J Grid Computing (2013) 11:633–651
DOI 10.1007/s10723-013-9257-4

Budget-Deadline Constrained Workflow Planning for Admission Control

Wei Zheng · Rizos Sakellariou

Received: 24 October 2012 / Accepted: 6 May 2013 / Published online: 23 May 2013

Abstract 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 feasible plan for the execution of the workflow which would allow providers to decide whether they can agree with the specific constraints set by the user. If they agree to admit the workflow, providers can allocate services for its execution in a way that both deadline and budget constraints are met while account is also taken of the existing load in the provider's environment (confirmed reservations from other users whose requests have been accepted). A novel heuristic is proposed and evaluated using simulation with four different real-world workflow applications.

Keywords Admission control · Bi-criteria DAG scheduling · SLA-based resource reservation · Workflow planning

1 Introduction

In Grid or cloud platforms where resource owners provide services of different capacities and/or of different prices [4, 18, 23], users may want to use these services to execute complex applications, such as workflows [5, 16]. 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 the real world, a Service Level Agreement (SLA) [14], which can be regarded 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. If the terms of the SLA are fulfilled, the user is expected to pay some fee to the provider. Conversely, if the terms of the SLA are not fulfilled, the provider may have to pay some penalty to the user, as prescribed by the SLA.

The first author is supported by the National Natural Science Foundation of China (Grant No.61202361) and the Fundamental Research Funds for the Central Universities (Grant No.2011121049).

W. Zheng
School of Information Science and Technology,
Xiamen University, Xiamen, China
e-mail: zhengw@xmu.edu.cn

R. Sakellariou (✉)
School of Computer Science,
University of Manchester, Manchester, UK
e-mail: rizos@cs.man.ac.uk

Thus, 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 requesting admission to and execution on the provider's resources. Such a plan will identify feasible reservation slots on the resources for every task of the workflow in order to provide an assurance that both the budget and deadline constraints requested by the user can be met according to the current load of the service provider's resources. We call such a plan a Budget-Deadline Constrained plan, or, in short, *BDC-plan*. The planning procedure to find such a plan is called *BDC-planning*. BDC-planning should be part of the *admission control* of a user's request to execute a workflow. 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 important for service- and market-oriented environments, as its outcome drives admission control and determines whether a user's request should be accepted. BDC-planning is also a remarkably challenging problem. First, such a planning problem is NP-complete [19]. Second, the non-dedicated nature of resources imposes more difficulties as the contention for shared resources (some of which are assigned to or reserved by 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.

The general form of the BDC-planning problem 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 [7, 19, 24, 29, 31, 37, 38]. However, some of them do not take the existing load of resources into account (or modifying 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 easily justified). The need for fast and efficient heuristics, suitable for the specific problem of BDC-planning, which also takes into account existing load of the resources, motivates the work presented in this paper.

In the 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 [32], 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 effective 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 called 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 well compared to sophisticated heuristics, but costs much less in terms of computation and communication overheads.

This paper is an extended version of a paper that first appeared in [40]. The main changes made in this paper include: (1) a detailed description of the BDC-planning model with a sequence diagram to depict the BDC-planning procedure has been added; (2) a more sophisticated example to illustrate the proposed heuristic has been included; (3) a more flexible model to calculate

execution time and cost without the restriction of the notion of power (Section 3 in [40]); (4) two of the four DAGs used in the evaluation in [40], were replaced by DAGs with a larger number of nodes to allow the investigation of a wider spectrum of different workflow types.

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), as well as an illustrative example, 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 [35] investigated the advance impact of inaccurate runtime estimates for deadline constrained job admission control in clusters. Yin et al. [36] 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 [1, 13]. Nevertheless, none of these papers takes budget requirements from users into account; moreover, the applications they target are not workflows. Cost and deadline constrained admission control for workflows in IaaS clouds was studied in [17], where algorithms for both task scheduling and resource provisioning were proposed and assessed. However, the resource model considered in these algorithms consists of homogeneous virtual machines.

Admission control for workflows in market-oriented Grids requires bi-criteria DAG planning techniques. A Grid capacity planning approach is presented in [27], 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 [21, 22] proposed mapping heuristics to meet deadline constraints, at the same time minimizing the reservation cost of workflows, but they assumed that workflow tasks can be multiprogramming, something not commonly encountered in workflow scheduling studies [33]. Based on the model of Utility Grids, the time-cost constrained optimization has been studied for meta-scheduling [9–11] 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 [29, 31, 37, 38]. 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. Even though an accelerated genetic algorithm has been proposed for multi-criteria job scheduling in Grid environments [12], the application model and constraints are different to our work. Another multi-objective scheduling effort for heterogeneous environments is presented in [8], where a multi-objective list scheduling (MOLS) algorithm is proposed to find a solution which dominates or converges to a constraint vector (a set of constraint values specified by the user for several objectives). MOLS differs from our work on the aim the algorithm wants to achieve. More specifically, the aim of MOLS is to find a dominant solution by using Pareto relations. In contrast, our proposed heuristic focuses on maximizing the likelihood that a BDC-plan can be found in the presence of varying user constraints and existing loads.

There are also bi-criteria scheduling heuristics for workflow applications derived from local search and list scheduling techniques. Wieczorek et al. [19] 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 [24], two scheduling heuristics based on guided local optimization, LOSS and GAIN, were proposed to adjust a schedule; these may be based on a time-optimized heuristic or a cost-optimized heuristic, to meet users' budget constraints. As an extension to the DLS algorithm [28], BDLS, presented in [7], 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.

However, the key 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. With an added communication phase between the planner and service provider (as will be described in Section 4.3) and a slight change in algorithm design, these heuristics may be modified in order to produce BDC-plans without reservation conflicts. In Section 5, such modified heuristics will be compared with our proposed heuristic, BHEFT, in terms of both planning performance and overhead.

To the best of our knowledge, there is no previous study which attempts to address equally all four key elements of the BDC-planning 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 with respect 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

Given a workflow request with budget B and deadline D, the BDC-planning problem is to map every workflow task onto a suitable service instance (i.e., a resource) and specify an appropriate start time for each mapped task so that the overall cost and execution time of the workflow are within B and D, respectively. Of course, such a produced plan cannot overlap with existing reservations. We note that finding an appropriate start time for every task results in a reservation for every task on a specific resource; then, the whole plan consists of a set of reservation slots for all the tasks. Such an approach for task execution is commonly used in practice [20, 26, 29, 30, 34, 37, 39]. As explicitly shown in [39], such reservations may also include some slack to provide ample time for the successful execution of each task.

From the service provider's perspective, there is an incentive to maximize the number of workflow requests that are serviced. Thus, as long as a BDC-plan can be found to satisfy the constraints of a new request to execute a workflow, it is expected that the provider's admission control will give consent to the admission of a request. Therefore, a key objective of a BDC-planning heuristic is to maximize the likelihood that a BDC-plan can be successfully found for a given workflow request, which, in turn, can maximize the acceptance ratio of admission control for the provider.

3.1 Notation and Assumptions

The notation used and the assumptions made to solve the BDC-planning problem are summarized below:

- A Directed Acyclic Graph (DAG) G is used to represent the submitted workflow application. A DAG consists of a set of nodes V, each of which denotes a workflow task, and a set of edges E, each of which denotes a dependency between two dependent tasks.
- A set of resources, which are assumed to be heterogeneous (that is, of different capacities), is given. It is also assumed that each

resource provides a set of service types and each task is associated with a particular service type. For simplicity, it is assumed that each resource can provide every type of service, hence it can serve any task of the DAG. Thus, when task t_i runs on resource r_j, it means that task t_i uses the service $s_{i,j}$ provided by resource r_j.

- For each task of the DAG, t_i, on each resource, r_j, an estimated execution time, denoted by $et_{i,j}$, is known. For each resource, the price of running on the resource, denoted by p_j, is also known. Then, the cost of running t_i on r_j, denoted by $cost_{i,j}$, can be calculated by $cost_{i,j} = et_{i,j} \times p_j$. In addition, the amount of data that needs to be transmitted between tasks is known, as well as the transmission time per data unit between resources. So the data transmission time, denoted by dt, between any two allocated tasks can be determined.
- A user specifies a deadline (that is, the time by which the whole DAG/workflow must finish), denoted by D, and budget (that is, the maximum cost that the user is willing to pay), denoted by B. In real practice, users may specify an earliest start time as well as a latest finish time. In our setting, without loss of generality, we can assume that the execution of the application starts at time zero.
- In every 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 denotes the start time of a reservation and ft denotes the finish time of this reservation. Here, it is assumed that only one service can run at a time on a resource. Thus, each reservation reserves the whole resource for a certain period of time to execute a task which makes use of a service instance provided by the resource.
- The planner has to communicate with resource owners to produce a plan without reservation conflicts. 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 all resources is not considered, since individual resource owners may not want their workload, which may be commercially sensitive, to be exposed. 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 $et_{i,p}$. For instance, let $\mathcal{L}_1 = \{(0, 6), (8, 12), (30, 50)\}$ and for task 0, $dat_{0,1} = 0$ and $et_{0,1} = 3$, then it holds that $f_Q(0, 1, 0, 3) = (12, 15)$.

3.2 Planning Model

As indicated above, three different entities are considered in our model of the BDC-planning problem: *user*, *planner*, and *local resource manager*(LRM).

A **local resource manager (LRM)** owns resources and provides particular services available from its resources. The information related to these services is registered in a service repository to be retrieved by the planner or published to users. Moreover, the local resource manager responds to enquiries from the planner about the availability of a requested time slot on its resources, information which is needed to make planning decisions.

A **user** is the consumer of the provided services. To run an application on the resources, the user needs to submit first a request specifying the workflow he/she wants to run, as well as the budget and deadline constraints.

A **planner** comes up with a plan of how the submitted workflow can run on the resources owned by LRMs, taking into account their existing reservations. The planner does not own any resource, so it is not supposed to directly schedule the workflow tasks to the resources. Instead, it plans based on the retrieved information about resources from LRMs.

Figure 1 shows how the user, the planner and the LRMs interact during a BDC-planning

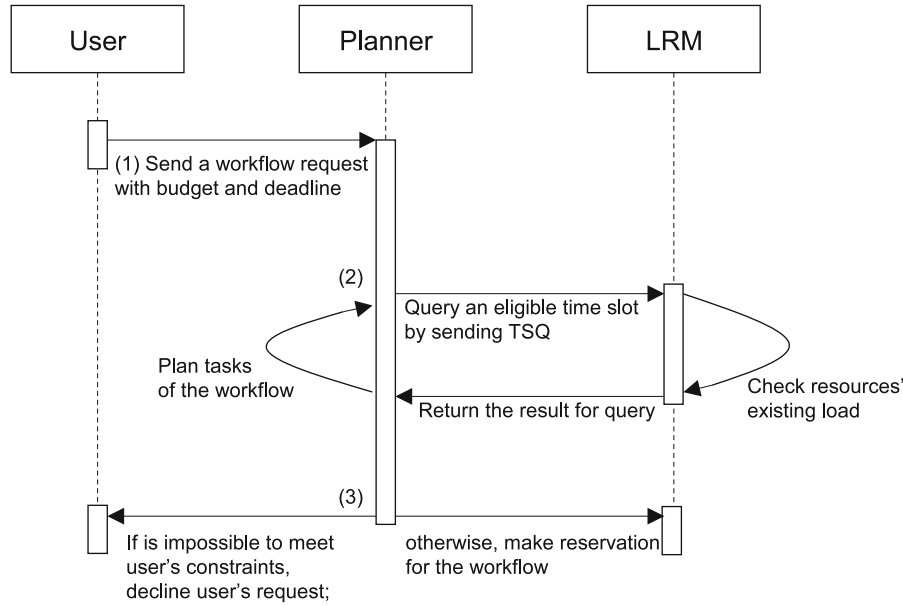


Fig. 1 Sequence diagram for BDC-planning

procedure. The protocol can be summarized as follows:

1. A user submits a workflow request with budget and deadline to the planner.
2. Upon receiving the workflow and associated constraints, the planner begins planning in order to find possible allocations for all tasks of the workflow. To do this, the planner needs to find the earliest finish time of a task on a particular resource (assuming an earliest start time). To get this information, the planner needs to send an enquiry (TSQ) to the LRM which controls the resource.
3. Once planning completes, the planner checks whether the constraints can be satisfied and responds to the user. If any of the constraints is not met, the workflow request will be rejected; otherwise, the workflow tasks will be reserved according to the planning result and the user will be notified with acceptance.

It is also assumed that the user will accept the reservation as long as constraints are met. It is easy to modify the protocol to cover the case that even though user constraints are met, the user may still not proceed with a reservation.

4 Solution of the BDC-Planning Problem

4.1 The Proposed Heuristic

The proposed heuristic, BHEFT, is an extension of the well-known DAG scheduling heuristic HEFT by taking a budget constraint into account when planning tasks. Similar to the original HEFT algorithm, BHEFT also has two major phases: *task prioritizing* and *service selection*. BHEFT is shown in Fig. 2.

In the task prioritizing phase, the priorities of all tasks are computed using upward ranking which is the same as defined in the original version of HEFT [32]. The rank of a task i is recursively defined by

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

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 assess-

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.(5)) 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 3.1.
10. **endfor**
11. Select a service for task k according to the defined selection rules.
12. **endfor**

Fig. 2 The BHEFT heuristic

ment of the spare budget which varies as planning proceeds. For this assessment, three variables are used: *Spare Application Budget* (SAB), *Current Task Budget* (CTB), and *Adjustment Factor* (AF). Suppose that the kth task is being allocated, SAB_k and CTB_k are respectively computed by

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

$$CTB_k = \overline{c}_k + SAB_k \times AF_k \tag{3}$$

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. It is not difficult to see that SAB_k is a value intended to depict the expected spared budget when planning task t_k, CTB_k is a value intended to quantify the budget allocated to t_k, and AF_k is a value intended to act as a weight that tunes the impact of SAB_k on CTB_k. We note that different values for AF_k may lead to different variants of BHEFT with different results. Apparently, given certain values of $\overline{c}_k$ and SAB_k, CTB_k grows as AF_k increases. For a task k, the larger CTB_k is, the more likely it is that k will be allocated to a more expensive but more powerful resource. This is because AF_k is used to adjust the amount of spare budget for the whole workflow (SAB_k) given to the current task. A reasonable approach is to make AF_k equal to the ratio between the average cost of the current task to the sum of the average costs of the remaining tasks as follows:

$$AF_k = \begin{cases} \overline{c}_k / \sum_{i=k}^{n-1} \overline{c}_i & : \quad SAB_k \geq 0 \\ 0 & : \quad SAB_k < 0 \end{cases} \tag{4}$$

Based on the allocated budget to task t_k, a set $\mathcal{S}_k^*$ is constructed consisting of an *affordable service* for task k, i.e.,

$$\mathcal{S}_k^* = \{s_{x,p} | \exists s_{x,p}, c_{k,p} \leq CTB_k\} \tag{5}$$

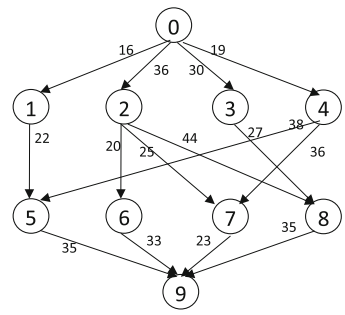
Then the "best possible" service is selected by the selection rules as follows:

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;

The algorithm terminates when all tasks, as ranked in the task prioritizing phase, are considered.

4.2 An Example

An example workflow with 10 tasks is used here to illustrate the BHEFT heuristic. The example is shown in Fig. 3. More specifically: Fig. 3a shows the structure of the DAG and the amount of data transferred as a result of each dependence;



(a) An example DAG

Task	R_0	R_1	R_2	Task	R_0	R_1	R_2
0	26	12	45	5	47	37	33
1	46	47	31	6	48	11	25
2	39	11	31	7	49	36	45
3	16	37	12	8	22	36	10
4	14	35	23	9	29	47	27

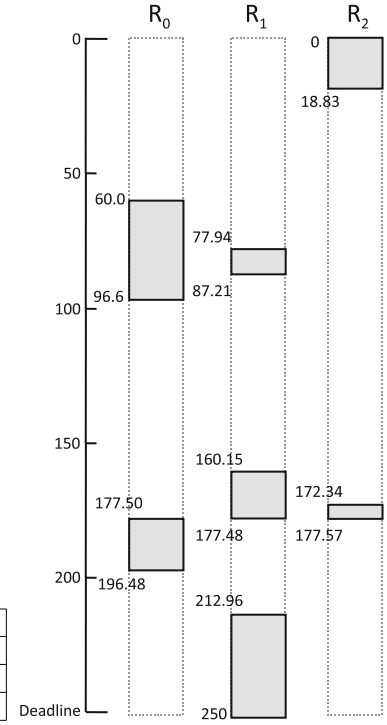
(b) Estimated execution times of tasks on resources

Resources	Time to transmit a data unit
R_0-R_1	0.52
R_1-R_2	0.68
R_0-R_2	1.40

(c) Time to transmit a data unit between different resources

Resource	Price
R_0	0.92
R_1	0.29
R_2	0.40

(d) Price for running a task on different resources



(e) Existing load on different resources

Fig. 3 An example of BDC-planning

Fig. 3b gives the estimated execution time of each task on three different resources; Fig. 3c gives the data transmission cost between different resources; Fig. 3d gives the price for running tasks on each resource; and Fig. 3e depicts the existing load (that is, existing reservations, which are denoted by the shaded part and annotated with specific start and finish times next to each part) of each resource.

We note that the arrows in Fig. 3a, denoting a dependence, may cause a delay to transfer data between two dependent tasks if the tasks are executed on different resources. This delay is computed by the product of the amount of data transferred and the transmission cost. For example, task 8 needs to transmit 35 units of data to task 9 and the transmission cost between resource 0 and resource 2 is 1.40. This means that if task 8 is executed on resource 0 and task 9 is executed on resource 2, there will be a delay equal to $35 \times 1.40 = 49$ until all the data generated by task 8 and needed by task 9 is transferred across resources. We assume that this delay is zero if two tasks, connected by a dependence, are executed on the same resource.

Assume a deadline of 250 and a budget of 150. Then, the steps taken by BHEFT to find a possible allocation for each task (and, hence, a BDC-plan) can be summarized as shown in Table 1; workflow tasks are sorted in the order that they get planned (as ranked). The values computed for each planned task clearly suggest how BHEFT guides the planning to be within budget and deadline constraints. For instance, when the first task is planned, the expected spare budget for the whole application (represented by SAB_0) is less than zero. Then there is only one affordable service for task 0, which is provided by the cheapest resource. When task 8 is planned, there is an abundance of spare budget, so all three services are affordable. In this case, for task 8, the service with the minimum execution time can be chosen.

Figure 4 shows the outcome of BDC-planning using BHEFT on the example DAG. In order to satisfy the budget constraint in this example, BHEFT allocates most of the workflow tasks to

Table 1 An example to illustrate the steps of BHEFT using the workflow in Fig. 3

$Task_k$	$Rank_k$	SAB_k	CTB_k	AF_k	S_k^*	Selected	Start	Finish	Cost
0	184.51	−20.51	15.13	0.00	$s_{0,1}$	$s_{0,1}$	0.0	12.0	3.48
1	147.6	−8.86	22.78	0.00	$s_{1,1}, s_{1,2}$	$s_{1,2}$	22.88	53.88	12.4
4	139.51	1.53	10.87	0.08	$s_{4,1}, s_{4,2}$	$s_{4,2}$	12.0	47.0	10.15
2	132.40	2.12	17.46	0.14	$s_{2,1}, s_{2,2}$	$s_{2,1}$	47.0	58.0	3.19
3	114.49	16.09	11.63	0.10	$s_{3,1}, s_{3,2}$	$s_{3,2}$	53.88	65.88	4.8
5	93.56	21.37	27.45	0.24	$s_{5,1}, s_{5,2}$	$s_{5,2}$	72.84	105.84	13.2
7	90.96	30.56	34.87	0.34	$s_{7,1}, s_{7,2}$	$s_{7,1}$	87.21	123.21	10.44
6	81.40	44.63	37.00	0.40	$s_{6,1}, s_{6,2}$	$s_{6,2}$	105.84	130.84	10.0
8	77.22	53.74	33.29	0.40	$s_{8,0}, s_{8,1}, s_{8,2}$	$s_{8,0}$	103.68	125.68	20.24
9	34.33	45.06	62.10	1.00	$s_{9,0}, s_{9,1}, s_{9,2}$	$s_{9,2}$	177.57	204.57	10.8

The given budget is 150, and sum = 98.70

the cheapest resource R_2, while only one task is allocated to the more expensive resource R_0. It can also be seen that the start time of task 7 and task 9 relies not only on task dependencies, but the existing load as well.

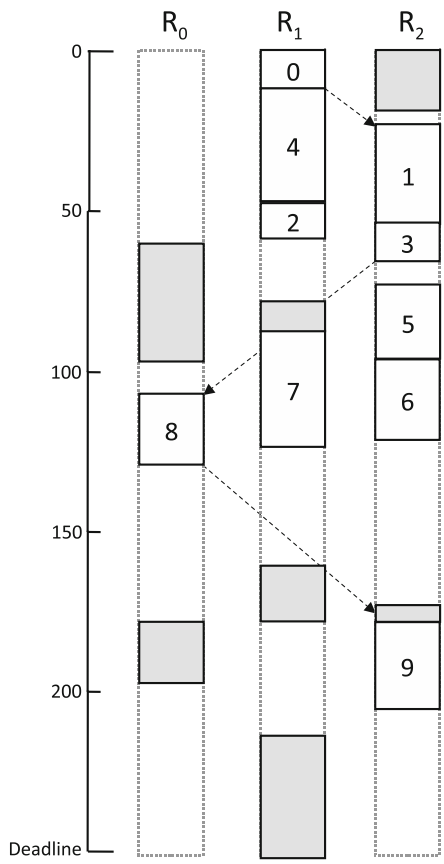


Fig. 4 The possible schedule plan generated by BHEFT using the workflow in Fig. 3

4.3 Extensions to Existing Bi-Criteria Planning Heuristics

As already mentioned in Section 2, there are several bi-criteria scheduling heuristics for workflows, which, however, were not designed specifically for the BDC-planning problem and need to be modified to produce BDC-plans. This means that they need to incorporate a mechanism for obtaining existing reservations from resources, by means of Time Slot Queries (TSQs), as specified in Section 3.1.

There are two ways for a scheduling heuristic to produce a plan that avoids reservation conflicts. One way is to produce an initial plan without considering the existing reservations and then, using TSQs, to reallocate the time slot for each mapped task in the order that tasks are initially scheduled (essentially, time slots will be shifted towards a later time, which can fully accommodate the slots onto a resource). In this case, the communication costs may be small but the overall performance of the heuristic may degrade, as a result of longer makespans. The second way is to modify algorithms to take into account existing reservations, but this requires more fine-grain changes to the algorithms. The first approach is less costly and we used it to extend DCA [19], which already has a high execution time cost, whereas we used the second approach to modify LOSS [24] and BDLS [7] as will be described next.

The basic idea of the LOSS approach [24] is as follows. The approach uses the schedule produced by any DAG scheduling heuristic (e.g., HEFT [32], HBMCT [25], etc.) as an initial

assignment. If the cost of this assignment exceeds the budget, the LOSS routine is invoked. The approach computes the LOSS weight for each task to each resource, and recursively re-assigns tasks until the budget constraint is met or all possible reassignments have been tried. Here, the LOSS weight is defined as follows:

$$LossWeight_{i,j} = \frac{T_{\text{new}} - T_{\text{old}}}{C_{\text{old}} - C_{\text{new}}} \quad (6)$$

where T_{old} and C_{old} (T_{new} and C_{new}, respectively) stand for some time property and cost property associated with the current assignment (or the assignment after re-assigning task i to resource j, respectively). For example, T and C can refer to the execution time and cost for an individual task; or they can refer to the makespan and the overall cost for the whole application. Based on our experience, using the latter leads to better performance for LOSS.

Within this setting, in order to ensure the plan produced by LOSS will not conflict with existing reservations, we make a simple extension. We take into account the existing load every time the LOSS weight is computed. That is to say, given an assignment of tasks to resources, for each task, the planner has to communicate with a local resource manager to get an idea about the earliest finish time of the task on a particular resource. This can be realized by using TSQ as introduced in Section 3.1. Using such an extension, LOSS will no longer produce a plan conflicting with existing reservations. However, as LOSS needs to compute the LOSS weight for each task on each resource, the computation and communication cost will be considerably higher with this extension.

The extension of BDLS is similar to that of LOSS. We change BDLS in such a way that every time there is a need to compute the earliest finish time of a task on a resource, TSQ is used.

5 Performance Evaluation

5.1 Experimental Setting

To run the experiments, a job planner and a set of resources were simulated by Java programs distributed on computing nodes with Intel I3 CPU with 3.1 GHz, 2 GB memory and connected through Gigabit Ethernet. The communication between the job planner 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 of a reservation slot is $\overline{RD} = UR/ATL$; for the average duration of an idle slot we can use the formula $\overline{ID} = (1 - UR)/ATL$.

The following procedure describes how the existing load of resource p ($\mathcal{L}_p$) was constructed for a given period of time, specified by the interval $[a, b]$.

1. Set $\mathcal{L}_p = \emptyset$ and current time $CT = a$.
2. Randomly determine current state among RESERVED and IDLE with equal probability.
3. If RESERVED: (a) randomly generate reserved duration RD by normal distribution with mean $\overline{RD}$ and standard deviation $\overline{RD}/2$ using only positive values for RD ($RD > 0$); (b) set $\mathcal{L}_p = \mathcal{L}_p \cup (CT, CT + RD)$; (c) set $CT = CT + RD$; (d) switch current state to IDLE.
4. If IDLE: (a) randomly generate idle duration ID by normal distribution with mean $\overline{ID}$ and standard deviation $\overline{ID}/2$ using only positive values for ID ($ID > 0$); (b) set $CT = CT + ID$; (c) switch current state to RESERVED.
5. Repeat Steps 3 and 4 until CT reaches b.

There were two service providers in the evaluation, each of which managed three resources, hence, there were 6 resources in total. Note that the model of task execution time and cost in this paper is different to the model of our earlier work presented in [40]. Instead of assuming task execution times and costs are consistent with resource power as in [40], we assume there is arbitrary heterogeneity with respect to task execution time. For each task on each resource, the estimated task execution times are randomly chosen from

the interval [10, 100]. For each resource, the price for running a task on it is randomly chosen from the interval [0.1, 1]. The period considered for modelling existing reservations was [0, 5000].

Four types of DAGs, corresponding to real-world workflow applications, were considered in the experiments. These are:

- AIRSN [15] with 53 nodes
- LIGO [6] with 77 nodes
- Montage [3] with 98 nodes
- SDSS [2] with 124 nodes

The communication computation ratio (CCR) was randomly selected from the interval [0.1, 1.0] and the data amount transmitted between tasks is randomly generated according to the CCR.

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 [32] to the DAG without considering the existing load of resources. The deadline constraint DC was considered to be located between the lower bound $LB_{dc} = M_{\text{HEFT}}$ and the upper bound $UB_{dc} = 3 \times M_{\text{HEFT}}$. 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 [19], LOSS [24] and BDLS [7] in the experiments. As mentioned in Section 3, some modification is needed to use these heuristics, which do not consider the existing load of resources, to produce a contention-free BDC-plan. According to the evaluation in [19], 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 as 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. The original names, without the suffix, are used for short in the discussion. In terms of the configuration of DCA and BDLS, the same settings as used in [19] are adopted, i.e., 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. Furthermore, LOSS3 in [24] is assumed to represent LOSS. Finally, 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:

$$PSR = 100 \times \frac{\text{number of times for which a BDC-plan was found}}{\text{number of total repeated times of experiment}} \tag{7}$$

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 to 0.5 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

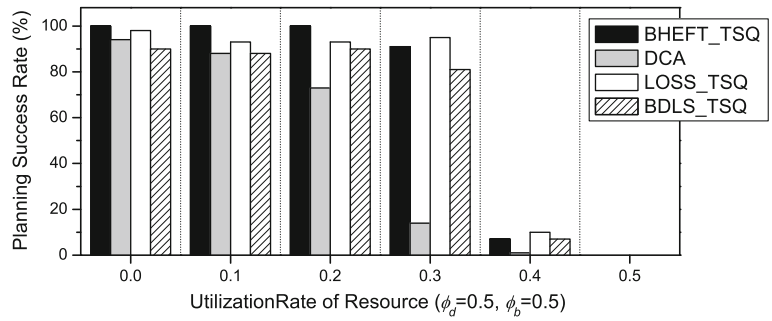
generated in the interval [0.1, 0.4], and the ATL was computed correspondingly. ϕ_d and ϕ_b were selected from the set {0.25, 0.5, 0.75} to form 9 combinations which covered a wide spectrum of diverse user requests; the experiment was then repeated 500 times for each combination. Thus, the value of PSR 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.

5.2 Experimental Results

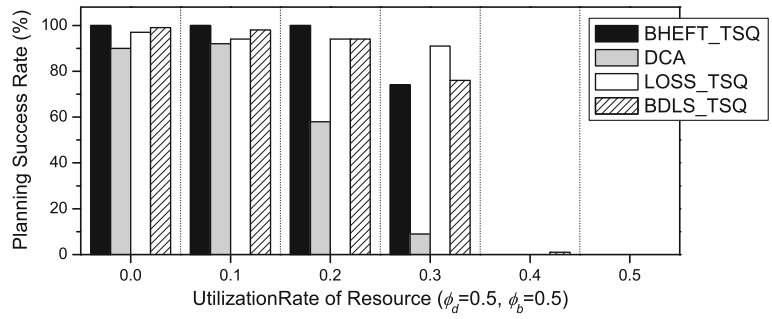
First set of experiments Figure 5 shows the results of the first set of experiments where the impact of the existing load of resources is investigated by considering six values for the Utilization Rate, UR, from 0 to 0.5. Here, ϕ_d and ϕ_b are both fixed to be 0.5 to avoid unnecessary disturbance caused by setting the user constraints to be too tight or too relaxed. It can be seen from Fig. 5 that the behaviour of the compared heuristics in terms of their PSR follows the same pattern regardless of the type of DAG. BHEFT shows the best performance in most cases where the utilization rate is lower than 0.3. When the utilization rate is equal to 0.3, LOSS outperforms BHEFT. LOSS's superiority is more profound as we move from the smallest DAG (AIRSN with 53 nodes) to the largest DAG (SDSS with 124 nodes). Only in the case where SDSS is used and utilization rate equals 0.3, BDLS performs the best and clearly better than BHEFT. As expected, all heuristics perform worse as UR increases. With a fixed setting of user constraints, it looks like the performance of LOSS and BDLS is more stable than BHEFT. In the third experiment, we will see how, with different constraints, the performance of these heuristics changes as the utilization rate grows.

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 6 shows the value of PSR for different types of DAG and different budget-deadline constraints. Again, it can be seen from Fig. 6 that the behaviour of the compared heuristics in terms of their PSR follows the same pattern regardless of the type of DAG. One interesting observation is that when both the deadline constraint and the budget constraint are tight, for example, $\phi_d = 0.25$ and $\phi_b = 0.25$, all four heuristics obtain low PSRs; among them, BHEFT achieves the best PSR which is between 40 to 60 %, and LOSS achieves the second best PSR which is between 30 to 50 %. In addition, although BHEFT performs worse than either LOSS or BDLS in some cases, it never performs clearly worse than both LOSS and BDLS at the same time. This suggests that BHEFT may be less sensitive to the tightness of budget and deadline constraints, compared to LOSS and BDLS. In fact, when the deadline constraint is tight, BDLS performs particularly bad; while when the budget constraint becomes tight, LOSS's performance degrades significantly. In contrast, BHEFT deals with the impact of the constraints in a more gracious way.

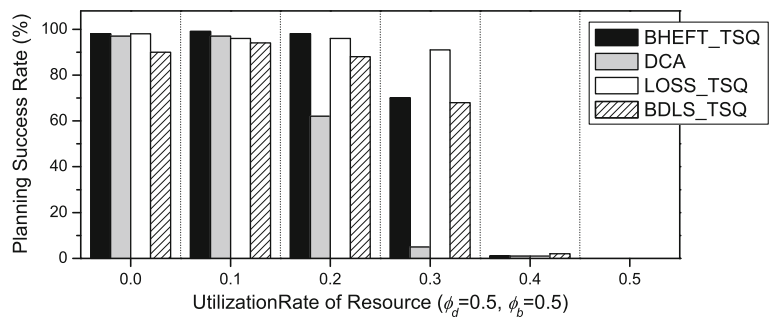
Third set of experiments In order to consider the impact of the Utilization Rate in more detail, we studied the PSR for the nine different combinations of user constraints and three different values of utilization rate. The results, for two types of DAG, Montage and LIGO, are shown in Figs. 7 and 8. Once again, BHEFT performed the best among the competitive heuristics when both deadline and budget constraints are tight. The performance of DCA, which is the only heuristic that does not consider the existing load during planning, degrades dramatically as the utilization rate grows. This highlights the impact that the existing load of resources may have on BDC-planning. With different values of utilization rate, we can again observe that BHEFT is less sensitive to the variance of constraints than LOSS and BDLS. As expected, when the Utilization Rate is low, that is when there is little existing load on resources, and the constraints for budget and deadline are



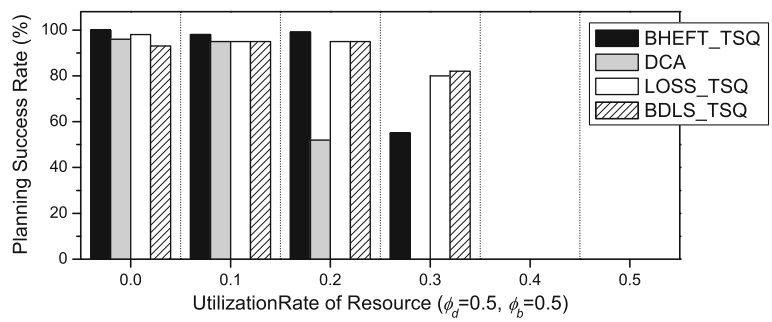
(a) AIRSN, 53 nodes



(b) LIGO, 77 nodes

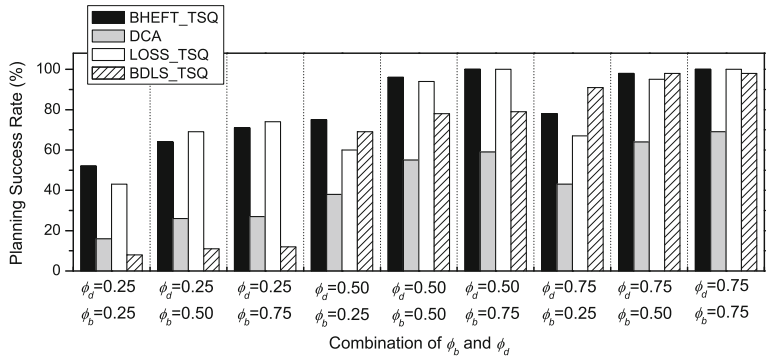


(c) Montage, 98 nodes

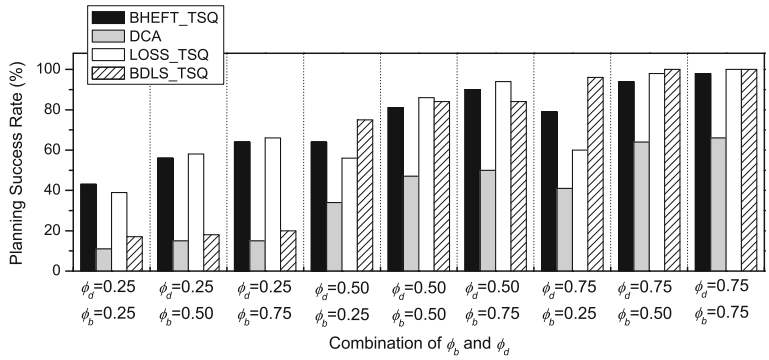


(d) SDSS, 124 nodes

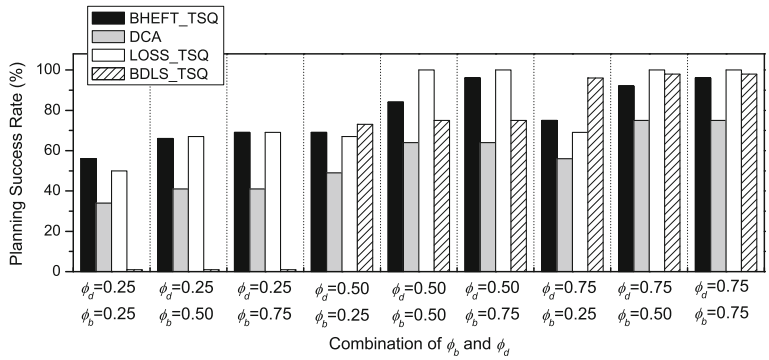
Fig. 5 First set of experiments: *PSR* with different utilization rate of resources



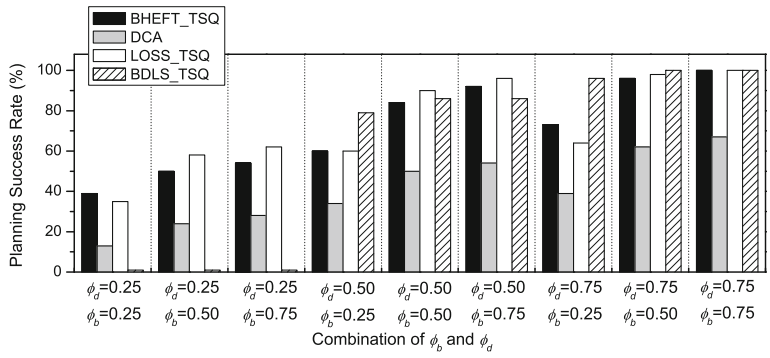
(a) AIRSN, 53 nodes



(b) LIGO, 77 nodes

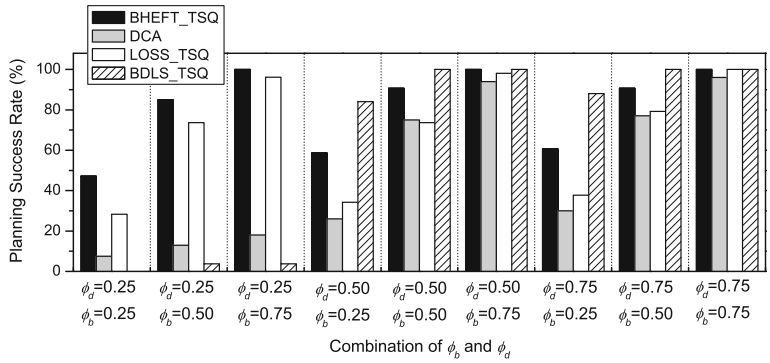


(c) Montage, 98 nodes

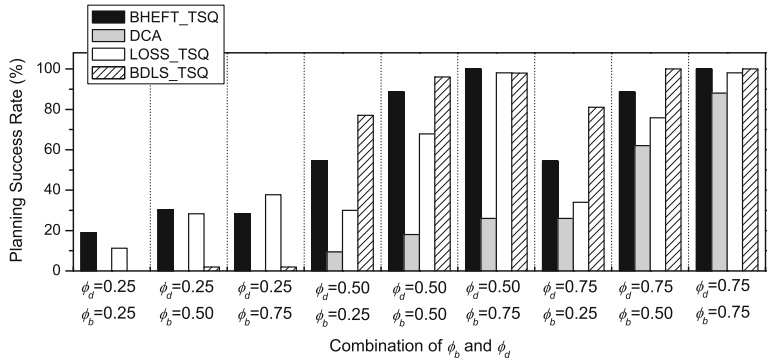


(d) SDSS, 124 nodes

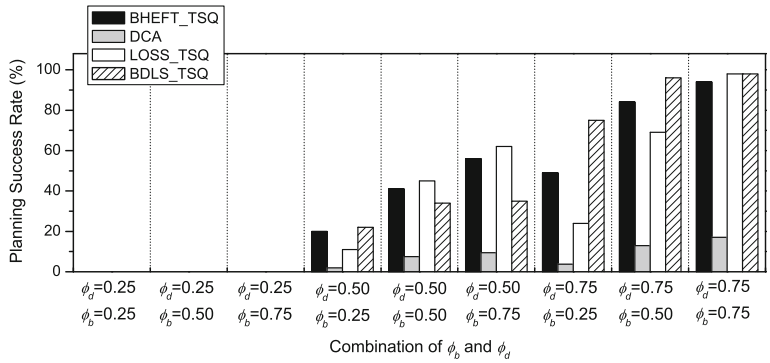
Fig. 6 Second set of experiments: *PSR* with different types of constraints



(a) Montage, Utilization Rate = 0.2



(b) Montage, Utilization Rate = 0.3

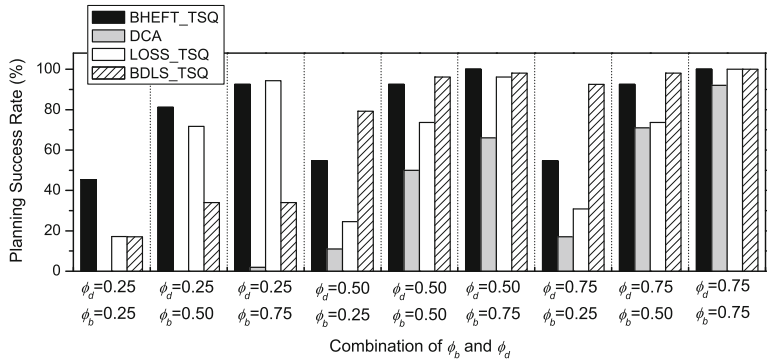


(c) Montage, Utilization Rate = 0.4

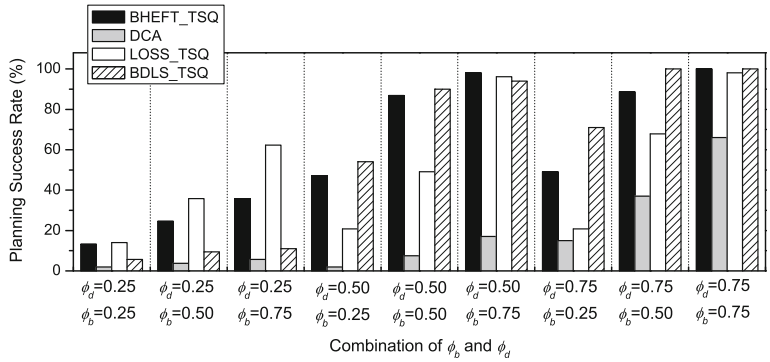
Fig. 7 Third set of experiments: *PSR* with different utilization rates and constraints for Montage

relaxed (e.g., $\phi_d = 0.75$ and $\phi_b = 0.75$), all heuristics perform equally well.

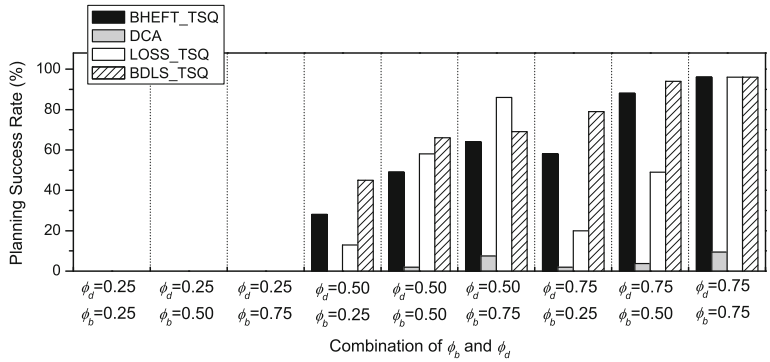
In the first, second, and third set of expriments, we compared the four competing heuristics in a total of 114 different cases. We also counted how each of the four heuristics ranked in comparison to the others. To do this, first we excluded those cases where none of the heuristics obtained a *PSR* over 10 %. Such cases exist when: (i) the Utilization Rate is 0.4 or 0.5 in Fig. 5; and (ii) $\phi_d = 0.25$ in Figs. 7c and 8c. Then, a total of 100 cases remained. In these 100 cases, for each heuristic, we count how many times the heuristic ranks first, second, third or fourth. The relevant count is denoted by R_1, R_2, R_3 and R_4, respectively, in Table 2. Note that two or more heuristics can share the same rank if there is a tie. Finally, an average ranked value, *AR*, is obtained by comput-



(a) LIGO, Utilization Rate = 0.2



(b) LIGO, Utilization Rate = 0.3



(c) LIGO, Utilization Rate = 0.4

Fig. 8 Third set of experiments: *PSR* with different utilization rates and constraints for LIGO

ing $AR = (R_1 + 2R_2 + 3R_3 + 4R_4)/100.0$ for each heuristic. As shown in Table 2, the results clearly indicate that BHEFT outperforms other competitors on average.

Fourth set of experiments In the fourth experiment, the execution time needed by each algorithm to obtain a planning result was studied. Figure 9 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 may not be scalable to

Table 2 Ranking count results for the first, second and third set of experiments

Heuristic	R_1	R_2	R_3	R_4	AR
BHEFT	39	51	10	0	1.71
DCA	0	2	26	72	3.70
LOSS	36	27	36	1	2.02
BDLS	44	14	23	19	2.17

large applications and too time-consuming for online 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, BDLS and BHEFT, are both based on list scheduling, but BHEFT needs evidently less running time than BDLS due to simpler computation and the fact that less communication is needed when making scheduling decisions. Furthermore, BHEFT is the most scalable in terms of the growth of DAG size (and potentially the number of resources which is considered constant in this experiment). As can be seen in the graph, when planning SDSS with 124 nodes on 6 resources, BHEFT only needs around 0.08 s on average. This suggests that BHEFT copes well with the real-time requirements of workflow planning.

5.3 Summary of Observations

The experimental results lead to the following observations:

- The existing load of resources may have significant impact on BDC-planning. Directly applying a heuristic that does not consier 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 sometimes perform slightly better than BHEFT, but at a higher execution time cost and, thus, they are naturally non-scalable as the size of DAG increases. Such heuristics are not suitable for BDC-planning with real-time requirements.
- Compared to the results in our previous work [40], where task execution times and costs were assumed to be consistent with the resource power, the performance of BHEFT and DCA is similar in the evaluation of this paper too, where task execution times and costs are assumed to be arbitrary. The notable

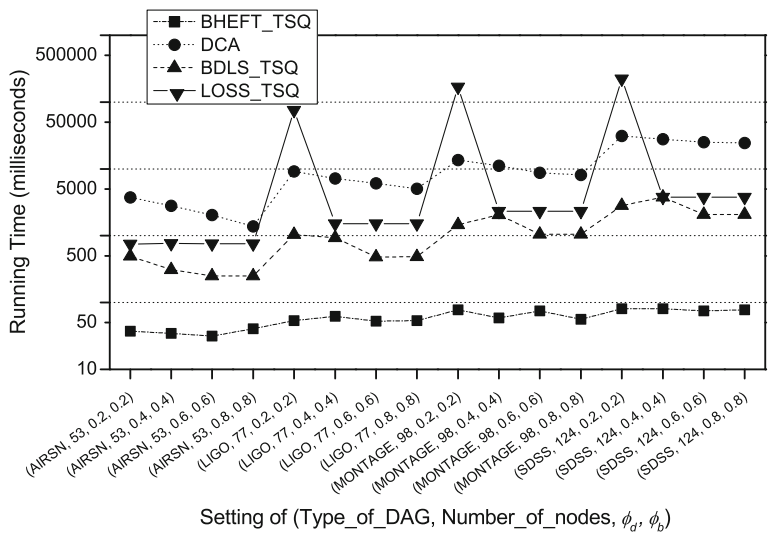


Fig. 9 Fourth set of experiments: execution time for each heuristic with different DAGs and user constraints

exception is LOSS, which performs much better when task execution times and costs are assumed to be arbitrary. This is possibly because of its use of the weight (6), which can easily take into account differences of cost and execution. This implies that LOSS may be better used in computing environments where there is inconsistent heterogeneity even though further investigation is required to support this.
– 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 satisfying the requirements of BDC-planning.

6 Conclusion and Future Work

BDC-planning is required before an SLA is established in order to guarantee that a service provider can meet the SLA without risking its failure. This paper proposed BHEFT, a novel low-cost bi-criteria heuristic based on the well-known DAG scheduling heuristic 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, from tight to more relaxed. Thus, the use of BHEFT can enable a quick admission control decision (i.e., a judgement of whether or not a submitted user request is acceptable), and make it possible to automate the creation of an SLA (from the provider's point of view) over diverse user constraints.

Based on the work in this paper, further work could try to examine the performance of BHEFT using different DAGs, settings of resources, and possibly a different core DAG scheduling heuristic (that is, not HEFT). Further experiments could also investigate how BHEFT can cope with significant overestimations or underestimations of task execution time and assess its robustness against such uncertainties.

References

1. Almeida, J., Almeida, V., Ardagna, D., Cunha, I., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. J. Parallel Distrib. Comput. **70**(4), 344–362 (2010)
2. Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S., Foster, I.: Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In: Proceedings of the 2002 ACM/IEEE Conference on SuperComputing, pp. 56–69 (2002)
3. 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), pp. 593–596 (2004)
4. 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)
5. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. **25**, 528–540 (2009)
6. 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 02), pp. 225–234 (2002)
7. Dŏgan, A., Özgüner, R.: Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. Comput. J. **48**(3), 300–314 (2005)
8. Fard, H.M., Prodan, R., Barrionuevo, J.J.D., Fahringer, T.: A multi-objective approach for workflow scheduling in heterogeneous environments. In: CCGRID, pp. 300–309. IEEE (2012)
9. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility Grids: time and cost trade-off management. In: 32nd Australasian Computer Science Conference (ACSC 2009), vol. 91, pp. 139–147 (2009)
10. Garg, S.K., Buyya, R., Siegel, H.J.: Time and cost trade-off management for scheduling parallel applications on utility Grids. Future Gener. Comput. Syst. **26**(8), 1344–1355 (2010)
11. 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), pp. 493–517 (2008)

12. Gkoutioudi, K., Karatza, H.D.: Multi-criteria job scheduling in Grid using an accelerated genetic algorithm. J. Grid Comput. **10**(2), 311–323 (2012)
13. Han, Y., Youn, C.: A new Grid resource management mechanism with resource-aware policy administrator for SLA-constrained applications. Future Gener. Comput. Syst. **25**(7), 768–778 (2009)
14. Hiles, A.: Service Level Agreements: Measuring Cost and Quality in Service Relationships. Chapman & Hall (1993)
15. Horn, J.V., Dobson, J., Woodward, J., Wilde, M., Zhao, Y., Voeckler, J., Foster, I.: Grid-based computing and the future of neuroscience computation. In: Methods in Mind, pp. 141–170. MIT Press (2006)
16. Juve, G., Deelman, E., Berriman, G.B., Berman, B.P., Maechling, P.: An evaluation of the cost and performance of scientific workflows on amazon ec2. J. Grid Comput. **10**(1), 5–21 (2012)
17. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In: Hollingsworth, J.K. (ed.) SC, p. 22. IEEE/ACM (2012)
18. Mills, K.L., Dabrowski, C.: Can economics-based resource allocation prove effective in a computation marketplace? J. Grid Comput. **6**(3), 291–311 (2008)
19. Prodan, R., Wieczorek, M.: Bi-criteria scheduling of scientific Grid workflows. IEEE Trans. Autom. Sci. Eng. **7**, 364–376 (2010)
20. Prodan, R., Wieczorek, M.: Negotiation-based scheduling of scientific Grid workflows through advance reservations. J. Grid Comput. **8**(4), 493–510 (2010)
21. Quan, D.M.: Mapping heavy communication workflows onto Grid resource within SLA context. In: Proceedings of the International Conference of High Performance Computing and Communication (HPCC06), pp. 727–736 (2006)
22. Quan, D.M., Kao, O.: Mapping Grid job flows to Grid resources within SLA context. In: Proceedings of the European Grid Conference (EGC2005), pp. 1107–1116 (2005)
23. 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, Lecture Notes in Computer Science, vol. 5745, pp. 46–59. Springer (2009)
24. Sakellariou, R., Zhao, H.: A hybrid heuristic for DAG scheduling on heterogeneous systems. In: Proceedings of the 13th Heterogeneous Computing Workshop (2004)
25. 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 (2007)
26. Schneider, J., Linnert, B.: Efficiently managing advance reservations using lists of free blocks. In: SBAC-PAD, pp. 183–190. IEEE Computer Society (2011)
27. 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 (SC2006), pp. 103–118 (2006)
28. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst. **4**(2), 175–187 (1993)
29. 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)
30. Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., Wang, J.: Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput. (2013). doi:10.1016/j.parco.2013.03.002; url: http://www.sciencedirect.com/science/article/pii/S0167819113000355
31. Talukder, A.K.M., Kirley, M., Buyya, R.: Multiobjective differential evolution for scheduling workflow applications on global Grids. Concurr. Comput. Pract. Exp. **21**(13), 1742–1756 (2009)
32. 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)
33. Wieczorek, M., Hoheisel, A., Prodan, R.: Taxonomies of the multi-criteria Grid workflow scheduling problem. In: Proceedings of the CoreGRID Workshop on Grid Middleware, pp. 237–264 (2007)
34. Wieczorek, M., Siddiqui, M., Villazón, A., Prodan, R., Fahringer, T.: Applying advance reservation to increase predictability of workflow execution on the Grid. In: e-Science, p. 82. IEEE Computer Society (2006)
35. 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 (ICPP2006), pp. 451–458 (2006)
36. 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 (SCC08), pp. 49–56 (2008)
37. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program. **14**, 217–230 (2006)
38. Yu, J., Buyya, R.: Multi-objective planning for workflow execution on Grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pp. 10–17 (2007)
39. Zhao, H., Sakellariou, R.: Advance reservation policies for workflows. In: 12th Workshop on Job Scheduling Strategies for Parallel Processing (2006)
40. Zheng, W., Sakellariou, R.: Budget-deadline constrained workflow planning for admission control in market-oriented environments. In: Vanmechelen, K., Altmann, J., Rana, O.F. (eds.) GECON, Lecture Notes in Computer Science, vol. 7150, pp. 105–119. Springer (2011)