Budget-Constrained Resource Provisioning for Scientific Applications in Clouds

Hamid Mohammadi Fard, Thomas Fahringer, Radu Prodan
Institute of Computer Science, University of Innsbruck, Austria
Email: {hamid,tf,radu}@dps.uibk.ac.at

***Abstract*—Public commercial clouds emerged as new and attractive resource provisioning option for scientific computing. This new alternative raises new challenges for users of such clouds, since optimizing the completion time of scientific applications might substantially increase the monetary cost of leasing cloud resources. In this paper, we first propose a set of basic rescheduling operations covering a broad set of scenarios for reducing the costs of running scientific workflows in clouds. Based on them, we design two heuristic scheduling algorithms. The first algorithm aims at reducing the cost of resource provisioning while still attaining the optimal makespan. The second algorithm further reduces the costs to meet a budget constraint with a small increase in the makespan. The experiments conducted using real-world and synthetic workflow applications demonstrate important benefits compared to related state-of-the-art approaches.**

***Index Terms*—cloud computing, resource provisioning, scientific workflows, scheduling, makespan, monetary cost.**

I. Introduction

Several technological advantages of clouds, such as elasticity, scalability, accessibility and reliability, have made them an attractive alternative to replace private in-house IT infrastructures for scientific computing [1]. By running virtual machine images on leased resources, users simply create new instances, which from their perspective, can be considered as dedicated physical hosts. In traditional parallel and distributed systems, scientists usually attempt to use the maximum number of available machines to improve their computational throughput. In commercial clouds, users have theoretical access to an infinite amount of resources, but careless in leasing instances increases the costs or decreases the performance substantially. The wide variety of instance types and prices causes difficulties for scientists to compare cloud providers and services. Thus, resource provisioning techniques play a critical role in executing scientific applications in clouds.

Generally, resource provisioning and scheduling in cloud can be viewed from two different perspectives hidden from each other: provider-level and user-level. At provider-level, the question is how the providers manage the virtual machines on the physical resources to satisfy the expected objectives (e.g. using consolidation to optimize utilization). At user-level, the question is how users lease different instances to meet their objectives (e.g. optimizing execution time). Our focus in this paper is on user-level resource provisioning for scientific applications, also known as scheduling, which is a highly challenging optimization problem in clouds involving multiple objectives such as execution time and monetary cost. Two situations may occur in user-level provisioning: overprovisioning and underprovisioning. Overprovisioning implies unused resources and consequently wastes money, while underprovisioning commonly increases the execution time. In other words, in overprovisioning there is the risk of underutilization and in under-provisioning there is the risk of saturation.

Commercial cloud providers today typically charge the users by an hourly-based pricing model. In this model, the cost is not calculated based on the real amount of resources used, but based on a time unit model (hourly-based), meaning that the users have to pay for the whole leased hour even if they use the instance for one second. The challenge becomes how to lease resources to execute an application within a proper time and with an affordable cost. In this paper, we first propose a set of basic rescheduling operations covering a broad set of scenarios for reducing the costs of running scientific workflows in clouds. Based on them, we design two heuristic scheduling algorithms. The first algorithm aims at reducing the cost of resource provisioning while still attaining an optimal makespan. The second algorithm further reduces the costs to meet a budget constraint with a small increase in the makespan. Our algorithm overcomes the drawbacks of existing budget-constrained scheduling methods proposed for utility grids (e.g. [2]) employing an exact use pricing model (i.e. users pay for the exact use of resources) that are not appropriate for cloud environments. Furthermore, existing budget-constrained scheduling methods for hybrid clouds (e.g. [3]) are not applicable in our scenario, as they use cloud instances only to extend the original internal resource pool.

This paper is organized as follows. We review the related work in Section II. In Section III, we formally introduce the problem addressed. Section IV presents the basic rescheduling operations, followed by an algorithm to reduce the workflow execution cost with an optimal makespan in Section V. Section VI proposes an algorithm for executing a workflow under a budget constraint. We evaluate our work in Section VII and conclude the paper in Section VIII.

II. Related Work

Workflow scheduling in distributed systems is a well-studied research area. Inside of this wide area, we focus here on the researches dealing with execution time and cost.

LOSS and GAIN [2] are two algorithms for scheduling workflows in utility grids based on a set of static rescheduling steps in two phases: the first phase optimizes one objective

978-0-7695-5095-4/13 $31.00 © 2013 IEEE
DOI 10.1109/CloudCom.2013.48

(time or cost) and in the second phase the specified budget constraint is applied. A bi-criteria genetic optimization algorithm has been proposed in [4] for utility grids. Considering budget constraint, the fitness function is defined as a combination of the partial fitness functions of the two objectives. The workflow scheduling method proposed in [5] for utility grids minimizes the execution cost of a workflow while meeting the deadline. In the first step, it divides the overall workflow deadline into sub-deadlines for all tasks. In the second step, it models the sequential tasks as a Markov decision process solved using a value iteration method. The authors in [6] customize well-known multi-objective optimization evolutionary algorithms for the workflow scheduling problem which approximate of the Pareto set. All these approaches employ an exact use pricing model not applicable to today's clouds that use time unit-based pricing. In Section VII we will observe that the scheduling methods proposed for the exact use pricing model are not efficient in cloud infrastructures.

Other works targeted the monetary cost of workflow executions in hybrid clouds (surge computing), which are a combination of in-house IT infrastructures and public clouds. The authors in [7] solve the problem of resource provision for preemptible tasks in a hybrid cloud using a linear programming technique. The goal is maximizing the utilization of the in-house data center and minimizing the cost of using the public cloud while fulfilling a deadline constraint. HOCC [3] attempts to optimize the cost of running a workflow in a hybrid cloud by considering a deadline by assuming that VMs are assigned to dedicate services and are not able to run other services. In the other words, each instance is able to execute only a single kind of task. The research in [8] proposes two time and cost optimization techniques for resource provisioning in hybrid clouds that meet a deadline within a budget constraint. The tasks are independent (i.e. no workflows targeted) and their load are not known in advance but are approximated at runtime.

The authors of [9] argue that static scheduling approaches in clouds operate more efficiently than other distributed systems such as grids. Cloud providers guarantee the SLAs and the instances are dedicated to users (i.e. no dynamic load on the VMs). In the other words, users have a full access-right on their instances and are aware of the current loads of the instances. In contrast, grid resources are shared between users and usually there is no guarantee about the performance of resources. Moreover, the authors discuss that task scheduling in clouds is even more complex than in grids since the number of resource in clouds can be theoretically infinite.

In [10] a bi-objective scheduling algorithm for BPEL workflows based on SPEA2 [11] is presented. The assumption is that tasks of the workflow are geographically distributed on some data centers. BaTS [12] is a budget-constrained scheduling approach for a set of independent tasks modeled as a bag of tasks in multi-cloud environment. The completion time of tasks is not known in advance but the algorithm learns to how estimate them at runtime. The algorithm assumes that the tasks can be preempted and rescheduled later. The approach proposed in [13] targets the resource provisioning for a workflow meeting a deadline and minimizing the cost in a commercial cloud. Similar to [5], the idea is to divide the deadline of workflow to sub-deadlines for tasks and then select or start the instances which are able to meet the determined sub-deadlines. Several static and dynamic scheduling and cloud resource provisioning algorithms for a set of workflows with different priorities have been proposed in [14]. The goal is to maximize the number of workflows that are completed inside of the specified deadline and budget constraints.

To the best of our knowledge, our work is the first to propose and implement a resource provisioning method for workflow applications (dependent tasks) in a public commercial cloud that optimizes the makespan while meeting budget constraints.

III. Model

We model a *scientific workflow* as a directed acyclic graph $W = (A, D)$ consisting a set of n tasks: $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through a set of control flow dependencies: $D = \{(A_i, A_j) \,|\, (A_i, A_j) \in A \times A\}$, where (A_i, A_j) indicates that A_i needs to be executed before A_j. We assume the data transfers between the tasks A_i and A_j as part of the execution time of A_j. We use $pred\,(A_j)$ to denote the set of the immediate predecessors of task A_j:

$$A_i \in pred\,(A_j) \Leftrightarrow (A_i, A_j) \in D.$$

Each workflow has a task A_{entry} with no predecessors (i.e. $pred\,(A_{entry}) = \emptyset$) and a task A_{exit} with no successors (i.e. $\nexists\,(A_{exit}, A_j) \in D$). Finally, we assume the availability of the computational workload of each task A_i measured in million of instructions, denoted as $work\,(A_i)$.

We assume that the *cloud providers* offer a set of k virtual machine *instance types* $T = \bigcup_{i=1}^{k} \{T_i\}$ with an hourly-based pricing model, without loss of the generality. Each instance type T_i is specified by several characteristics: computational speed s_i in million instructions per second, cost per hour c_i and boot delay d_i. We denote the set of leased instances as: $I = \bigcup_{i=1}^{m} \{I_i\}$, whose number m can dynamically change since instances can be leased and terminated at any time during the execution of a workflow. Every instance I_i has an instance type $type\,(I_i)$, defined as a function $type : I \mapsto T$.

We calculate the *execution time* $t\,(A_i, I_j)$ of a task A_i on an instance I_j as follows:

$$t\,(A_i, I_j) = \begin{cases} \frac{work(A_i)}{s_j}, & I_j \in I; \\ d_j + \frac{work(A_i)}{s_j}, & I_j \notin I. \end{cases}$$

To calculate the execution time of a task A_i on an instance I_j, we need to consider two cases: when I_j has already been leased by the user ($I_j \in I$) and when I_j is new ($I_j \notin I$) in which case we need to consider the booting delay too. The completion time of a task A_i considering its predecessors is:

$$end\,(A_i) = \begin{cases} t\,(A_i, I_j)\,, & A_i = A_{entry}; \\ \max\limits_{A_p \in pred(A_i)} \{end(A_p) + t(A_i, I_j)\}, & A_i \neq A_{entry}. \end{cases}$$

We calculate the *cost* $c(A_i, I_j)$ of executing a task A_i on an instance I_j as follows:

$$c(A_i, I_j) = \begin{cases} 0, & I_j \in I \wedge t(A_i, I_j) \leq u_j; \\ \left\lceil \frac{t(A_i, I_j) - u_j}{h} \right\rceil \cdot c_j, & I_j \in I \wedge t(A_i, I_j) > u_j; \\ \left\lceil \frac{t(A_i, I_j)}{h} \right\rceil \cdot c_j, & I_j \notin I. \end{cases}$$

where u_j is the unused tine of instance I_j and h is the time unit (i.e the hour) charged by the cloud. This equation describes that executing a task within the unused time of an existing instance has no cost as the resource is already leased.

We finally calculate the *makespan* and the *cost* of the workflow execution as follows:

$$Makespan(W) = end(A_{exit});$$
$$Cost(W) = \sum_{A_i \in A} c(A_i, I_j).$$

We define the *schedule* of a workflow $W = (A, D)$ on a set of cloud instances I as a mapping:

$$sched : A \mapsto I, sched(A_i) = I_j.$$

Our goal is to lease the proper instances to the set I and to schedule the tasks of a workflow W so that we meet a budget constraint B while optimizing the makespan.

IV. Basic Rescheduling Operations

In this section, we introduce several basic rescheduling operations for workflow tasks and investigate their impact on the workflow makespan and cost in clouds.

Early booting (see Figure 1a) leases an instance earlier than necessary such that the instance is able to execute the task with the same cost and the same makespan.

Late booting (see Figure 1b) leases an instance later than necessary such that it is able to execute the assigned task with the same cost and the same makespan.

Gap moving (see Figure 1c) starts a task later than its ready time such that the workflow makespan remains constant. This operation can be useful if there are two parallel tasks on two separate instances. The shorter task has the possibility to start later without increasing the makespan of the entire workflow. This operation has no effect on the execution cost.

Gap filling (see Figure 1d) moves a task from one instance to the unused part of another instance without increasing the workflow makespan. Depending on the instance type, this operation might increase or decrease the execution cost.

Cheap leasing (see Figure 1e) tries to execute a task on a new cheaper instance. This operation decreases the cost but may increase the makespan. In case of two independent tasks with a common immediate successor, a cheaper leasing for one tasks may have no effect on the makespan of the workflow.

Task serialization (see Figure 1f) executes two parallel tasks (assigned to different instances) on the same instance in sequence. This operation increases the makespan but may decrease the cost.

Table I summarizes the impact of these operations on makespan and cost of workflow execution in the d.

TABLE I: Impact of the basic rescheduling operations.

Operation	*Makespan*	*Cost*
Early booting	no impact	no impact
Late booting	no impact	no impact
Gap filling	no impact	decrease, increase, or no impact
Gap moving	no impact	no impact
Cheap leasing	increase or no impact	decrease
Task serialization	increase	may decrease

Algorithm 1: Cost Efficient Fastest Makespan (CEFM).

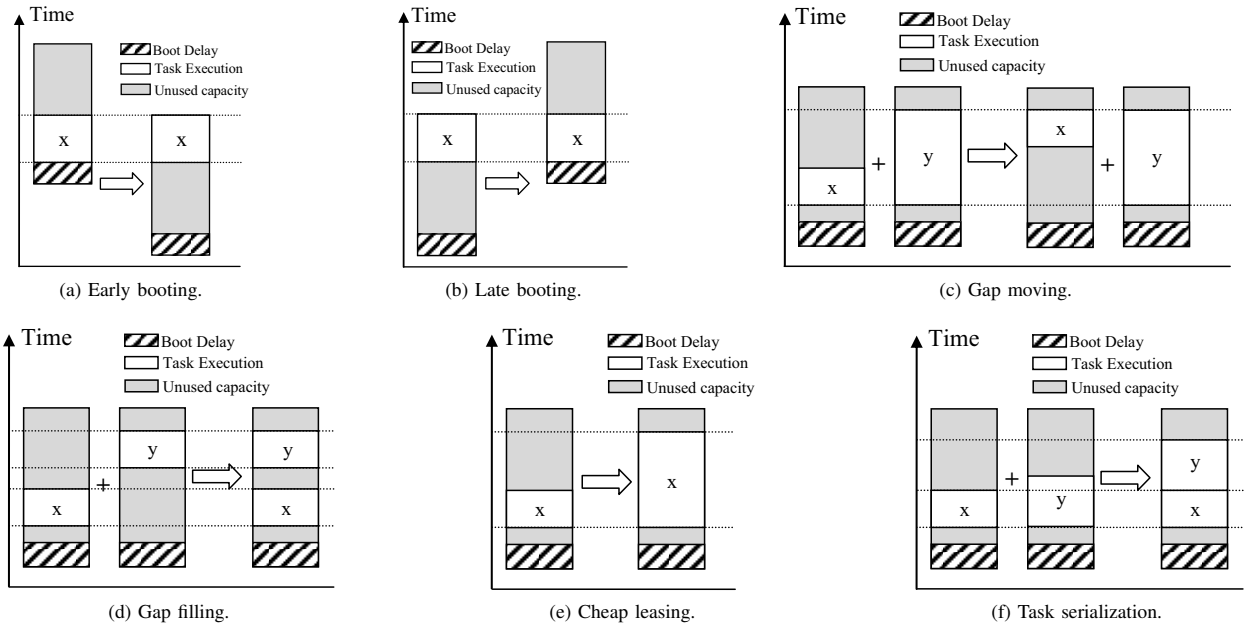
Input: Workflow application: $W = (A, D)$; Set of instance types: T
Output: Schedule of W on cloud instances: $sched_W : A \rightarrow I$

1 **begin**
  `// Compute initial schedule that considers a separate fastest instance for each task`
2 $sched_W : A \rightarrow I, sched_W(A_i) = I_j$ so that $(\nexists I_k : t(A_i, I_k) < t(A_i, I_j) \wedge type(I_k) \neq type(I_j))$
  `// 1st optimization: fill gaps of current mapping`
3 **for** $\forall I_i \in I$ **do**
4 **for** $\forall I_j \in I \wedge i \neq j$ **do**
5 **if** *gap filling is possible* **then**
6 $utilization[I_j] \leftarrow$ utilization of I_j if all tasks in I_i move to I_j
7 **end**
8 **end**
9 **if** $\exists I_j \in I : utilization[I_j] \neq \emptyset$ **then**
10 $I_{dest} \leftarrow \max_{I_j \in I} \{utilization[I_j]\}$
11 $GapFilling(I_i, I_{dest})$ `// move all tasks from` I_i `to` I_{dest}
12 $Remove(I_i, I)$ `// remove` I_i `from` I
13 **end**
14 **end**
  `// 2nd optimization: lease cheap instances for current mapping`
15 **for** $\forall I_i \in I$ **do**
16 **if** *cheap leasing is possible* **then**
17 $CheapLeasing(I_i)$ `// select cheapest instance`
18 **end**
19 **end**
20 **return** $sched_W$
21 **end**

V. Cost Efficient Fast Makespan Algorithm

In this section, we propose a new *Cost Efficient Fast Makespan (CEFM)* algorithm to optimize makespan and cost of a workflow execution in a cloud infrastructure. The CEFM algorithm (see Algorithm 1) starts with an initial mapping phase (line 2), created by selecting a separate instance of the fastest instance type for each task considering the task dependencies too. The fastest instance type for each task might be different, depending on the task's characteristics (e.g. an instance type with a large memory might operate faster for a database query). Communication latencies between instances are implicitly considered to be a part of the tasks' execution time. The initial mapping yields an optimal makespan, with the side effect in a commercial cloud of low resource utilization and high monetary cost. Based on this initial mapping, the CEFM algorithm attempts to reduce the monetary cost of the workflow execution, without increasing the makespan.

As shown in Table I, early booting, late booting and gap moving have no impact on the makespan and cost, but are able to prepare the conditions for gap filling which can reduce the cost. In order to increase the utilization and decrease costs, the first optimization loop (lines $3 - 14$) attempts to fill existing



(a) Early booting. (b) Late booting. (c) Gap moving.

(d) Gap filling. (e) Cheap leasing. (f) Task serialization.

Fig. 1: Basic rescheduling operations.

unused gaps in the leased instances without exceeding the makespan using early booting, late booting, gap moving and gap filling. We introduce a *utilization* vector in each iteration of the loop comprising m elements in the presence of m leased instances. In the utilization vector of iteration I_i, the value of cell I_j presents the utilization of instance I_j if the tasks of instance I_i move to I_j. To have a possible gap filling, three preconditions must be respected: (1) task dependencies must not be violated, (2) the workflow makespan must not be exceeded, and (3) the total unused times in the destination instance after gap filling must not be larger than the unused times of both source and destination instances before the gap filling. If these conditions are not satisfied, we assume that gap filling is not possible and the corresponding element in the utilization vector has no value.

After applying gap filling by the first optimization loop, the second loop (lines $15-19$) applies the cheap leasing operation for the existing instances by selecting the cheapest instance type. Finally, the workflow schedule is returned (line 20).

Since the CEFM algorithm targets a cost reduction without exceeding the makespan, it considers all basic operations except task serialization (see Section IV) which always increases the makespan. Task serialization is considered in Algorithm 2 on budget-constrained scheduling discussed next.

VI. Budget-Constrained Scheduling in Clouds

In this section, we introduce a *Budget-Constrained Scheduling in Clouds (BCSC)* algorithm (see Algorithm 2) to provide resources for execution of workflow applications with a nearly-optimal makespan while meeting a specified budget constraint. To reduce the monetary costs, we attempt to move workflow tasks to the unused parts of the existing instances and to lease cheaper instances. The side effect of this cost reduction is an increase in makespan. The core idea of BCSC is a continuous tradeoff between serialization and leasing cheaper instance types.

Algorithm 2: Budget-Constrained Scheduling in Cloud (BCSC).

Input: Workflow application: $W = (A, D)$; Instance types: T; Budget: B
Output: Schedule of W on cloud instances: $sched_W : A \mapsto I$

1 **begin**
2 $sched_W \leftarrow CEFM(W, T)$, where $sched_W : A \mapsto I$
3 **while** $Cost(sched_W) > B$ **do**
4 $R_S(A_i, I_j) \leftarrow$ calculate ranks for rescheduling activity A_i to instance I_j using serialization, $\forall A_i \in A, \forall I_j \in I$
5 $R_C(A_i, I_j) \leftarrow$ calculate ranks for rescheduling activity A_i to instance I_j using cheap instance, $\forall A_i \in A, \forall I_j \in I$
6 $R_m(A_m, I_m) \leftarrow \min_{\forall A_i \in A, \forall I_j \in I} \{R_S(A_i, I_j), R_C(A_i, I_j)\}$
7 **if** $R_m(A_m, I_m) = \infty$ **then**
8 **break** `// Budget not feasible`
9 **end**
10 $sched_W \leftarrow$ generate a new schedule by applying the operation that yields $R_m(A_m, I_m)$ by rescheduling A_m to I_m
11 **end**
12 **return** $sched_W$
13 **end**

Algorithm 2 starts with the solution of CEFM (see Algorithm 1) and attempts to continuously reduce the cost of workflow execution while still satisfying the specified budget constraint (lines $3-11$). In each iteration of BCSC, we attempt to reschedule each task A_i on each instance I_j different than the one already assigned by CEFM using the cheap leasing and task serialization operations. We then compute the rank of each new schedule as the ratio between the workflow makespan

increase and the cost decrease (lines $4-5$):

$$R(A_i, I_j) = \begin{cases} \frac{Makespan'(W) - Makespan(W)}{Cost(W) - Cost'(W)}, & Cost'(W) < Cost(W); \\ \infty & Cost'(W) \geq Cost(W), \end{cases}$$

where $Makespan'$ and $Cost'$ denote the makespan and cost of executing the new schedule. Since our goal is reducing the cost, we assign infinite ranks to the schedules that do not decrease it. From the set of new schedules, we select the one with the minimum rank (line $6-10$) and repeat the loop until we meet the budget constraint. However, if a cost improvement is not possible, we break the loop and report the budget as too low and unfeasible (lines $7-9$). Following this approach, the expensive instance types are replaced by cheaper types and parallel tasks serialized until we meet the budget constraint. In other words, this approach converges to a serial execution of the tasks on cheaper instances to meet the budget.

VII. Evaluation

We implemented the proposed algorithms as part of the ASKALON environment [15] supporting the development and execution of scientific workflows in grid and cloud infrastructures. Our experiments consist both real and simulation-based executions. To support flexible simulations required by a thorough evaluation, we interfaced ASKALON at the backend with the GroudSim simulator [16] supporting modeling of grid and cloud computational and network resources, job submissions, file transfers, failure models, background load, and cost models. To simulate the execution time of tasks, we used the trace data collected from historical executions conducted in the Austrian Grid (http://www.austriangrid.at).

A. Experimental Setup

We used a dual approach to evaluate our method: real workflow applications executed in the Amazon EC2 cloud and simulation of synthetic workflows that allow a more flexible investigation of features not covered by the real applications.

WIEN2k (http://www.wien2k.at) is a material science workflow for performing electronic structure calculations of solids, which contains two parallel sections with sequential synchronization tasks in between. Povray is a workflow for creating stunning three-dimensional graphics, based on the POV-Ray tools (http://www.povray.org). Povray workflow consists a set of parallel tasks to render frames in PNG format and some sequential tasks to generate the final movie and transferring it to local host. Since the tasks of the WIEN2k workflows are serial codes, we only consider the single-core instance types (`m1.small` and `m1.medium`) for this workflow. Since the tasks in Povray are parallelizable (the set of frames in each rendering task can be divided in independent sub-sets), we do not consider any limitation in the instance types running these workflows. However, because of the cost, we limited the instance types to `m1.small`, `m1.medium`, `m1.large`, `m1.xlarge` and `c1.xlarge` in the `us-east-1d` zone.

To investigate the impact of tasks' workload, workflow size and workflow shape on the results, we generated a set of synthetic workflows using the random workflow generator proposed in [17]. Four normal distributions N, N', N'' and N''' as input parameters adjust the generated workflow characteristics. The normal distribution $N\left(\mu, \sigma^2\right)$ with the mean $\mu = 0.5$ and the variance $\sigma^2 \in [0.0, 1.0]$ defines the depth (the number of levels) of the workflow. A value of zero for the variance means that all workflow activities (except the entry and exit activities) are in the same level. A value one means that there is only one activity per level (sequential activities). Another normal distribution $N'\left(\mu', \sigma'^2\right)$ with the mean $\mu' = 0.5$ and the variance $\sigma'^2 \in [0.0, 1.0]$ defines how the workflow is balanced. The value of zero for the variance means a workflow is balanced (i.e. different parallel branches have equal depth) and the value one represents an unbalanced workflow (i.e. with different depths in parallel branches). Two other normal distributions N'' and N''' with arbitrary mean and variance values generate the number of activities and the size of data transfers. To investigate the impact of the workflow size, we categorized the workflows into three classes based on their number of activities, as displayed in Figure IIa: small, medium, and large. Similarly, we classified the tasks' workloads in short, medium and long classes (see Figure IIb) and the workflow shape based on the depths of the parallel branches in balanced, semi-balanced and unbalanced (see Figure IIc). We simulated a cloud testbed using GroudSim that includes five instance types with 1.2 GHz Compute Units (CU) described in Table III (inspired from EC2). We randomly generated the boot delay in the interval of [30, 100] seconds.

TABLE II: Experimental setup.

(a) Workflow size classification

Workflow	*Small*	*Medium*	*Large*
Random	$10-500$	$10000-20000$	$20000-50000$

(b) Load classification

Load class	$work(A_i)$
Short	$100-500$
Medium	$500-3000$
Long	$3000-6000$

(c) Balance classification

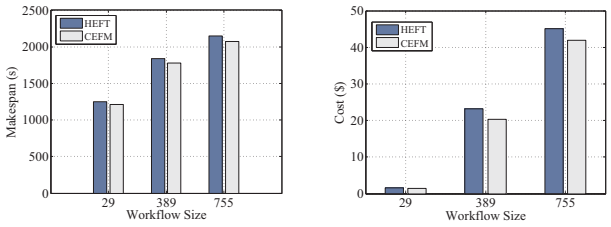
Balanced	*Semi-balanced*	*Unbalanced*
$0.0 \leq \sigma^2 < 0.2$	$0.2 \leq \sigma^2 < 0.5$	$0.5 \leq \sigma^2 \leq 1.0$
$0.0 \leq \sigma'^2 < 0.2$	$0.2 \leq \sigma'^2 < 0.5$	$0.5 \leq \sigma'^2 \leq 1.0$

TABLE III: Simulated instance types.

	`micro`	`small`	`medium`	`large`	`xlarge`
Number of CUs	0.5	1	2	4	8
Price [$ / hour]	0.02	0.08	0.20	0.32	0.64

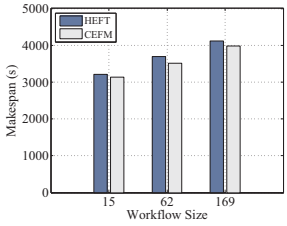
B. Experimental Results

In the first part of the experiments, we compare the time and cost of running the workflows scheduled by CEFM with the well-known Heterogeneous Earliest Finish Time (HEFT) [18] algorithm. We adapted HEFT for clouds such that, whenever there are no free instance available for the ready tasks, it requests new instances of the best type for those tasks (considering the number of ready tasks only to avoid leasing extra cores). To predict the execution time of tasks, we performed a training phase on different instance types before the workflow execution that yields an average accuracy of 97%. Figure 2 presents the cost and time of running the WIEN2k and Povray

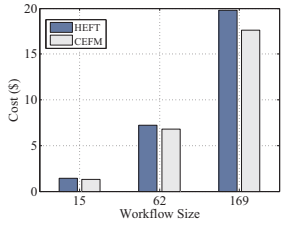


(a) Makespan of WIEN2k.

(b) Cost of WIEN2k.



(c) Makespan of Povray.



(d) Cost of Povray.

Fig. 2: Results for WIEN2k and Povray workflows.

workflows on Amazon EC2. As expected, the cost of the CEFM solution (see Figures 2b and 2d) is always lower than (or equal to) the cost of HEFT. Although the makespan of CEFM and HEFT are theoretically the same (see Section V), the makespan of CEFM in the real experiments is mostly lower than HEFT. Having more parallel instances in HEFT leads to increased communications between instances and consequently, to higher makespans.

Figure 3 presents the dispersion of costs in HEFT and CEFM for synthetic workflows. An all situations, the dispersion of costs in CEFM is lower than in HEFT. We observed that for balanced workflows, users have to spend more money to obtain shorter makespans. The reason is that balanced workflows have more parallel tasks and attaining optimal makespans requires leasing more parallel instances leading to lower utilization and consequently, to higher costs. This scenario is getting worse for shorter parallel tasks that lead to larger unused leasing times.

Figure 4 presents the average utilization in the simulated experiments. As expected, balanced workflows waste in general more resources and consequently cause less utilization. Another important factor affecting the utilization is the load of tasks. The tasks with medium loads achieve better utilization since they are able to efficiently exploit the unused instances times. Although CEFM yields a better utilization, there are still important unused instance times, exploited by the BCSC algorithm as explained in Section VI.

In the final experiments, we compared BCSC with LOSS [2] for the execution of the WIEN2k and Povray workflows. LOSS is a budget-constrained algorithm for utility grids that uses an exact usage pricing model. Since in the exact use pricing model the user only pays for the real resource usage (not hourly-based as in clouds), we developed for a fair comparison a cloud-aware version of LOSS called Extended-LOSS that uses the same pricing model as BCSC for taking scheduling decisions. Figure 5 compares the makespans of BCSC and Extended-LOSS for different budget constraints, defined based on a factor of the cost of CEFM collected from the experiments in Figure 2 (denoted as b in Figure 5).

We evaluate in Figure 6 the ability of the two algorithms of using the maximum available budget for optimizing the workflow makespan. We quantify the quality of the solution as the difference between the budget B and the cost of the solution $Cost(W)$ relative the budget: $distance = \frac{B-Cost(W)}{B}$. A negative distance means that the algorithm is not able to meet the targeted budget and exceeds it. As shown, BCSC efficiently spends its available budget to reduce the makespan and never exceeds it. In contrast, Extended-LOSS is not able to find a solution meeting the deadlines in several scenarios (see Figure 6a), while a feasible solution is still possible.

VIII. Conclusion

We propose in this paper a cost-efficient algorithm for makespan scheduling of scientific workflows in public commercial clouds based on a set of rescheduling operations. Compared to state-of-the-art, our solution provides competitive completion time and lower monetary cost. Based on this proposed solution, we introduced a budget-constrained algorithm capable of meeting budget constraints. We observed through extensive experimental evaluation on both real-world and simulation environments that the proposed algorithms are efficiently applicable in cloud infrastructures.

Acknowledgment

Austrian Science Fund project TRP 237-N23 and Standortagentur Tirol project RainCloud funded this research.

References

[1] G. Juve and E. Deelman, Scientific workflows in the Cloud, in Grids, Clouds and Virtualization, Springer, pp. 71-91, 2010.

[2] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos, Scheduling workflows with budget constraints, in Integrated Research in GRID Computing, ser. CoreGRID, Springer Verlag, pp. 189-202, 2007.

[3] L. F. Bittencourt and E. R. M. Madeira, HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds, Journal of Internet Services and Applications, vol. 2, no. 3, pp. 207-227, 2011.

[4] J. Yu and R. Buyya, A Budget constrained scheduling of workflow applications on utility Grids using genetic algorithms, Workshop on Workflows in Support of Large-Scale Science, HPDC 2006, IEEE CS Press, June 19-23, 2006.

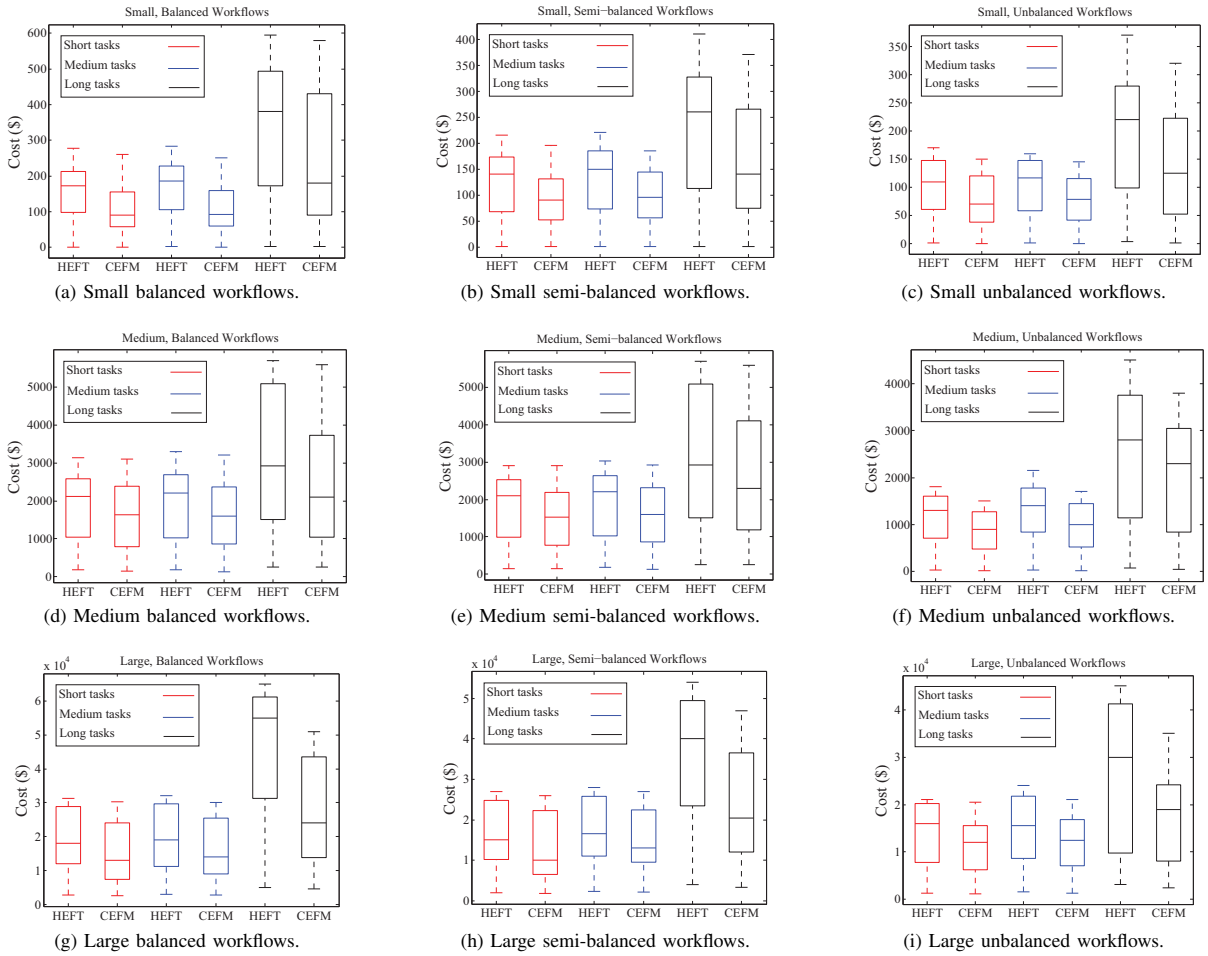
[5] J. Yu, R. Buyya, and C. K. Tham, Cost-based scheduling of scientific workflow applications on utility Grids, 1st International Conference on e-Science and Grid Computing, IEEE Computer Society Press, pp. 140-147, 2005.

[6] J. Yu, M. Kirley, and R. Buyya, Multi-objective planning for workflow execution on Grids, 8th IEEE International Conference on Grid Computing, pp. 10-17, 2007.

[7] R. Van den Bossche, K. Vanmechelen and J. Broeckhove, Cost-optimal scheduling in hybrid IaaS Clouds for deadline constrained workloads, 3rd International Conference on Cloud Computing (CLOUD), pp.228-235, July 2010.

[8] M. Amini Salehi and R. Buyya, Adapting Market-Oriented Scheduling Policies for Cloud Computing, 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010), LNCS 6081, Springer, pp. 351-362, May 2010.

[9] T. A. Henzinger, A. V. Singh, V. Singh, T. Wies and D. Zufferey, Static scheduling in Clouds, 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '11), 2011

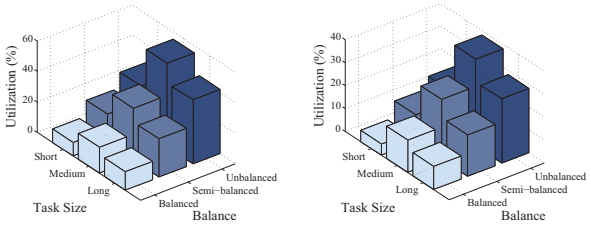


(a) Small balanced workflows. (b) Small semi-balanced workflows. (c) Small unbalanced workflows.

(d) Medium balanced workflows. (e) Medium semi-balanced workflows. (f) Medium unbalanced workflows.

(g) Large balanced workflows. (h) Large semi-balanced workflows. (i) Large unbalanced workflows.

Fig. 3: Dispersion of cost of CEFM and HEFT for synthetic workflows.



(a) CEFM. (b) HEFT.

Fig. 4: Average utilization for synthetic workflows.

[10] E. Juhnke, T. Drnemann, D. Bck, B. Freisleben, Multi-objective scheduling of BPEL workflows in geographically distributed Clouds, IEEE International Conference on Cloud Computing (CLOUD) 2011, pp. 412-419.

[11] E. Zitzler, M. Laumanns, and L. Thiele, SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization, Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems, pp. 95-100, International Center for Numerical Methods in Engineering, 2002.

[12] A. Oprescu, T. Kielmann, Bag-of-tasks scheduling under budget constraints, 2nd International Conference on Cloud Computing Technology and Science (CloudCom), pp. 351-359, 2010.

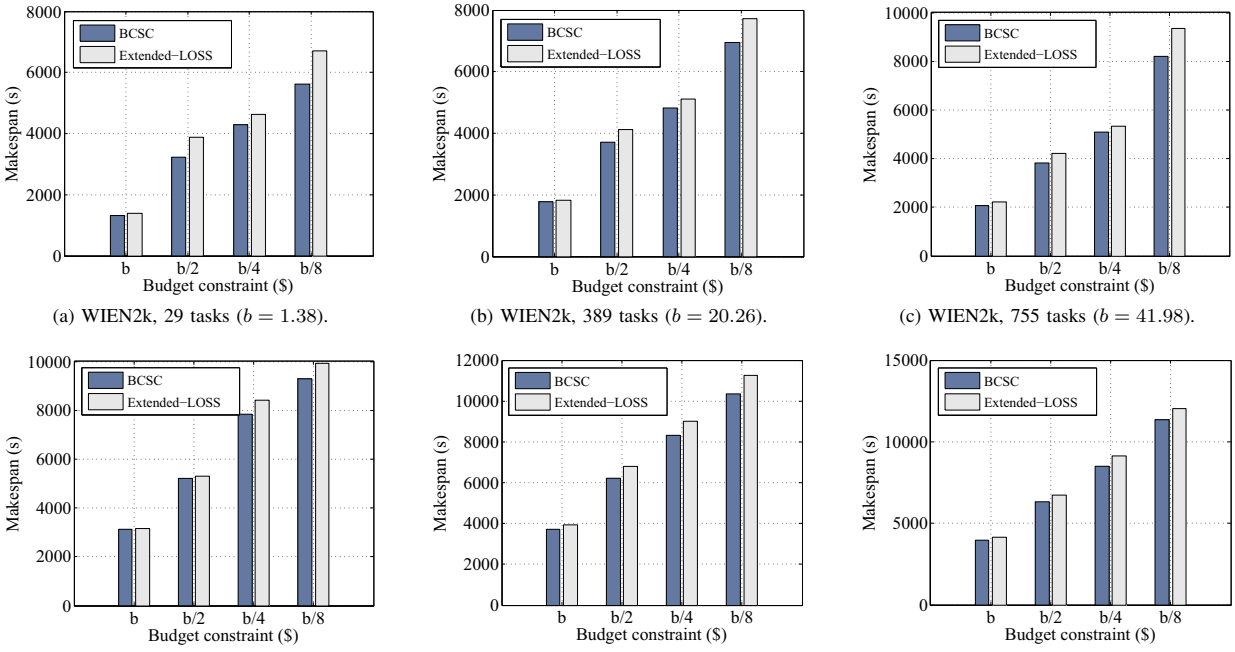
[13] M. Mao and M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, International Conference on High Performance Computing Networking, Storage and Analysis (SC '11), 2011, ACM.

[14] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12), ACM.

[15] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. L. Truong, A. Villazon, and M. Wieczorek, ASKALON: a Grid application development and computing environment, 6th International Conference on Grid Computing, pp. 122-131, November 2005, IEEE.

[16] S. Ostermann, K. Plankensteiner, R. Prodan and T. Fahringer, GroudSim: An event-based simulation framework for computational Grids and Clouds, CoreGRID/ERCIM Workshop on Grids and Clouds, Springer, August 2010.

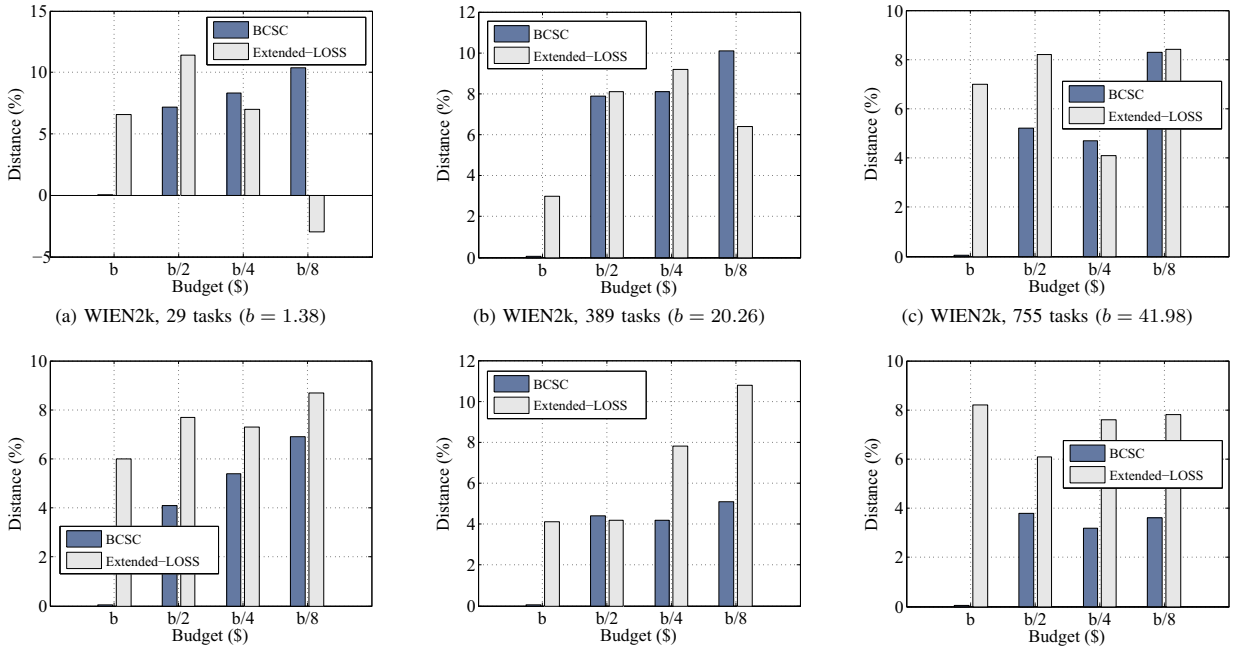
[17] H. Mohammadi Fard, R. Prodan and T. Fahringer, A truthful dynamic workflow scheduling mechanism for commercial multi-Cloud environments, IEEE Transactions on Parallel and Distributed Systems, 24(6), pp. 1203-1212, June 2013.



(a) WIEN2k, 29 tasks ($b = 1.38$). (b) WIEN2k, 389 tasks ($b = 20.26$). (c) WIEN2k, 755 tasks ($b = 41.98$).

(d) Povray, 15 tasks ($b = 1.3$). (e) Povray, 62 tasks ($b = 6.82$). (f) Povray, 169 tasks ($b = 17.66$).

Fig. 5: Makespan of WIEN2k and Povray workflows.



(a) WIEN2k, 29 tasks ($b = 1.38$) (b) WIEN2k, 389 tasks ($b = 20.26$) (c) WIEN2k, 755 tasks ($b = 41.98$)

(d) Povray, 15 tasks ($b = 1.3$) (e) Povray, 62 tasks ($b = 6.82$) (f) Povray, 169 tasks ($b = 17.66$)

Fig. 6: Distance metric for WIEN2k and Povray workflows.

[18] H. Topcuouglu, S. Hariri, M. you Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems, 13(3), pp. 260-274, 2002.