/2$ whereas only $ID > 0$ is adopted; (4.b) set $CT = CT + ID$; (4.c) switch current state to RESERVED. (5) Repeat Steps 3 and 4 till CT reaches b.

There were 2 service providers in the evaluation, each of which managed 3 resources, hence, there were 6 resources in total. There were 4 service types having a standard execution time of 10, 15, 25 and 30 respectively. For each resource p, the capability ratio α_p was randomly generated from the interval [0.5, 2.0]. The period considered for existing load modelling was [0, 5000].

Four types of DAGs, corresponding to real-world workflow applications, were considered in the experiments; these are: fMRI [27] with 17 nodes, Montage [2] with 34 nodes, AIRSN [11] with 53 nodes and LIGO [4] with 77 nodes. For each task i, y_i was randomly selected from the provided service types, and z_i was randomly generated from [0.5, 2.0]. The communication computation ratio (CCR) was randomly selected from [0.1, 1.0].

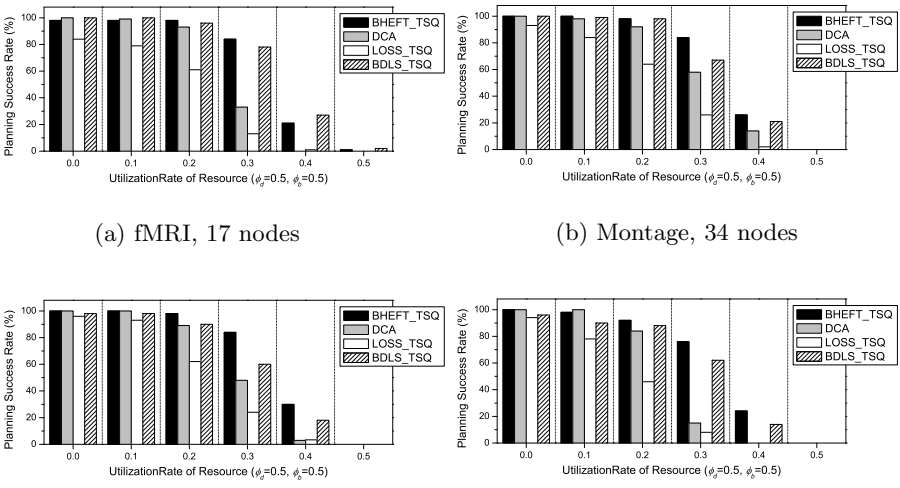
Given a DAG, constraints for reasonable values for deadline and budget were generated as follows. For simplicity, a job was always assumed to start at time 0. The makespan M_{HEFT} was computed by applying the HEFT algorithm [21] to the DAG without considering the existing load of resources. The deadline constraint DC was considered to be located between the lower bound $LB_{dc} = M_{HEFT}$ and the upper bound $UB_{dc} = 5 \times M_{HEFT}$. A deadline ratio ϕ_d was used to depict the position of DC by $DC = LB_{dc} + \phi_d \times (UB_{dc} - LB_{dc})$, where $0 \leq \phi_d \leq 1.0$. For budget constraint, LB_{bc} was the lowest total cost obtained by mapping each task to the cheapest service, and UP_{bc}, the highest total cost obtained conversely. Similarly, a budget ratio ϕ_b was used to specify the possible budget constraint $BC = LB_{bc} + \phi_b \times (UB_{bc} - LB_{bc})$, where $0 \leq \phi_b \leq 1.0$.

BHEFT was compared with DCA [12], LOSS [16] and BDLS [5] in the experiments. As mentioned in Section 3, some modification is needed to adapt these heuristics, which do not consider the existing load of resources, to produce a contention-free plan. According to the evaluation in [12], where existing loads on resources (and hence TSQ) are not considered, DCA, which is based on extensive local search, has the best optimization performance but the highest time overhead, as opposed to BDLS which is a static list scheduling heuristic using a dynamic priority. Therefore, TSQ was introduced into LOSS and BDLS only, while DCA was modified in the other way mentioned in Section 3 (that is, a plan is first generated without considering existing loads and, then, TSQ is used to reallocate the time slot allocated to each task to resolve reservation conflicts). When showing the experimental results in figures, the suffix '_TSQ' was added to the names of the algorithms which used TSQ, to distinguish them from DCA which does not consider TSQ, while the original names are used for short in the discussion. In terms of the configuration of DCA and BDLS, the same settings as used in [12] are adopted, i.e., LOSS3 in [16] is adopted to represent LOSS, a memorization table consisting of 100 cells with up to 10 intermediate solutions stored in each cell was used by DCA, and the parameter δ for BDLS was determined by a binary search with a maximum of 15 loop iterations. Moreover, all heuristics terminate immediately when a BDC-plan is found.

For each experiment, all of the parameters except for those which were given and fixed, were re-initialized at random with the above specifications. After a heuristic was run, if a BDC-plan was found, the planning succeeded, otherwise, a failure was reported. To analyze the performance of each heuristic, the experiment was repeated multiple times and the metric *Planning Success Rate* (PSR) was used, as defined below:

$$PSR = 100 \times \frac{\text{number of times for which a BDC-plan was found}}{\text{number of total repeated times of experiment}} \tag{5}$$

Four sets of experiments were carried out. In the first one, ϕ_d and ϕ_b were fixed to be 0.5, while UR was varied for each resource from 0.0 to 0.6 in the step of 0.1 with the corresponding $ATL = 0.05 \times UR$. The experiment was repeated 500 times to observe how the existing load of resources affected the PSR of each heuristic. In the second set of experiments, UR was randomly generated in the interval [0.1, 0.4], and the ATL was computed correspondingly. ϕ_d and ϕ_b were selected from the set {0.25, 0.5, 0.75} to form 9 combinations which covered a wide spectrum of diverse user requests; the experiment was then repeated 500 times for each combination. Thus, the value of PSR was investigated under various constraints (from tight to relaxed). In the third set of experiments, we studied the same 9 combinations for user requests but for three specific values of UR. Finally, in the fourth experiment, the average running time of each heuristic to do planning was measured. This experiment was repeated 100 times for each workflow with various combinations of constraints.



(a) fMRI, 17 nodes

(b) Montage, 34 nodes

(c) AIRSN, 53 nodes

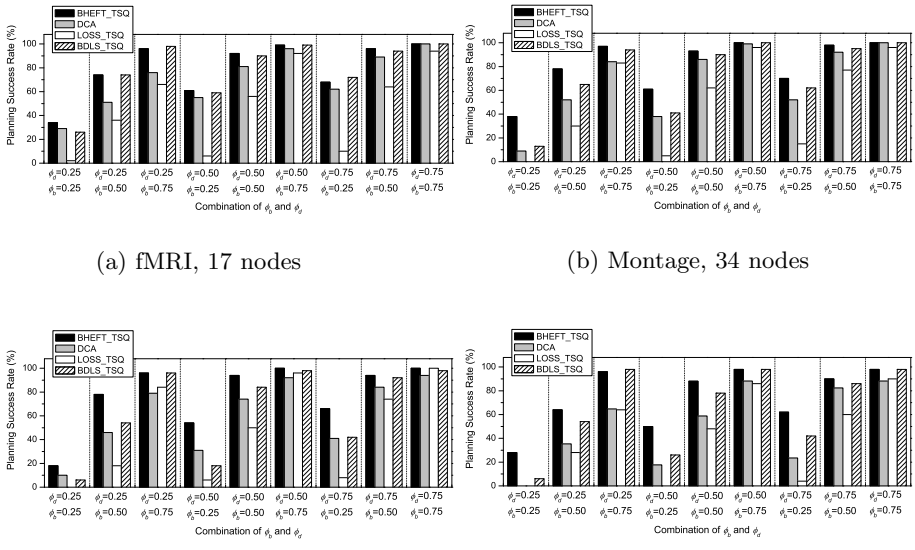
(d) LIGO, 77 nodes

Fig. 4. *PSR* with different utilization rate of resources

5.2 Experimental Results

First set of experiments: Figure 4 shows the results of the first set of experiments where the impact of the existing load of resources is investigated. Here, ϕ_d and ϕ_b are both fixed to be 0.5 to avoid unnecessary disturbance caused by setting the user constraints to be too tight or too relaxed. It can be seen from Figure 4 that the behaviour of the compared heuristics in terms of their *PSR* follows the same pattern regardless of the type of DAG. BHEFT almost always shows the best performance, with a notable exception in the case of fMRI, which could be attributed to its small number of nodes. As expected, all heuristics perform worse as *UR* increases. In such cases, the better performance exhibited by BHEFT is more profound in the graphs. Its performance is followed by BDLS, which appears to be second best outperforming LOSS and DCA. It is noted that this performance classification changes (or differences become less clear) when there is no existing load on resources (as also observed in [12] where LOSS seems to give better performance than BDLS).

Second set of experiments: In the second set of experiments, the performance of each heuristic was investigated under various circumstances of user constraints, from tight to relaxed. As already mentioned we considered nine combinations of different types of constraints. Figure 5 shows the value of *PSR* for different types of DAG and different budget-deadline constraints. The first observation is that when both the deadline constraint and the budget constraint are tight, for example, ϕ_d=0.25 and ϕ_b=0.25, all four heuristics obtain low *PSR*s; among them, BHEFT achieves the best *PSR* which is between 20% to 40%. When a small DAG (e.g., fMRI) is used, both DCA and BDLS obtain *PSR*s which are



(a) fMRI, 17 nodes

(b) Montage, 34 nodes

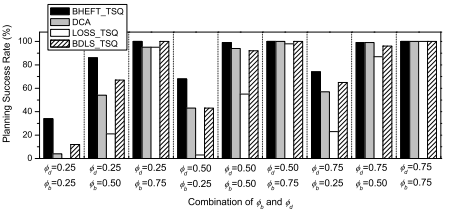
(c) AIRSN, 53 nodes

(d) LIGO, 77 nodes

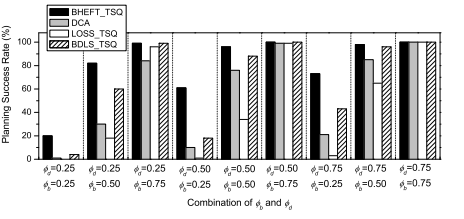
Fig. 5. *PSR* with different types of constraints

comparable to those of BHEFT. However, their *PSR*s turn significantly lower when a DAG such as LIGO is used. It is interesting to note that the performance of LOSS is particularly poor when the budget constraint is tight. This may be because the initial plan of LOSS, constructed using HEFT, usually has a small makespan regardless of the monetary cost; then, it may not be straightforward for LOSS to adjust the plan to meet the budget constraint with a limited number of local searches. BDLS can be almost as effective as BHEFT in many cases, for example, when both budget and deadline constraints are above 50%. In the case where a small DAG (e.g., fMRI) is used, BDLS can occasionally achieve a better *PSR* than BHEFT. However, overall, in most cases, BHEFT performs better than BDLS. The advantage of BHEFT is more profound when at least one of the constraints is tight and the used DAG has a large number of nodes.

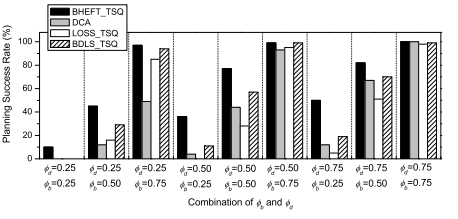
Third set of experiments: In order to consider the impact of the Utilization Rate in more detail, we studied the *PSR* for the nine different combinations of user constraints and three different values of utilization rate. The results, for two types of DAG, Montage and LIGO, are shown in Fig. 6. Once again, BHEFT performed the best among the competitive heuristics in most of the circumstances. The results highlight the impact that the existing load of resources may have on BDC-planning. As expected, when the Utilization Rate is low, that is, there is little existing load on resources, and the constraints for budget and deadline are relaxed (e.g., ϕ_d=0.75 and ϕ_b=0.75), all heuristics perform equally well.



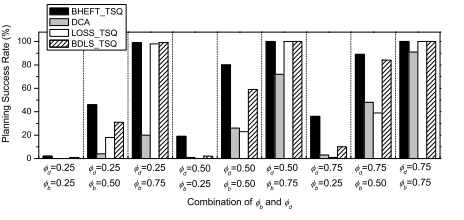
(a) Montage, UtilizationRate = 0.2



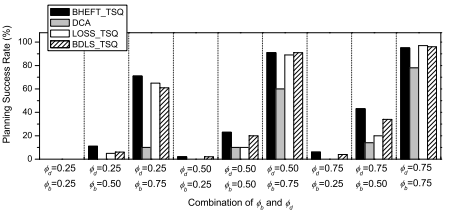
(b) LIGO, UtilizationRate = 0.2



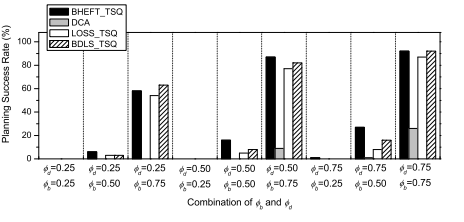
(c) Montage, UtilizationRate = 0.3



(d) LIGO, UtilizationRate = 0.3



(e) Montage, UtilizationRate = 0.4



(f) LIGO, UtilizationRate = 0.4

Fig. 6. *PSR* with different utilization rates and constraints for Montage and LIGO

Fourth experiment: In the fourth experiment, the execution time needed by each algorithm to obtain a planning result was studied. Figure 7 shows how the running time of each heuristic varies over diverse types of DAG and constraint settings. It is not surprising that, in most of the cases, LOSS has the highest time costs due to the overhead caused by numerous TSQs. It can be easily imagined that some other sophisticated algorithms, such as DCA or genetic algorithms, if using TSQ when scheduling, may need even more time compared to LOSS. Our results suggest that even LOSS is not scalable to large applications and too time-consuming for on-line workflow planning. Although not using TSQ, the DCA heuristic considered in the experiment still has an execution time comparable to BDLS, and this is significantly higher than BHEFT. The latter two algorithms are both based on list scheduling, whereas BHEFT needs evidently less running time than BDLS due to simpler computation and the fact that less communication is needed when making scheduling decisions. Moreover, BHEFT is the most scalable in terms of the growth of DAG size (and potentially the

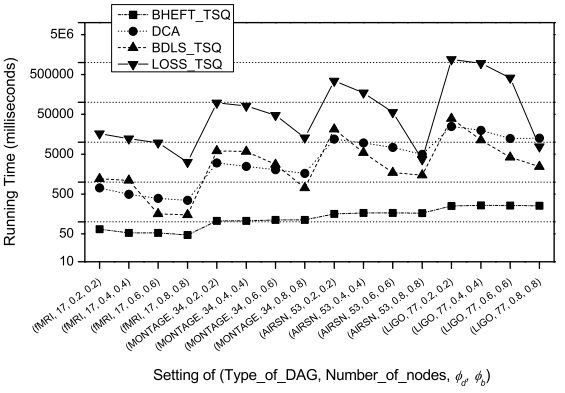


Fig. 7. Execution time for each heuristic with different DAGs and user constraints

number of resources which is considered constant in this experiment). As can be seen in the graph, when planning LIGO with 77 nodes on 6 resources, BHEFT only needs around 0.2 seconds on average. This suggests that BHEFT copes well with the real-time requirements of workflow planning.

Summary of observations: The results lead to the following observations:

- The existing load of resources may have significant impact on BDC-planning. Directly applying a heuristic not considering the existing load of resources in job planning (e.g., DCA) may result in a significant degradation of *PSR*. In contrast, BHEFT, which takes the existing load of resources into account, is able to achieve a significant improvement on the success rate of finding a BDC-plan which simultaneously satisfies deadline and budget constraints.
- Some guided local search heuristics (for example, LOSS) may be too sensitive regarding the existing load of resources and cannot perform reasonably well for BDC-planning, even when the existing load of resources is taking into account when making planning decisions.
- In the context of BDC-planning, simple list scheduling bi-criteria heuristics (for example, BHEFT and BDLS) may be as effective as more sophisticated heuristics based on extensive local search, such as DCA.
- With low running cost, BHEFT seems to be a good choice for BDC-planning.

6 Conclusion and Future Work

BDC-planning is required to establish an SLA in order to provide a certain level of QoS for workflow execution in market-based environments. This paper proposed BHEFT, a novel low-cost bi-criteria heuristic based on HEFT, to fulfill the specific requirements of BDC-planning. The experimental results suggest that BHEFT appears to be at least as effective, or even more so than other existing

sophisticated bi-criteria workflow scheduling heuristics, and has a lowest execution time cost and good scalability. It also appears that BHEFT can effectively and efficiently find a BDC-plan under various circumstances of constraints. This enables a quick admission control decision (i.e., a judgement of whether or not the submitted user request is acceptable), and provides the feasibility of automating the creation of an SLA over diverse user constraints. Based on the work in this paper, our future work will try more experiments using different applications and platforms and will consider the overestimation of task execution time in BDC-planning to cope with prediction uncertainty. In addition, we think it is worth investigating BDC-planning with more sophisticated pricing policies.

References

1. Almeida, J., Almeida, V., Ardagna, D., Cunha, I., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing 4(70), 344–362 (2010)
2. Berriman, G.B., Good, J.C., Laity, A.C., Bergou, A., Jacob, J., Katz, D.S., Deelman, E., Kesselman, C., Singh, G., Su, M.H., Williams, R.: Montage: A grid enabled image mosaic service for the national virtual observatory. In: The Conference Series of Astronomical Data Analysis Software and Systems XIII, ADASS XIII (2004)
3. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems 25, 528–540 (2009)
4. Deelman, E., Kesselman, C., Mehta, G., Meshkat, L., Pearlman, L., Blackburn, K., Ehrens, P., Lazzarini, A., Williams, R., Koranda, S.: GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In: High Performance Distributed Computing (HPDC 2002), pp. 225–234 (2002)
5. Dõgan, A., Özgüner, R.: Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. The Computer Journal 3(48), 300–314 (2005)
6. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility grids: Time and cost trade-off management. In: Thirty-Second Australasian Computer Science Conference (ACSC 2009), vol. 91, pp. 139–147 (2009)
7. Garg, S.K., Buyya, R., Siegel, H.J.: Time and cost trade-off management for scheduling parallel applications on utility grids. Future Generation Computer Systems 8(26), 1344–1355 (2010)
8. Garg, S.K., Konugurthi, P., Buyya, R.: A linear programming driven genetic algorithm for meta-scheduling on utility grids. In: Proceedings of the 16th International Conference on Advanced Computing and Communication, ADCOM 2008 (2008)
9. Han, Y., Youn, C.: A new grid resource management mechanism with resource-aware policy administrator for SLA-constrained applications. Future Generation Computer Systems 7(25), 768–778 (2009)
10. Hiles, A.: Service level agreements: measuring cost and quality in service relationships. Chapman & Hall (1993)
11. Horn, J.V., Dobson, J., Woodward, J., Wilde, M., Zhao, Y., Voeckler, J., Foster, I.: Grid-based computing and the future of neuroscience computation. Methods in Mind (2005)
12. Prodan, R., Wieczorek, M.: Bi-criteria scheduling of scientific grid workflows. IEEE Transactions on Automation Science and Engineering 7, 364–376 (2010)

13. Quan, D.M.: Mapping Heavy Communication Workflows onto Grid Resources Within an SLA Context. In: Gerndt, M., Kranzlmüller, D. (eds.) HPCC 2006. LNCS, vol. 4208, pp. 727–736. Springer, Heidelberg (2006)
14. Quan, D.M., Kao, O.: Mapping Workflows onto Grid Resources Within an SLA Context. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 1107–1116. Springer, Heidelberg (2005)
15. Risch, M., Altmann, J., Guo, L., Fleming, A., Courcoubetis, C.: The GridEcon Platform: A Business Scenario Testbed for Commercial Cloud Services. In: Altmann, J., Buyya, R., Rana, O.F. (eds.) GECON 2009. LNCS, vol. 5745, pp. 46–59. Springer, Heidelberg (2009)
16. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: Gorlatch, S., Danelutto, M. (eds.) Integrated Research in GRID Computing, pp. 189–202. Springer, Heidelberg (2007)
17. Siddiqui, M., Villazon, A., Fahringer, T.: Grid capacity planning with negotiation-based advance reservation for optimized QoS. In: Proceedings of the 2006 IEEE/ACM Conference in Supercomputing (SC 2006), pp. 103–118 (2006)
18. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transactions on Parallel and Distributed Systems 2(4), 175–187 (1993)
19. Singh, G., Kesselman, C., Deelman, E.: A provisioning model and its comparison with best-effort for performance-cost optimiazation in grids. In: Proceedings of the 16th International Symposium on High Performance Distributed Computing, pp. 117–126 (2007)
20. Talukder, A.K.M., Kirley, M., Buyya, R.: Multi-objective differential evolution for scheduling workflow applications on global grids. Concurrency and Computation: Practice and Experience 21(13), 1742–1756 (2009)
21. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 3(13), 260–274 (2002)
22. Wieczorek, M., Hoheisel, A., Prodan, R.: Taxonomies of the multi-criteria grid workflow scheduling problem. In: Proceedings of the CoreGRID Workshop on Grid Middleware (2007)
23. Yeo, C.S., Buyya, R.: Managing risk of inaccurate runtime estimates for deadline constrained job admission control in clusters. In: Proceedings of the 35th International Conference on Parallel Processing (ICPP 2006), pp. 451–458 (2006)
24. Yin, J., Wang, Y., Hu, M., Wu, C.: Predictive admission control algorithm for advance reservation in equipment grid. In: Proceedings of IEEE International Conference on Service Computing (SCC 2008), pp. 49–56 (2008)
25. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming 14, 217–230 (2006)
26. Yu, J., Buyya, R.: Multi-objective planning for workflow execution on grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (2007)
27. Zhao, Y., Wilde, M., Foster, I., Voeckler, J., Jordan, T., Quigg, E., Dobson, J.: Grid middleware services for virtual data discovery, composition, and integration. In: The 2nd Workshop on Middleware for Grid Computing (2004)