Future Generation Computer Systems 48 (2015) 1–18

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds

Maciej Malawski [a,*], Gideon Juve [b], Ewa Deelman [b], Jarek Nabrzyski [c]

[a] AGH University of Science and Technology, Department of Computer Science, Al. Mickiewicza 30, Krakow, Poland
[b] USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, USA
[c] University of Notre Dame, Center for Research Computing, 111 ITC, Notre Dame, IN, USA

HIGHLIGHTS

- We define the problem of scheduling prioritized workflow ensembles on IaaS clouds.
- We analyze and develop several dynamic (online) and static (offline) algorithms.
- We address both problems of task scheduling and resource provisioning.
- Our simulator models uncertainties in estimates, provisioning delays, and failures.
- We use synthetic workflow ensembles based on important, real scientific applications.

ARTICLE INFO

Article history:
Received 4 September 2013
Received in revised form
24 October 2014
Accepted 14 January 2015
Available online 9 February 2015

Keywords:
Scientific workflows
Workflow ensembles
Cloud computing
Resource provisioning

ABSTRACT

Large-scale applications expressed as scientific workflows are often grouped into ensembles of inter-related workflows. In this paper, we address a new and important problem concerning the efficient management of such ensembles under budget and deadline constraints on Infrastructure as a Service (IaaS) clouds. IaaS clouds are characterized by on-demand resource provisioning capabilities and a pay-per-use model. We discuss, develop, and assess novel algorithms based on static and dynamic strategies for both task scheduling and resource provisioning. We perform the evaluation via simulation using a set of scientific workflow ensembles with a broad range of budget and deadline parameters, taking into account task granularity, uncertainties in task runtime estimations, provisioning delays, and failures. We find that the key factor determining the performance of an algorithm is its ability to decide which workflows in an ensemble to admit or reject for execution. Our results show that an admission procedure based on workflow structure and estimates of task runtimes can significantly improve the quality of solutions.

1. Introduction

Large-scale scientific workflows, usually represented as Directed Acyclic Graphs (DAGs), are an important class of applications that lead to challenging problems in resource management on grid and cloud infrastructures. In addition, workflows for large computational problems are often composed of several inter-related workflows grouped into *ensembles*. Workflows in an ensemble typically have a similar structure, but they differ in their input data, number of tasks, and individual task sizes.

There are many applications that require scientific workflow ensembles. CyberShake [1], for example, uses ensembles to generate seismic hazard maps. Each workflow in a CyberShake ensemble generates a hazard curve for a particular geographic location, and several hazard curves are combined to create a hazard map. In a 2013 study, CyberShake was used to generate a set of hazard maps over 286 sites that required an ensemble of 2288 workflows [2]. Similarly, users of Montage [3] often need several workflows with different parameters to generate a set of image mosaics that can be combined into a single, large mosaic. The Galactic Plane ensemble [4], which generates several mosaics in different wavelengths, consists of 17 workflows, each of which contains 900 sub-workflows. Another ensemble example is the Periodograms application [5], which searches for extrasolar planets by detecting periodic dips in the light intensity of their host star. Due to the large scale of the input data, this application is

* Corresponding author.
E-mail addresses: malawski@agh.edu.pl (M. Malawski), gideon@isi.edu (G. Juve), deelman@isi.edu (E. Deelman), naber@nd.edu (J. Nabrzyski).

http://dx.doi.org/10.1016/j.future.2015.01.004
0167-739X/

often split up into multiple batches processed by different workflows. Additional workflows are created to run the analysis using different parameters. An example analysis of Kepler satellite data required three ensembles of 15 workflows, each workflow with about 14,000 tasks.

Workflows in an ensemble may differ not only in their parameters, but also in their priority. For example, in CyberShake some geographic locations may be in heavily populated areas or have sensitive facilities such as power plants, while others may be less important. Scientists typically prioritize the workflows in such an ensemble so that important workflows are finished first. This enables them to see critical results early, and helps them to choose the most important workflows when the time and financial resources available for computing are limited.

Infrastructure-as-a-Service (IaaS) clouds offer the ability to provision resources on-demand according to a pay-per-use model. These systems are regarded by the scientific community as a potentially attractive source of low-cost computing resources [6,7]. In contrast to clusters and grids, which typically offer best-effort quality of service, clouds give more flexibility in creating a controlled and managed computing environment. Clouds provide the ability to adjust resource capacity according to the changing demands of the application, often called auto-scaling. However, giving users more control also requires the development of new methods for task scheduling and resource provisioning. Resource management decisions required in cloud scenarios not only have to take into account performance-related metrics such as workflow makespan or resource utilization, but must also consider budget constraints, since commercial cloud resources usually have monetary costs associated with them [8].

In this paper, we aim to gain insight into resource management challenges when executing scientific workflow ensembles on clouds. We address a new and important problem of maximizing the number of completed workflows from an ensemble under both budget and deadline constraints. The motivation for this work is to answer the fundamental question of concern to a researcher: *How much computation can be completed given the limited budget and timeframe of a research project?*

The main contributions of this paper are:

- we define the problem of scheduling prioritized workflow ensembles under budget and deadline constraints,
- we analyze and develop several dynamic (online) and static (offline) algorithms for task scheduling and resource provisioning that rely on workflow structure information (critical paths and workflow levels) and estimates of task runtimes,
- we evaluate these algorithms using a simulator based on CloudSim [9], which models the infrastructure and the application, taking into account uncertainties in task runtime estimates, provisioning delays, and failures,
- we discuss the performance of the algorithms on a set of synthetic workflow ensembles based on important, real scientific applications, using a broad range of different application scenarios and varying constraint values.

This paper is an extension of our earlier conference publication [10]. Here we present a more detailed performance evaluation, including the discussion of results based on varying deadline and budget constraints. We also report on new results and discussion of task granularity in the context of the resource billing cycle, which is an important aspect of scheduling and resource provisioning on cloud infrastructures. The related work section and the algorithm examples have been also extended to provide a better positioning and increase the readability of the paper.

The paper is organized as follows: after an analysis of related work in Section 2, Section 3 describes the infrastructure and application model we are targeting in this paper. In Section 4 we describe the dynamic and static algorithms we developed. Section 5 presents test scenarios and performance metrics, while Section 6 discusses the results of simulation studies of these algorithms. Finally, general conclusions, lessons learned, and future work are outlined in Section 7.

2. Related work

General policy-based and rule-based approaches to dynamic provisioning (such as Amazon Auto Scaling [11] and RightScale [12]) allow the size of a resource pool to be adjusted based on infrastructure and application metrics. A typical infrastructure-specific metric is system load, whereas application-specific metrics include response time and length of a task or of a request queue. It is possible to set thresholds and limits to tune the behavior of these autoscaling systems, but no support for complex applications is provided.

Policy-based approaches for scientific workloads (e.g. [13–15]) also enable scaling the cloud resource pool or the extension of the capabilities of clusters using cloud-burst techniques. Our approach is different in that we consider workflows, while policy-based approaches typically consider bags of independent tasks or unpredictable batch workloads. This enables us to take advantage of workflow-aware heuristics that cannot be applied to independent tasks.

Our work is related to the strategies for deadline-constrained cost-minimization workflow scheduling, developed for utility grid and cloud systems. However, our problem is different from [16–18] in that we consider ensembles of workflows instead of individual workflows. Our work is also different from cloud-targeted autoscaling solution [19] in that we consider ensembles of workflows rather than unpredictable workloads containing workflows. We also consider budget constraints rather than cost minimization as a goal. In other words, we assume that there is more work to be done than the available budget, so some work must be rejected. Therefore, cost is not something we optimize (i.e. an objective), but rather a constraint. In [20] authors introduced a novel single DAG static scheduling PEFT algorithm, which outperforms well-known list-based heuristics for makespan optimization, but the budget is not taken into account.

This work is related to bi-criteria scheduling and multi-criteria scheduling of workflows [21–25]. These approaches are similar to ours in that we have two scheduling criteria: cost and makespan. The challenge in multi-criteria scheduling is to derive an objective function that takes into account all of the criteria. In our case one objective (amount of work completed) is subject to optimization, whereas time and cost are treated as constraints, which is not addressed by other works.

Our work can also be regarded as an extension of budget-constrained workflow scheduling [26], cost optimization for workflows under deadline constraints [16], or methods of stretching and compacting the workflow to optimize time and cost [18], in the sense that we are dealing with workflow ensembles and the deadline constraint is added. An interesting discussion of uncertainties and the way to address them using a Monte Carlo approach is also presented in [27], where the objective is makespan optimization.

Scheduling optimization for multiple workflows has been also considered in previous research, either in the context of heterogeneous systems [28], where makespan and fairness optimization is applied, or in the context of grids [29], where multiple heuristics are evaluated with respect to the makespan of schedules they produce. Scheduling multiple workflows in [30,31] applies multiobjective optimization of time and cost under storage bandwidth and capacity constraints with the use of heuristics based on game theory. The approach proposed in [32] focuses on enforcement of QoS

measured as throughput of processing by individual workflows. The problem of workflow admission addressed in [33] is similar to the one solved by our algorithms, but the difference is that we address the problem from the end-user perspective, while in their work they assume that the resource provider decides whether to admit or reject the workflow based on the resource estimates and the deadline constraints.

Other approaches for scheduling workflows on grids and clouds such as [34–36] use metaheuristics that usually run for a long time before producing good results, which makes them less useful in the scenarios we consider in this paper.

Also noteworthy are methods based on mathematical programming, using linear programming or mixed-integer programming. Such a method is applied to scheduling workflows on hybrid clouds using time discretization in [37], but the model presented there is limited to small scale workflows. Large-scale applications on hybrid clouds are addressed in [38,39], but their model considers bag-of-task applications instead of workflows. The cloud bursting scenario described in [40], where a private cloud is combined with a public one, addresses workflows, and the goal is to minimize the cost while satisfying the deadline. In our related paper [41], we also use the mixed-integer programming approach to schedule multi-level deadline-constrained workflows, but workflow ensembles are not considered.

The analysis of related work is summarized in Table 1, where we classify the methods based on 7 categories. The infrastructure type can be a grid or a cloud, where a grid includes utility grids and heterogeneous computing systems, where cost can be relevant (such as in [26]). Application types of interest are bags of tasks, single workflows, or multiple workflows. The workload can be static, where the scheduling process can plan the tasks in advance, or dynamic, where the algorithm has to deal with unpredictable stream of jobs arriving. The provisioning methods are divided into dynamic and static approaches, where static means that the provisioning plan is prepared prior to execution. To simplify, we classify all the algorithms for grid and heterogeneous systems into the static category, since these systems do not allow for creation of new resources on demand, even if some mechanisms for dynamic provisioning such as pilot jobs [42] are available. The related work addresses multiple optimization objectives, including time, monetary cost, reliability, energy or fairness, and there are examples of multi-criteria optimization among them. Most of the scheduling approaches also consider constraints for optimization, and the most commonly addressed ones are budget and deadline. Other constraints, such as bandwidth, storage capacity or job throughput, are also present. Finally, we divide the algorithmic methods into heuristics, including various list scheduling techniques based on graph analysis, metaheuristics that use genetic or evolutionary algorithms, and the approaches using mathematical programming such as linear or mixed-integer programming for which ready to use solvers exist.

An analysis of the table suggests that there are many approaches to workflow scheduling and resource provisioning on clouds, but there is still room for improvement. New infrastructure types, including hybrid and federated clouds should be investigated, more emphasis should be put on problems with multiple interrelated workflows, and new methods for combining static and dynamic workloads and algorithms offer potentially interesting areas of research. Finally, other objectives and constraints in addition to budget and execution time can be explored.

In this context, we consider our attempt to address workflow ensembles using both static and dynamic algorithms presented in this paper an interesting and original contribution that has not been explored before.

3. Problem description

3.1. Resource model

We assume a resource model similar to Amazon's Elastic Compute Cloud (EC2), where virtual machine (VM) instances may be provisioned on-demand and are billed by the hour, with partial hours being rounded up. Although there may be heterogeneous VM types with different amounts of CPU, memory, disk space, and I/O, for this paper we focus on a single VM type because we assume that for most applications there will typically be only one or two VM types with the best price/performance ratio for the application [43]. We assume that a submitted task has exclusive access to a VM instance and that there is no preemption. We also assume that there is a delay between the time that a new VM instance is requested and when it becomes available to execute tasks.

3.2. Application model

The target applications are ensembles of scientific workflows that can be modeled as DAGs, where the nodes in the graph represent computational tasks, and the edges represent data- or control-flow dependencies between the tasks. We assume that runtime estimates for the workflow tasks are known, but that they are not perfect and may vary based on a uniform distribution of $\pm p\%$.

This study uses synthetic workflows that were generated using historical data from real applications [44]. The applications come from a wide variety of domains including: bioinformatics (Epigenomics, SIPHT: sRNA identification protocol using high-throughput technology), astronomy (Montage), earthquake science (CyberShake), and physics (LIGO). The synthetic workflows were generated using code developed in [45], with task runtimes based on distributions gathered from running real workflows.

Although workflows are often data-intensive, the algorithms described here do not currently consider the size of input and output data when scheduling tasks. Instead, it is assumed that all workflow data is stored in a shared cloud storage system, such as Amazon S3, and that intermediate data transfer times are included in task runtimes. It is also assumed that data transfer times between the shared storage and the VMs are equal for different VMs so that task placement decisions do not impact the runtime of the tasks.

We assume that each workflow in an ensemble is given a numeric priority that indicates how important the workflow is to the user. As such, the priorities indicate the utility function of the user. These priorities are absolute in the sense that completing a workflow with a given priority is more valuable than completing all other workflows in the ensemble with lower priorities combined. The goal of the workflow ensemble scheduling and cloud provisioning problem is to complete as many high-priority workflows as possible given a fixed budget and deadline. Only workflows for which all tasks are finished by the deadline are considered to be complete — partial results are not usable in this model.

3.3. Performance metric

In order to precisely define the objective of the algorithms it is necessary to introduce a metric that can be used to score the performance of the different algorithms on a given problem (ensemble, budget, and deadline). The simplest approach is to count the number of workflows in the ensemble that each algorithm is able to complete within the budget before the deadline, but this metric does not account for the priority-based utility function specified by the user. Using the counting approach, a less efficient algorithm

Table 1
Summary table of related work on workflow scheduling and provisioning.

	Infrastructure		Application type			Work load		Provisioning		Objectives					Constraints					Method		
	Grid	Cloud	Bags of tasks	Workflow	Multiple workflows	Static	Dynamic	Static	Dynamic	Time	Cost	Reliability	Energy	Fairness	Budget	Deadline	Bandwidth	Storage	Throughput	Heuristics	Metaheuristics	Mathematical programming
Yu et al. '05 [16]	+	−	−	+	−	+	−	+	−	−	+	−	−	−	−	+	−	−	−	+	−	−
Zhao et al. '06 [28]	+	−	−	−	+	+	−	+	−	+	−	−	−	+	−	−	−	−	−	+	−	−
Sakellariou et al. '07 [26]	+	−	−	+	−	+	−	+	−	+	−	−	−	−	+	−	−	−	−	+	−	−
Dongarra et al. '07 [23]	+	−	−	+	−	+	−	+	−	+	−	+	−	−	−	−	−	−	−	+	−	−
Wieczorek et al. '09 [21]	+	−	−	+	−	+	−	+	−	+	+	−	−	−	−	−	−	−	−	+	−	−
Prodan et al. '10 [22]	+	−	−	+	−	+	−	+	−	+	+	−	−	−	−	−	−	−	−	+	−	−
Pandey et al. '10 [35]	−	+	−	+	−	+	−	+	−	+	+	−	−	−	−	−	−	−	−	−	+	−
Marshall et al. '10 [13]	−	+	+	−	−	−	+	−	+	−	−	−	−	−	−	−	−	−	−	+	−	−
Van den Bossche et al. '10 [38]	−	+	+	−	−	−	+	+	−	−	+	−	−	−	−	+	−	−	−	−	−	+
Hirales-Carbajal et al. '10 [29]	+	−	−	−	+	+	−	+	−	+	−	−	−	−	−	−	−	−	−	+	−	−
Mao et al. '11 [19]	−	+	−	+	−	−	+	−	+	−	+	−	−	−	−	+	−	−	−	+	−	−
Kim et al. '11 [14]	+	−	+	−	−	+	−	−	+	−	−	−	−	−	−	−	−	−	−	+	−	−
Bittencourt et al. '11 [40]	−	+	−	+	−	+	−	+	−	−	+	−	−	−	−	+	−	−	−	−	−	+
Duan et al. '12 [30]	+	−	−	−	+	+	−	+	−	+	+	−	−	−	−	−	−	−	−	+	−	−
Durillo et al. '12 [25]	−	+	−	+	−	+	+	+	−	+	+	+	+	−	−	−	−	−	−	+	−	−
Tolosana-Calasanz et al. '12 [32]	−	+	−	−	+	−	+	−	+	−	−	−	−	−	−	−	−	−	+	−	+	−
Van den Bossche et al. '13 [39]	−	+	+	−	−	−	+	+	−	−	+	−	−	−	−	+	−	−	−	−	−	+
Lee et al. '13 [18]	−	+	−	+	−	+	−	+	−	+	+	−	−	−	−	−	−	−	−	+	−	−
Genez et al. '13 [37]	−	+	−	+	−	+	−	+	−	−	+	−	−	−	−	+	−	−	−	−	−	+
Talukder et al. '13 [34]	+	−	−	+	−	+	−	+	−	+	+	−	−	−	−	−	−	−	−	−	+	−
Abrishami et al. '13 [17]	−	+	−	+	−	+	−	+	−	−	+	−	−	−	−	+	−	−	−	+	−	−
Durillo et al. '13 [24]	+	−	−	+	−	+	−	−	+	+	−	−	+	−	−	−	−	−	−	+	−	−
Zheng et al. '13a [33]	−	+	−	−	+	−	+	+	−	−	−	−	−	−	+	+	−	−	−	+	−	−
Zheng et al. '13b [27]	+	−	−	+	−	+	−	+	−	+	−	−	−	−	−	−	−	−	−	+	−	−
Arabnejad et al. '14 [20]	+	−	−	+	−	+	−	+	−	+	−	−	−	−	−	−	−	−	−	+	−	−
Duan et al. '14 [31]	−	+	+	+	−	+	−	+	−	+	+	−	−	−	−	−	+	+	−	+	−	−
Durillo et al. '14 [36]	−	+	−	+	−	+	−	+	−	+	+	−	−	−	−	−	−	−	−	+	+	−

Algorithm 1 Dynamic provisioning algorithm for DPDS

Require: c: consumed budget; b: total budget; d: deadline; p: price; t: current time; u_h: upper utilization threshold; u_l: lower utilization threshold; v_{max}: maximum number of VMs

```
1: procedure PROVISION
2:     V_R ← set of running VMs
3:     V_C ← set of VMs completing billing cycle
4:     V_T ← ∅                                   ▷ set of VMs to terminate
5:     n_T ← 0                                   ▷ number of VMs to terminate
6:     if b − c < |V_C| ∗ p or t > d then
7:         n_T ← |V_R| − ⌊(b − c)/p⌋
8:         V_T ← select n_T VMs to terminate from V_C
9:         TERMINATE(V_T)
10:    else
11:        u ← current VM utilization
12:        if u > u_h and |V_R| < v_max ∗ N_VM then
13:            START(new VM)
14:        else if u < u_l then
15:            V_I ← set of idle VMs
16:            n_T ← ⌈|V_I|/2⌉
17:            V_T ← select n_T VMs to terminate from V_I
18:            TERMINATE(V_T)
19:        end if
20:    end if
21: end procedure
```

may be able to complete a large number of low-priority workflows by executing the smallest workflows first. In order to account for the priority, we use an exponential scoring defined as:

$$Score(e) = \sum_{w \in Completed(e)} 2^{-Priority(w)} \quad (1)$$

where $Completed(e)$ is the set of workflows in ensemble e that was completed by the algorithm, and $Priority(w)$ is the priority of workflow w such that the highest-priority workflow has $Priority(w) = 0$, the next highest workflow has $Priority(w) = 1$, and so on. This exponential scoring function gives the highest priority workflow a score that is higher than all the lower-priority workflows combined:

$$2^{-p} > \sum_{i=p+1,\ldots} 2^{-i}.$$

This scoring is consistent with our definition of the problem, which is to complete as many workflows as possible, according to their priorities, given a set budget and deadline.

4. Algorithms

This section describes three algorithms that were developed to schedule and provision resources for ensembles of workflows on the cloud under budget and deadline constraints.

4.1. Dynamic provisioning dynamic scheduling (DPDS)

DPDS is an online algorithm that provisions resources and schedules tasks at runtime. It consists of two main parts: a provisioning procedure, and a scheduling procedure.

DPDS' provisioning procedure is based on resource utilization. DPDS starts with a fixed number of resources calculated based on the available time and budget, and adjusts the number of resources according to how well they are utilized by the application. Given a budget in dollars b, deadline in hours d, and the hourly price of a VM in dollars p, it is possible to calculate the number of VMs, N_{VM}, to provision so that the entire budget is consumed before the deadline:

$$N_{VM} = \lceil b/(d * p) \rceil. \quad (2)$$

DPDS provisions N_{VM} VMs at the start of the ensemble execution, then it periodically computes resource utilization using the percentage of idle VMs over time and adjusts the number of VMs if the utilization is above or below given thresholds. Because it is assumed that VMs are billed by the hour, DPDS only considers VMs that are approaching their hourly billing cycle when deciding which VMs to terminate. This dynamic provisioning algorithm is shown in Algorithm 1.

The set of VMs completing their billing cycle is determined by both the provisioner interval, and the termination delay of the provider. This guarantees that VMs can be terminated before they start the next billing cycle and prevents the budget from being overrun. The VMs terminated in line 9 of Algorithm 1 are the ones that would overrun the budget if not terminated in the current provisioning cycle. The VMs terminated in line 18 are chosen to increase the resource utilization to the desired threshold. In order to prevent instances that have already been paid for from being terminated too quickly, no more than half of the idle resources are terminated during each provisioning cycle. To avoid an uncontrolled increase in the number of instances, which may happen in the case of highly parallel workflows, the provisioner will not start a new VM if the number of running VMs is greater than the product of N_{VM} (from Eq. (2)) and an autoscaling parameter, v_{max}. Unless otherwise specified, v_{max} is assumed to be 1.

Algorithm 2 Priority-based scheduling algorithm for DPDS

```
1: procedure SCHEDULE
2:     P ← empty priority queue
3:     IdleVMs ← set of idle VMs
4:     for root task t in all workflows do
5:         INSERT(t, P)
6:     end for
7:     while deadline not reached do
8:         while IdleVMs ≠ ∅ and P ≠ ∅ do
9:             v ← SELECTRANDOM(IdleVMs)
10:            t ← POP(P)
11:            SUBMIT(t, v)
12:        end while
13:        Wait for task t to finish on VM v
14:        Update P with ready children of t
15:        INSERT(v, IdleVMs)
16:    end while
17: end procedure
```

In order to schedule individual workflow tasks onto available VMs, DPDS uses the dynamic, priority-based scheduling procedure shown in Algorithm 2. Initially, the ready tasks from all workflows in the ensemble are added to a priority queue based on the priority of the workflow to which they belong. If there are idle VMs available, and the priority queue is not empty, the next task from the priority queue is submitted to an arbitrarily chosen idle VM. The process is repeated until there are no idle VMs or the priority queue is empty. The scheduler then waits for a task to finish, adds its ready children to the priority queue, marks the VM as idle, and the entire process repeats until the deadline is reached.

DPDS guarantees that tasks from lower priority workflows are always deferred when higher-priority tasks are available, but lower-priority tasks can still occupy idle VMs when higher-priority tasks are not available. Since there is no preemption, long-running low-priority tasks may delay the execution of higher-priority tasks. In addition, tasks from low priority workflows may be executed even though there is no chance that those workflows will be completed within the current budget and deadline.

4.2. Workflow-aware DPDS (WA-DPDS)

DPDS does not use any information about the structure of the workflows in the ensemble when scheduling tasks. It does not consider whether a lower priority task belongs to a workflow that will never be able to complete given the current budget and

Algorithm 3 Workflow admission algorithm for WA-DPDS

Require: w: workflow; b: budget; c: current cost
1: **procedure** ADMIT(w, b, c)
2: $r_n \leftarrow b - c$ ▷ Budget remaining for new VMs
3: $r_c \leftarrow$cost committed to VMs that are running
4: $r_a \leftarrow$cost to complete workflows previously admitted
5: $r_m \leftarrow 0.1$ ▷ Safety margin
6: $r_b \leftarrow r_n + r_c - r_a - r_m$ ▷ Budget remaining
7: $c_w \leftarrow$ ESTIMATECOST(w)
8: **if** $c_w < r_b$ **then return** *TRUE*
9: **else return** *FALSE*
10: **end if**
11: **end procedure**

Algorithm 4 Ensemble planning algorithm for SPSS

Require: W: workflow ensemble; b: budget; d: deadline
Ensure: Schedule as much of W as possible given b and d
1: **procedure** PLANENSEMBLE(W, b, d)
2: $P \leftarrow \emptyset$ ▷ Current plan
3: $A \leftarrow \emptyset$ ▷ Set of admitted DAGs
4: **for** w *in* W **do**
5: $P' \leftarrow$ PLANWORKFLOW(w, P, d)
6: **if** $Cost(P') \leq b$ **then**
7: $P \leftarrow P'$ ▷ Accept new plan
8: $A \leftarrow A + w$ ▷ Admit w
9: **end if**
10: **end for**
11: **return** P, A
12: **end procedure**

deadline. As a result, DPDS may start lower priority tasks just to keep VMs busy that will end up delaying higher priority tasks later on, making it less likely that higher priority workflows will be able to finish.

The Workflow-Aware DPDS (WA-DPDS) algorithm extends DPDS by introducing a workflow admission procedure, which is invoked whenever WA-DPDS sees the first task of a new workflow at the head of the priority queue (i.e. when no other tasks from the workflow have been scheduled yet). The admission procedure – shown in Algorithm 3 – estimates whether there is enough budget remaining to admit the new workflow; if there is not, then the workflow is rejected and its tasks are removed from the queue. WA-DPDS compares the current cost (consumed budget) and remaining budget, taking into account the cost of currently running VMs, and the cost of workflows that have already been admitted. In addition, it adds a small safety margin of $0.10 (10% of the compute hour cost) to avoid going over budget. We found the admission procedure useful not only to prevent low-priority workflows from delaying high-priority ones, but also to reject large and costly workflows that would overrun the budget and admit smaller workflows that can efficiently utilize idle resources in ensembles containing workflows of non-uniform sizes. It would also be possible to extend this admission procedure to check other constraints, such as whether the estimated critical path of the new workflow exceeds the time remaining until the deadline.

4.3. Static provisioning static scheduling (SPSS)

The previous dynamic (online) algorithms make provisioning and scheduling decisions at runtime. By contrast, the SPSS algorithm creates a provisioning and scheduling plan before running any workflow tasks. This enables SPSS to start only those workflows that it knows can be completed given the deadline and budget constraints, and eliminates any waste that may be allowed by the dynamic algorithms.

The approach used by SPSS is to plan each workflow in the ensemble in priority order, rejecting any workflows that would exceed the deadline or budget. Once the plan is complete, the VMs are provisioned and tasks are executed according to the schedules given by the plan.

The disadvantage of the static planning approach used by SPSS is that it is sensitive to dynamic changes in the environment and the application that may disrupt its carefully constructed plan. For example, if there are provisioning delays, or if the runtime estimates for the tasks are inaccurate, then workflow execution may diverge from the plan. This issue will be discussed further in Sections 6.4 and 6.5.

Algorithm 4 shows how ensembles are planned in SPSS. Workflows from the ensemble are considered in priority order. For each workflow, SPSS attempts to build on top of the current plan by provisioning VMs to schedule the tasks of the workflow so that it finishes before the deadline with the least possible cost. If the cost of the new plan is less than the budget, then the new plan is accepted and the workflow is admitted. If not, then the new plan is rejected and the process continues with the next workflow in the ensemble. The idea is that, if each workflow can be completed by the deadline with the lowest possible cost, then the number of workflows that can be completed within the given budget will be maximized.

To plan a workflow, the SPSS algorithm assigns sub-deadlines to each individual task in the workflow, and then schedules each task so as to minimize the cost of the task while still meeting its assigned sub-deadline. If each task can be completed by its deadline in the least expensive way, then the cost of the entire workflow can be minimized without exceeding the deadline. SPSS assigns sub-deadlines to each task based on the slack time of the workflow, which is defined as the amount of extra time that a workflow can extend its critical path and still be completed by the ensemble deadline. For a workflow w, the slack time of w is: $ST(w) = d - CP(w)$ where d is the deadline and $CP(w)$ is the critical path of w. We assume that $CP(w) \leq d$, otherwise the workflow cannot be completed by the deadline and must be rejected. For large ensembles we expect the critical path of any individual workflow to be much less than the deadline.

A task's level is the length of the longest path between the task and an entry task of the workflow:

$$Level(t) = \begin{cases} 0, & \text{if } Pred(t) = \emptyset \\ \max\limits_{p \in Pred(t)} Level(p) + 1, & \text{otherwise.} \end{cases}$$

SPSS distributes the slack time of the workflow by level, so that each level of the workflow gets a portion of the workflow's slack time proportional to the number of tasks in the level and the total runtime of tasks in the level. The idea is that levels containing many tasks and large runtimes should be given a larger portion of the slack time so that tasks in those levels may be serialized. Otherwise, many resources need to be allocated to run all of the tasks in parallel, which may be more costly.

The slack time of a level l in workflow w is given by:

$$ST(l) = ST(w)\left[\left(\alpha \frac{N(l)}{N(w)}\right) + \left((1-\alpha)\frac{R(l)}{R(w)}\right)\right]$$

where $N(w)$ is the number of tasks in the workflow, $N(l)$ is the number of tasks in level l, $R(w)$ is the total runtime of all tasks in the workflow, $R(l)$ is the total runtime of all tasks in level l, and α is a parameter between 0 and 1 that causes more slack time to be given to levels with more tasks (large α) or more runtime (small α).

The deadline of a task t is then:

$$DL(t) = LST(t) + RT(t) + ST(Level(t)) \quad (3)$$

where $Level(t)$ is the level of t, $RT(t)$ is the runtime of t, and $LST(t)$ is the latest start time of t determined by:

$$LST(t) = \begin{cases} 0, & \text{if } Pred(t) = \emptyset \\ \max\limits_{p \in Pred(t)} DL(p), & \text{otherwise.} \end{cases}$$

Algorithm 5 Workflow planning algorithm for SPSS

Require: w: workflow; P: current plan; d: deadline
Ensure: Create plan for w that minimizes cost and meets deadline d

```
1:  procedure PlanWorkflow(w, P, d)
2:      P' ← copy of P
3:      DeadlineDistribution(w,d)
4:      for t in w sorted by DL(t) do
5:          v ← VM that minimizes cost and start time of t
6:          if FinishTime(t, v) < DL(t) then
7:              Schedule(t,v)
8:          else
9:              Provision a new VM v
10:             Schedule(t,v)
11:         end if
12:     end for
13:     return P'
14: end procedure
```

Algorithm 5 shows how SPSS creates low-cost plans for each workflow. The PlanWorkflow procedure first calls DeadlineDistribution to assign sub-deadlines to tasks. Then, PlanWorkflow schedules tasks onto VMs, allocating new VMs when necessary. For each task in the workflow, the least expensive slot is chosen to schedule the task so that it can be completed by its deadline. VMs are allocated in blocks of one billing cycle (one hour) regardless of the size of the task. When computing the cost of scheduling a task on a given VM, the algorithm considers idle slots in blocks that were allocated for previous tasks to be free, while slots in new blocks cost the full price of a billing cycle. For example, if a task has a runtime of 10 min, and the price of a block is $1, then the algorithm will either schedule the task on an existing VM that has an idle slot larger than 10 min for a cost of $0, or it will allocate a new block on an existing VM, or provision a new VM, for a cost of $1. If the cost of slots on two different VMs is equal, then the slot with the earliest start time is chosen. To prevent too many VMs from being provisioned, the algorithm always prefers to extend the runtime of existing VMs before allocating new VMs. The result of this is that the algorithm will only allocate a new block if there are no idle slots on existing blocks large enough or early enough to complete the task by its deadline, and it will only allocate a new VM if it cannot add a block to an existing VM to complete the task by its deadline.

4.4. Illustrative example

To illustrate how the algorithms perform, we prepared an artificial ensemble of three workflows, as seen in Fig. 1. The deadline is set to 6 h and the budget to $18. In this example we assume that there are no VM provisioning delays, the runtime estimates are accurate, there are no failures and the cost of 1 VM-hour is $1. For dynamic algorithms we assume that the low utilization threshold (u_l) for deprovisioning is 50% and that the autoscaling parameter $v_{max} = 1$.

The resulting schedule produced by DPDS algorithm is shown in Fig. 2. The algorithm at the beginning estimates that it needs 3 VM instances by dividing the budget by the deadline ($18/6 = 3$). Next, it schedules all the ready tasks in the priority order, so $a.70$ is started first, and immediately $b.70$ and $c.60$ are started, since no other tasks of workflow a are ready. When task $a.60$ completes, there are again other no higher priority tasks ready, so the next task from workflow c can be started ($c.45$). Since there is no preemption, this results in workflow a being delayed by the lower priority workflow c. Finally, the outcome of this dynamic scheduling policy is that none of the workflows completes within the deadline. Actually, tasks $b.100$, $c.65$ and $a.160$ are not finished since all the VMs are terminated due to running out of budget. The exponential score achieved by the algorithm computed according to Eq. (1) is 0. On the other hand, we can observe that the utilization

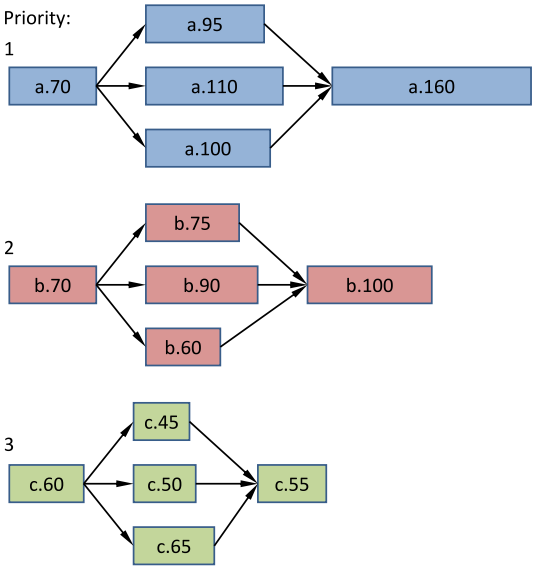


Fig. 1. Artificial ensemble consisting of three workflows $\{a, b, c\}$, with priorities $\{1, 2, 3\}$. Task labels indicate estimated execution time in minutes, so e.g. task $a.70$ has estimated execution time of 70 min.

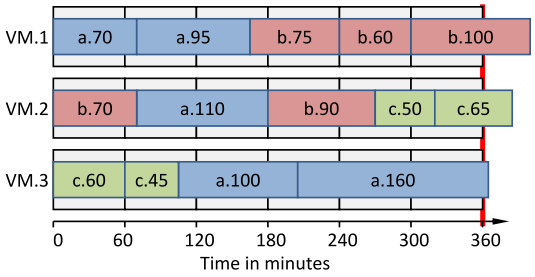


Fig. 2. Schedule produced by DPDS algorithm for the example ensemble from Fig. 1. White boxes indicate VM instances provisioned for their hourly time slots, and solid boxes indicate running tasks. None of the workflows completes within the deadline.

of the VMs is kept on the level of 100% throughout the whole execution time, which may be considered an advantage in some cases.

The resulting schedule produced by the WA-DPDS algorithm is shown in Fig. 3. The schedule is generated in a similar way as in DPDS algorithm, with one notable difference. After scheduling tasks $a.70$ and $b.70$, the algorithm estimates whether it has enough budget to admit workflow c. Since based on the sum of task runtimes there is no room for workflow c, the whole workflow is rejected. As a consequence, task $a.100$ can start at time $t = 70$ min. Therefore workflow a is not delayed as in the case of DPDS. One can also observe that $VM.2$ and $VM.3$ remain idle for one hour, which is the consequence of the conservative provisioning settings we assume in this example. This means that VM utilization at the level of $2/3$ does not trigger deprovisioning in this case, and the maximum autoscaling factor set to 1 prevents from provisioning of new VMs over the initial number even if utilization is 100%. In the resulting schedule only workflow a completes, so the score is $2^{-1} = 0.5$, which is better than DPDS.

The final schedule produced by the SPSS algorithm is shown in Fig. 4. SPSS generates this schedule by considering workflows in priority order. It first considers workflow a, and develops a low-cost schedule that completes a by its deadline using a minimum number of VM-hours. This process involves assigning a deadline to each task according to Eq. (3), and then scheduling tasks in deadline-order on the VM that can complete each task by its deadline with the minimum increase in cost. Once a has been

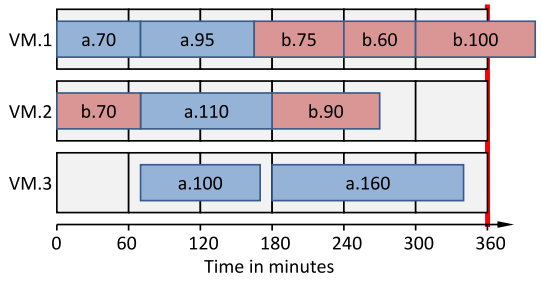


Fig. 3. Schedule produced by WA-DPDS algorithm for the example ensemble from Fig. 1. Workflow *a* completes within the deadline.

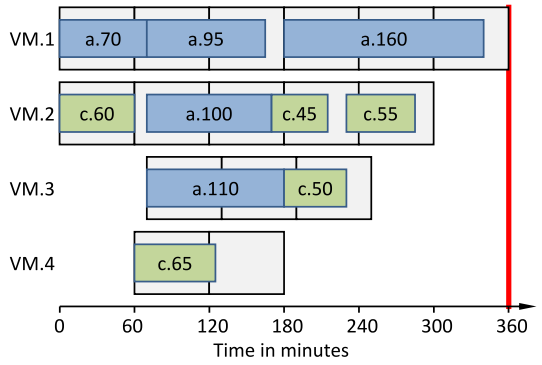


Fig. 4. Schedule produced by SPSS algorithm for the example ensemble from Fig. 1. Workflows *a* and *c* complete within the given budget and deadline.

scheduled, SPSS will develop an updated low-cost schedule that includes workflow *b*. However, after adding *b*, the algorithm will determine that the total cost of the combined schedule for *a* and *b* is greater than the budget, and will reject workflow *b* and remove it from the schedule. It will then consider workflow *c*, and develop a low-cost schedule that adds *c* to the existing schedule for *a*. In order to complete each job in *c* by its deadline, SPSS provisions 6 additional VM-hours, which includes adding an hour to the start time of *VM*.2 for task *c*.60, adding an hour to the end of *VM*.2 for task *c*.45, adding an hour to the end of *VM*.3 for *c*.50, adding a new VM, *VM*.4, for 2 h to run *c*.65, and finally adding another hour to the end of *VM*.2 for *c*.55. The resulting schedule has a total cost of 16 VM-hours, or $16, which is less than the budget of $18 and completes both *a* and *c* by the deadline. The exponential score is then $2^{-1} + 2^{-3} = 0.625$, which is better than WA-DPDS.

The example given above illustrates the main concepts of the algorithms using the artificial ensemble. Two additional example schedules generated using our simulator are shown in Fig. 5. The schedule generated using DPDS (Fig. 5(a)) illustrates how tasks from lower priority workflows backfill idle VMs when tasks from higher priority workflows are not ready for execution. On the other hand, the schedule generated by SPSS (Fig. 5(b)) shows how SPSS tends to start many workflows in parallel, running each workflow over a longer period of time on only a few VMs to minimize cost. In comparison, the dynamic algorithms tend to run one workflow at a time across many VMs in parallel.

5. Evaluation methods

5.1. Simulator

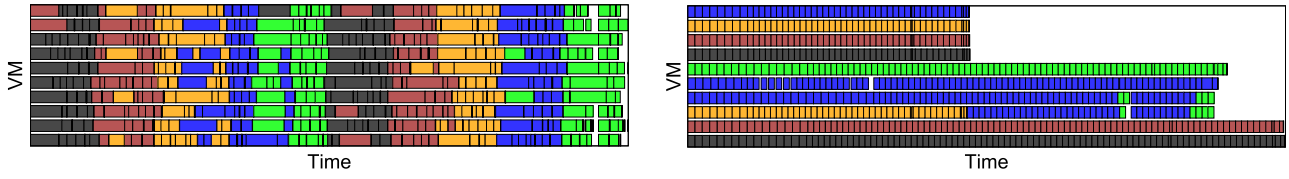
To evaluate and compare the three proposed algorithms, we developed a cloud workflow simulator [46] based on CloudSim [9]. Our simulation model consists of Cloud, VM and WorkflowEngine entities. The Cloud entity starts and terminates VM entities using an Amazon EC2-like API. VM entities simulate the execution of individual tasks, including randomized variations in runtime. The WorkflowEngine entity manages the scheduling of tasks and the provisioning of VMs based on the chosen algorithm. We assume that the VMs have a single core and execute tasks sequentially. Although CloudSim provides a more advanced infrastructure model, which includes time-sharing and space-sharing policies, we do not use these features since we are interested mainly in the execution of tasks on VMs and high-level workflow scheduling and provisioning. The simulator reads workflow description files in a modified version of the DAX format used by the Pegasus Workflow Management System [47].

The decision to use a simulator instead of actually executing workflows on clouds was motivated by our intention to perform an extensive analysis of the algorithms in various aspects. This results in a very large parameter space. For example, the analysis presented in Section 6.4 required 525,000 simulations of workflow ensembles, each ensemble consisting of 50 workflows with up to 1000 tasks per workflow. Not only would this be impractical to run on a real infrastructure for obvious reasons, but it would also not enable us to study interesting aspects of the problem, such as the influence of task granularity presented in Section 6.3. Our use of a simulator, although quite compute-intensive itself, enabled us to study all these aspects in detail.

5.2. Workflow ensembles

In order to evaluate the algorithms on a standard set of workflows, we created randomized ensembles using workflows available from the workflow generator gallery [45,48]. The gallery contains synthetic workflows modeled using structures and parameters that were taken from real applications. Ensembles were created using synthetic workflows from five real applications: SIPHT, LIGO, Epigenomics, Montage and CyberShake. For each application, workflows with 50, 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1000 tasks were created. For each workflow size, 20 different workflow instances were generated using parameters and task runtime distributions from real workflow traces. The total collection of synthetic workflows contains 5 applications, 11 different workflow sizes, and 20 workflow instances, for a total of 1100 synthetic workflows.

Using this collection of workflows, we constructed five different ensemble types: constant, uniform sorted, uniform unsorted, Pareto sorted and Pareto unsorted. In the unsorted ensembles, workflows of different sizes are mixed together and the priorities are assigned randomly. For many applications, however, large workflows are more important to users than small workflows



(a) DPDS. (b) SPSS.

Fig. 5. Example schedules generated by the algorithms. Each row is a different VM. Boxes are tasks colored by workflow. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

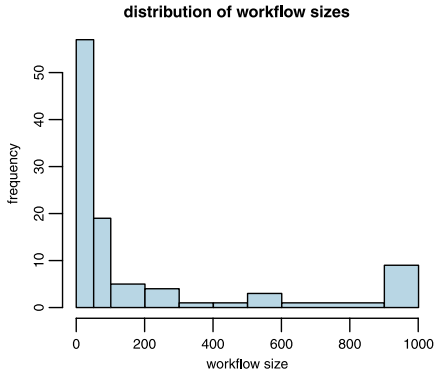


Fig. 6. Histogram of workflow sizes in Pareto ensembles. Workflow size is measured in number of tasks.

because they represent more significant computations. To model this, the sorted ensembles are sorted by size, so that the largest workflows have the highest priority.

Constant ensembles contain workflows that all have the same number of tasks. The number of tasks is chosen randomly from the set of possible workflow sizes. Once the size is determined, then N workflows of that size are chosen randomly for the ensemble from the set of synthetic workflows.

Uniform ensembles contain workflows with sizes that are uniformly distributed among the set of possible sizes. Each workflow is selected by first randomly choosing the size of the workflow and then randomly choosing a workflow of that size from the set of synthetic workflows.

Pareto ensembles contain a small number of larger workflows and a large number of smaller workflows. Their sizes are chosen according to a Pareto distribution. The distribution was modified so that the number of large workflows (of size ≥ 900) is increased by a small amount to produce a "heavy-tail". This causes Pareto ensembles to have a slightly larger number of large workflows, which reflects behavior commonly observed in many computational workloads. An example of the distribution of workflow sizes that occurs in a Pareto ensemble is shown in Fig. 6.

The number of workflows in an ensemble depends on the particular application, but we assume that ensemble sizes are on the order of between 10 and 100 workflows, typical of the real applications we have examined (see Section 1). For example, the number of geographical sites of interest to the users of the CyberShake application in the past has been on the order of 100. Second, smaller ensembles consisting of just a few workflows can be aggregated into a single workflow, so there is no need to treat them as an ensemble. Similarly, when the number of workflows grows and each workflow has a large number of tasks, either the deadline and budget constraints are low enough to prevent many of the workflows from running, or the problem of efficiently allocating them to the resources becomes similar to a bag-of-tasks problem, which is easier to solve efficiently.

5.3. Experimental parameters

In order to observe the interesting characteristics of the proposed algorithms, for each ensemble, we selected ranges for deadline and budget that cover a broad parameter space: from tight constraints, where only a small number of workflows can be completed, to more liberal constraints where all, or almost all, of the workflows can be completed. We computed constraint ranges based on the characteristics of each ensemble. The budget constraints are calculated by identifying the smallest budget required to execute one of the workflows in the ensemble (MinBudget), and the smallest budget required to execute all workflows in the ensemble (MaxBudget):

$$MinBudget = \min_{w \in e} Cost(w)$$

$$MaxBudget = \sum_{w \in e} Cost(w).$$

This range – [*MinBudget*, *MaxBudget*] – is then divided into equal intervals to determine the budgets to use in each experiment. Similarly, the deadline constraints are calculated by identifying the smallest amount of time required to execute a single workflow in the ensemble (MinDeadline), which is the length of the critical path for the workflow with the shortest critical path, and by identifying the smallest amount of time required to execute all workflows (MaxDeadline), which is the sum of the critical paths of all the workflows:

$$MinDeadline = \min_{w \in e} CriticalPath(w)$$

$$MaxDeadline = \sum_{w \in e} CriticalPath(w).$$

This range – [*MinDeadline*, *MaxDeadline*] – is then divided into equal intervals. By computing the budget and deadline constraints in this way we ensure that the experiments for each ensemble cover the most interesting area of the parameter space for the ensemble.

In all the experiments we assumed that the VMs have a price of $1 per VM-hour. This price was chosen to simplify interpretation of results and should not affect the relative performance of the different algorithms. In this study the heterogeneity of the infrastructure is not relevant since we assume that it is always possible to select a VM type that has the best price to performance ratio for a given application [43].

All the experiments were run with maximum autoscaling factor (υ_{max}) set to 1.0 for DPDS and WA-DPDS. After experimenting with DPDS and WA-DPDS we found that, due to the high parallelism of workflows used, the resource utilization remains high enough without adjusting the autoscaling rate. Based on experiments with the target applications, we set the SPSS α parameter for deadline distribution to be 0.7, which allocates slightly more time to levels with many tasks.

6. Discussion of results

6.1. Relative performance of algorithms

The goal of the first experiment is to characterize the relative performance of the proposed algorithms. This was done by simulating the algorithms on many different ensembles and comparing the scores computed using the performance metric based on exponential scoring defined in Section 3.3.

Fig. 7 shows the percentage of simulations for which each algorithm achieved the highest score for a given ensemble type. This experiment was conducted using all five applications, with all five types of ensembles. For each application and ensemble type, 10 random ensembles of 50 workflows each were created. Each ensemble was simulated with all three algorithms using 10 budgets and 10 deadlines (1000 simulations per application, ensemble type, and algorithm, or 75,000 in total). The best scores percentage is computed by counting the number of times that a given algorithm achieved the highest score and dividing by 1000. Note that it is possible for multiple algorithms to get the same high score (to tie), so the numbers do not necessarily add up to 100%. The sum is much higher than 100% in cases where the dynamic algorithms perform relatively well because DPDS and WA-DPDS, which are very similar algorithms, often get the same high score.

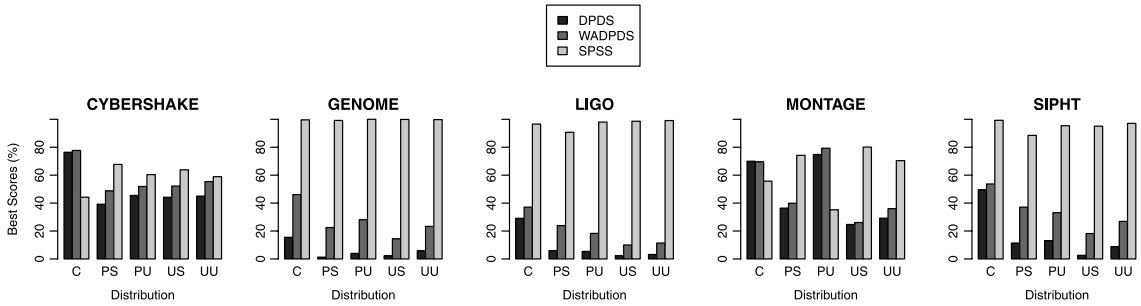


Fig. 7. Percentage of high scores achieved by each algorithm on different ensemble types for all five applications. C = Constant ensembles, PS = Pareto Sorted ensembles, PU = Pareto Unsorted ensembles, US = Uniform Sorted ensembles, and UU = Uniform Unsorted ensembles.

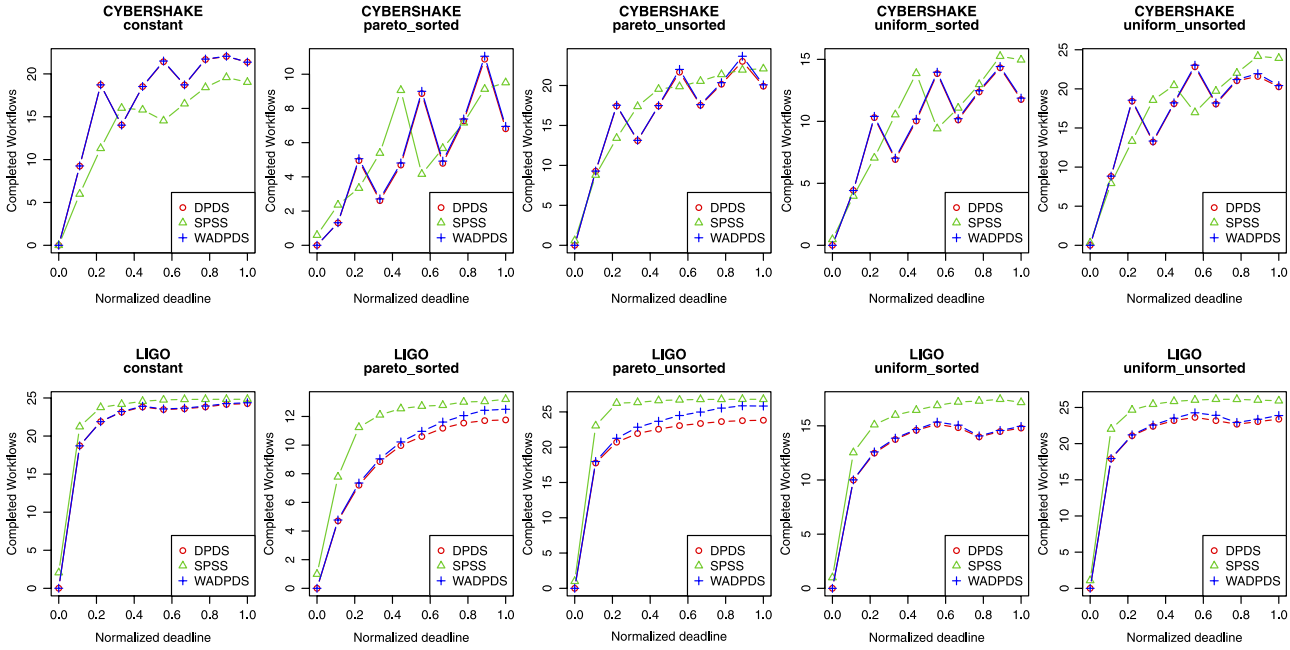


Fig. 8. Number of completed workflows from the ensemble depending on normalized deadline.

There are several interesting things to notice about Fig. 7. The first is that, in most cases, SPSS significantly outperforms both dynamic algorithms (DPDS and WA-DPDS). This is attributed to the fact that SPSS is able to make more intelligent scheduling and provisioning decisions because it has the opportunity to compare different options and choose the one that results in the best outcome. In comparison, the dynamic algorithms are online algorithms and are not able to project into the future to weigh the outcomes of their choices.

The second thing to notice is that, for constant ensembles, the dynamic algorithms perform significantly better relative to SPSS compared to other ensemble types. This is a result of the fact that, since all of the workflows are of approximately the same size and shape, the choice of which workflow to execute next has a smaller impact on the final result.

We can also see that the workflow-aware algorithms (WA-DPDS and SPSS) both perform better in most cases than the simple online algorithm that uses resource utilization alone to make provisioning decisions (DPDS). This suggests that there is a significant value in having information about the structure and estimated runtime of a workflow when making scheduling and provisioning decisions.

Finally, it is interesting that, for Montage and CyberShake, the relative performance of SPSS is significantly less than it is for other applications. We attribute this to the structure of Montage and CyberShake. The workflows for both applications are very wide relative to their height, and both have very short-running tasks, resulting in relatively short critical paths that makes them look more like bag-of-tasks applications, which are easier to execute than more structured applications. DPDS and WA-DPDS are able to pack more of the tasks into the available budget and deadline because there are (a) more choices for where to place the tasks, and (b) the different choices have a smaller impact on the algorithms' ability to execute the workflow within the constraints. In addition, the short critical paths put SPSS at a disadvantage. Because of the way SPSS assigns deadlines to individual tasks, it is prevented from starting workflows late, which prevents it from packing tasks into the idle VM slots at the end of the schedule.

6.2. Results vs. varying deadline and budget

In order to give more details about the performance of the algorithms, in this section we show how the results depend on the deadline and budget constraints. As the performance metric we use the number of completed workflows of the ensemble, since the exponential score is not suitable for showing trends. We use all deadline and budget parameters as defined in Section 5.3 and normalize them to the range from 0 to 1 to facilitate plotting. For each normalized deadline, we compute the mean number of completed workflows, averaged over all budgets and random seeds, so each point on the plot is based on 100 different simulation runs. In Figs. 8 and 9 we present the results of CyberShake and LIGO

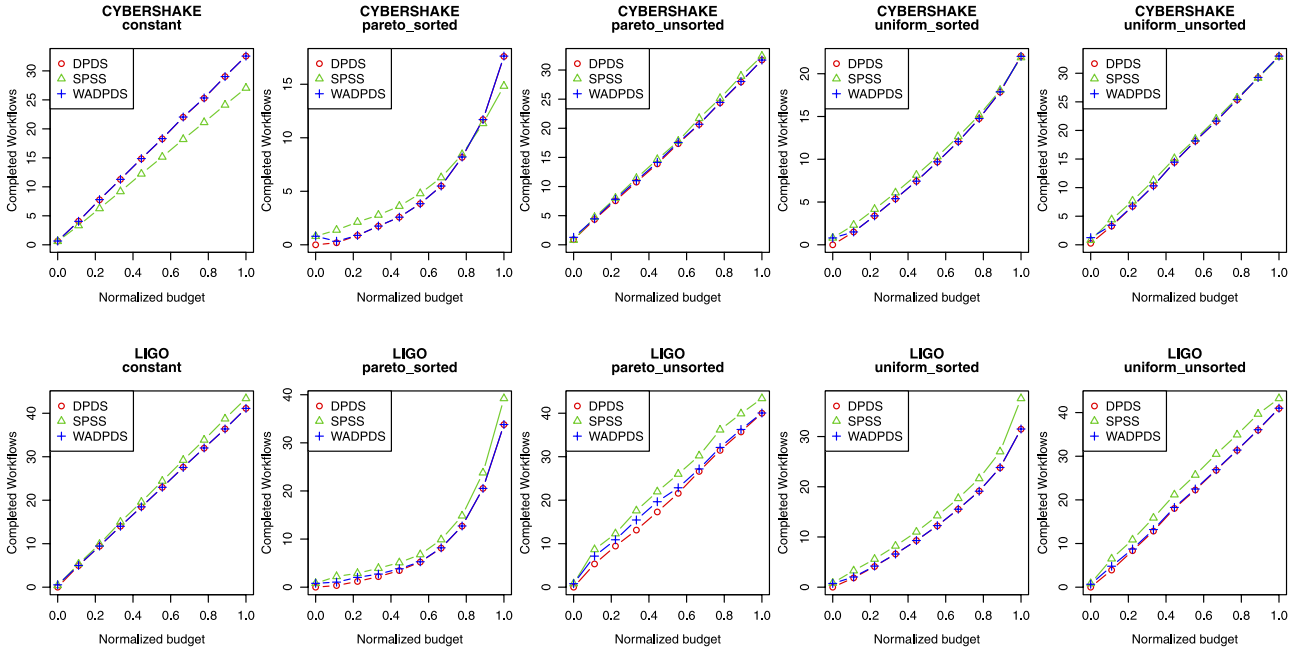


Fig. 9. Number of completed workflows from the ensemble depending on normalized budget.

applications, as representatives of fine-grained and coarse-grained workflows. Similar results were obtained for other applications.

Fig. 8 shows the results depending on normalized deadline. It can be observed, that with increasing deadline, the number of completed workflows grows steeply and then stabilizes. This means that for longer deadlines it is usually not possible to complete more workflows due to the budget constraint. For short deadlines, the algorithms need to allocate more VM instances in parallel, making it more difficult to utilize the resources efficiently, which results in lower numbers of completed workflows.

We can also observe that the algorithms are able to complete more workflows in the ensembles that have priorities assigned randomly (unsorted), in comparison to the ensembles where the priorities are assigned based on workflow size (sorted). This results from the fact that in the unsorted distributions the large workflows often have low priorities, so they can be rejected by the algorithms, thus making more space in the schedule for smaller high-priority workflows.

One interesting observation from Fig. 8 is that for CyberShake application the result curves have a sawtooth shape. This results from the combination of two effects. The first comes from the fact that dynamic algorithms perform better when the deadline is close to the full hour, due to the provisioning mechanism that tries to shut down unused VM instances before a full hour. The second one is that when the deadline increases, it may become possible to admit a larger workflow with higher priority instead of several smaller ones with lower priority. That is why the number of completed workflows may decrease, but the exponential score (utility function) based on priorities will be higher. These effects are visible for CyberShake application, for which the maximum deadline is 3 h, but they become less important for LIGO application, for which the maximum deadline is 20 h. See also the discussion on task granularity in Section 6.3.

Fig. 9 shows the results based on normalized budget, each point being the average over all deadline and random seed. What becomes apparent is that for constant and unsorted distributions the number of completed workflows is almost linear. This confirms that the budget constraint has the linear influence on the results, when the influence of the deadline is eliminated by averaging. On the other hand, for sorted distributions the functions are convex, increasing slowly for small budgets and steeper for larger budgets. This reflects the fact that the in sorted ensembles the largest workflows have the highest priority, so it requires a large increment of a small budget to fit the next workflow into the schedule.

The characteristics discussed in this section can be useful not only for analysis of our algorithms, but also for planning scientific experiments and understanding trade-offs between cost, deadline and the number of work complete [49].

6.3. Task granularity

In the experiments described in previous sections we noted that, for Montage and CyberShake, the short runtimes of their tasks made the dynamic algorithms perform better relative to SPSS. In order to test this theory we adjusted the granularity of the tasks in several Montage and CyberShake ensembles to see how this would affect the relative performance of the algorithms. The granularity adjustment was achieved by multiplying the runtime of each task by a fixed scaling factor.

Fig. 10 shows the relative performance of the algorithms as the scaling factor is increased from 1 to 16 for CyberShake and Montage applications. Each data point in the figure represents 500 simulations (5 ensembles with 10 budgets and 10 deadlines). The best scores percent was calculated the same way as it was for Fig. 7.

The figure shows that, as the scaling factor increases beyond 2, the relative performance of SPSS surpasses that of the dynamic algorithms. This result suggests that, in general, for fine-grained workflows the dynamic algorithms will produce better scores, and for coarse-grained workflows SPSS will produce better scores.

The granularity of the workflow tasks should be discussed in relation to the length of the cloud billing cycle. In this paper we assumed that the resources are billed by the hour, as is the case of VM instances on Amazon EC2. However, there are cloud providers that have higher billing frequency, e.g. every 5 min in the case of CloudSigma[1] or every 1 min in the case of Google Compute

[1] http://www.cloudsigma.com.

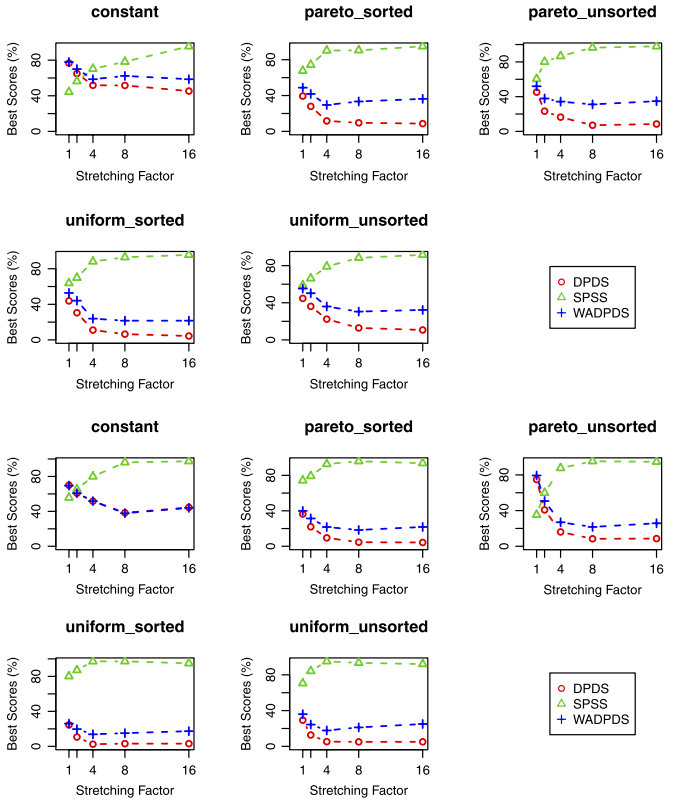


Fig. 10. Percentage of high scores achieved by each algorithm for CyberShake (top) and Montage (bottom) ensembles when task runtime is scaled by a constant factor. A scaling factor of x means that the runtime of tasks in each workflow was multiplied by x.

Engine[2] (after first 10 min). Our results indicate that when the task granularity is close to the length of the billing period, SPSS produces better results, while for tasks that are shorter than the billing period, dynamic algorithms perform better. This suggests that these granularity effects have to be taken into account when selecting a cloud provider, and also when using task clustering algorithms that group smaller tasks into larger groups [50].

6.4. Inaccurate task runtime estimates

Both of the workflow-aware algorithms rely on estimates of task runtimes to make better scheduling and provisioning decisions. Our experience suggests that such assumption is often reasonable, since we can obtain workflow performance characteristics from preliminary runs [44,47,51]. Some applications, like Periodograms [5] even include a performance model that estimates task runtimes and automatically annotates workflow description with these data. In practice, however, these estimates are often inaccurate. Given inaccurate estimates, the question is: How do errors in task runtime estimates impact the performance of our scheduling and provisioning algorithms? To examine this we introduced uniform errors in the task runtime and observed the behavior of the algorithms in terms of meeting the desired budget and deadline constraints.

In this experiment the actual runtime of each task is adjusted in the simulation by adding a random error to the estimated runtime of $\pm p\%$. Since the sampling is done uniformly, we expect to get just as many overestimates as underestimates in any given simulation. Our approach for generating errors is similar to the one used in [27], where the random error represents the Quality of Estimation metric.

Fig. 11 shows the results for estimate errors ranging from 0% to 50%. The figure summarizes the outcome of an extensive suite of 525,000 simulations (10 ensembles of 50 workflows $\times$ 5 applications $\times$ 5 distributions $\times$ 10 budgets $\times$ 10 deadlines $\times$ 7 error values $\times$ 3 algorithms). Box plots show the ratio of the ensemble cost to budget, and of ensemble makespan to deadline. Whiskers on the plots indicate maximum and minimum values. The ratio indicates whether the value (for example, the simulated cost) exceeded the constraint (the budget). Values greater than 1 indicate that the constraint was exceeded.

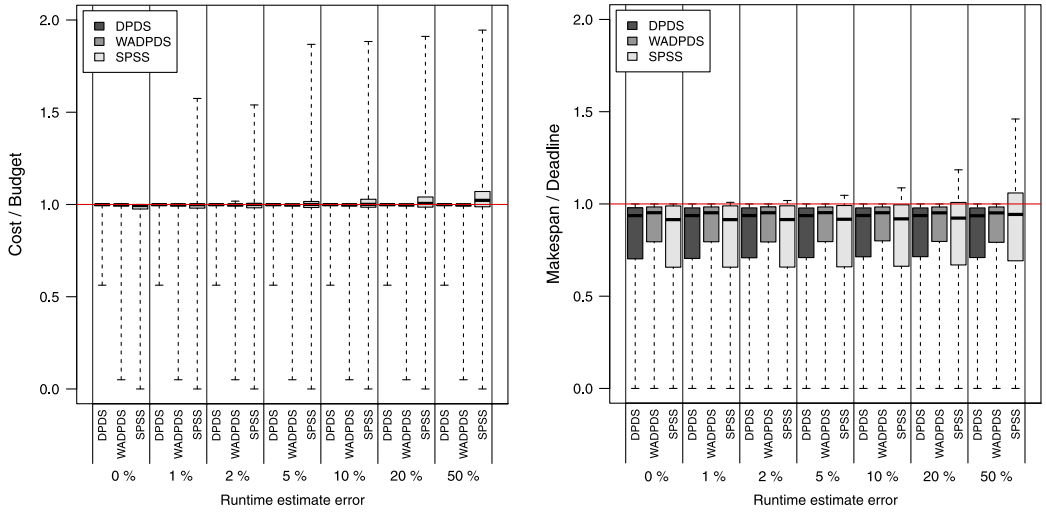
Fig. 11(a) shows the ratio of simulated cost to budget. This plot illustrates two important algorithm characteristics. First, the dynamic algorithms very rarely exceeded the budget, even with very large errors. This indicates that the dynamic algorithms are able to adapt to uncertainties at runtime to ensure that the constraints are not exceeded, even when the quality of information available to them is low. Second, unlike dynamic algorithms, the static algorithm frequently exceeded the budget constraint; by large amounts in some cases. This is a result of the fact that the static algorithm makes all of its decisions ahead of the execution and is not able to adapt to changing circumstances.

The SPSS plan describes what tasks to execute on which VMs, but not when to execute them. At runtime, the workflow engine is forced to spend extra money to extend the runtime of some VMs so that all of the tasks assigned to that VM can be completed. At the same time, other VMs can be terminated early because the tasks they were assigned finished earlier than expected. However, because of dependencies in the workflow, the latter case is less likely to happen, which causes gaps in the schedule. So the net result is that the overall cost is increased.

Fig. 11(b) shows the ratio of simulated makespan to budget. Interestingly, the deadline constraint is rarely exceeded, even in cases with very poor quality estimates. For the dynamic algorithms this is a result of the fact that they can adapt to poor estimates and stop submitting new tasks when the constraints are reached. Makespan is then computed as the finish time of the last fully completed workflow from the ensemble. For the static algorithm this is a result of the way that SPSS schedules workflows, and not of a particularly clever optimization. The SPSS algorithm tends to schedule workflows early, using up the budget long before the deadline is reached. This is a consequence of the deadline distribution function in SPSS, which prevents workflows from starting late. This is illustrated by the Gantt chart in Fig. 5(b), which shows how SPSS tends to pile up workflows at the beginning of the timeline. In comparison, the dynamic algorithms tend to spread out workflow starts over the entire duration as shown in Fig. 5(a). The consequence of this behavior is that SPSS plans tend to use up the budget but leave plenty of time before the deadline. As a result, when the runtime of the plan is increased by introducing errors, the SPSS plan has some room to expand without exceeding the deadline.

Fig. 12 shows relative performance as a percentage of the number of high scores achieved by each algorithm as the error increases. As in Figs. 9 and 8 we show results for CyberShake and LIGO as representatives of fine-grained and coarse-grained workflows. The figure shows that, for this experiment, the scores remain the same as the error increases. This is a result of the way the error is applied. Since the error is selected uniformly from $-p$ to $+p$ it is equally likely that the error will be positive as negative. Therefore, applying the error leaves the total runtime of the tasks in the workflow unchanged — the total number of CPU hours in each workflow remains nearly the same regardless of the error. The end result is that the dynamic algorithms are able to achieve the same score without exceeding the constraints by using a different

[2] https://cloud.google.com/pricing/compute-engine.



(a) Ratio of simulated cost to budget.

(b) Ratio of simulated makespan to deadline.

Fig. 11. Boxplots for budget and deadline ratios when runtime estimate error varies from ±0% to ±50% for all three algorithms. Values greater than 1 indicate that the budget/deadline constraint was exceeded.

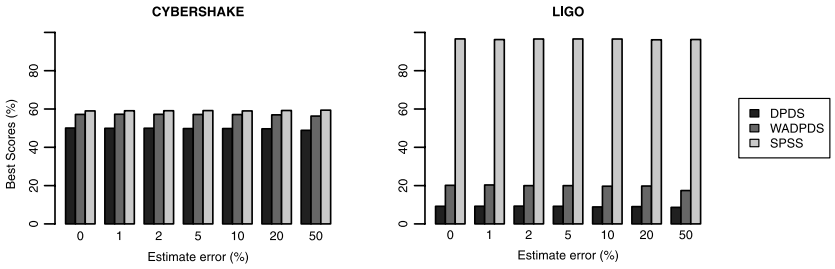


Fig. 12. Percentage of high scores achieved by each algorithm on CyberShake and LIGO when runtime estimate error varies from 0% to 50%.

schedule that finishes the same total amount of work using the same budget and deadline.

Note that the algorithms have *not* been changed to account for inaccurate runtime estimates in this experiment. It is likely that better performance could be achieved if the algorithms were given a hint as to the accuracy of the task runtimes. Investigating that optimization is left for future work. The goal here is to determine how well the current algorithms are able to stay within the constraints given inaccurate task runtime estimates.

It is possible that the static algorithm could be modified to add more breathing room in the schedule to account for situations where the runtime estimates may be inaccurate. Such a modification would result in slightly worse scores when the estimates are good, but would improve scores when the estimates are bad. Alternatively, the SPSS algorithm could be modified to generate a plan that specifies when to provision and deprovision each VM. In that case the workflow engine could be prevented from exceeding the budget and deadline constraints, but would prevent some workflows from finishing, which may significantly decrease the score. These two modifications are subjects for future work.

6.5. Provisioning delays

One important issue to consider when provisioning resources in the cloud is the amount of time between when a resource is requested, and when it actually becomes available to the application. Typically these provisioning delays are on the order of a few minutes, and are highly dependent upon the cloud architecture and/or the size of the VM image [52].

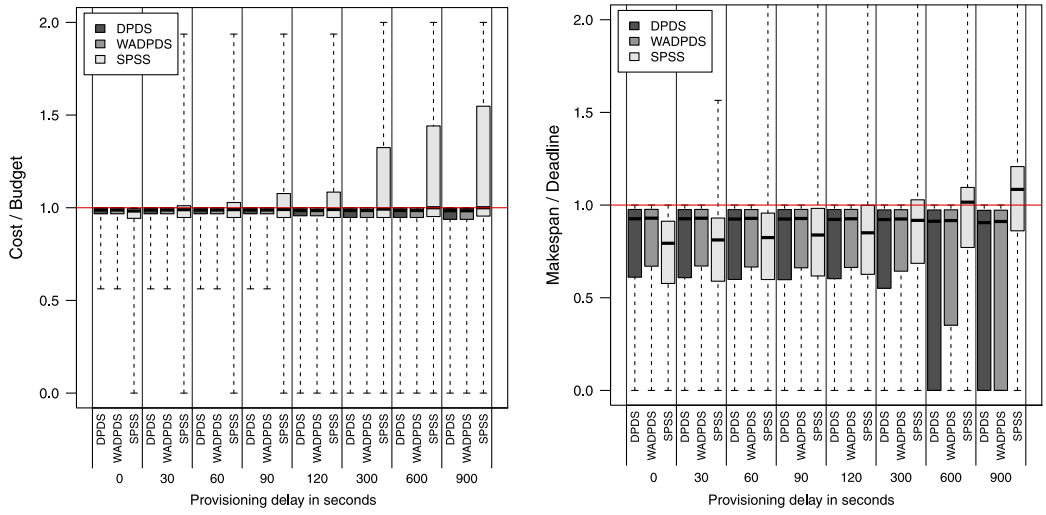
We assume that resources are billed from the minute that they are requested until they are terminated. As a result, provisioning delays have an impact on both the cost and makespan of an ensemble.

Fig. 13 shows the ratios of simulated values to constraints when the provisioning delay is increased from 0 s up to 15 min. The figure summarizes a suite of 105,000 simulations (10 ensembles of 50 workflows × 5 distributions × 10 budgets × 10 deadlines × 7 error values × 3 algorithms). To reduce the simulation time, only one application, Montage, was used in this experiment.

As in Fig. 11, the *y*-axis in each plot represents the ratio of the simulated value to the constraint value for the budget constraint or the deadline constraint.

The effect of provisioning delays on workflow performance is similar to that of inaccurate runtime estimates: when the delays are small, all algorithms are able to produce results within the constraints, but for larger delays, the dynamic algorithms are able to adapt to avoid exceeding the constraints while the static algorithm is not. In the case of delays of more than one minute, which is typical of what has been observed on academic clouds [53] such as Magellan [54] and FutureGrid [55], approximately half of the SPSS simulations exceeded the budget; in some cases by up to a factor of 2. In comparison, none of the simulations that used a dynamic algorithm exceeded the budget or the deadline constraint.

Fig. 14 shows the relative performance as a percentage of the number of high scores achieved by each algorithm as the provisioning delay increases. This figure shows that, as the delay increases, the relative performance of the static algorithm increases as well. This behavior is a result of the fact that the static algorithm is, in essence, cheating by using more time and money



(a) Ratio of simulated ensemble cost to budget.

(b) Ratio of simulated ensemble makespan to deadline.

Fig. 13. Boxplots for budget and deadline ratios when provisioning delay varies from 0 s to 15 min for all three algorithms. Values greater than 1 indicate that the budget/deadline constraint was exceeded.

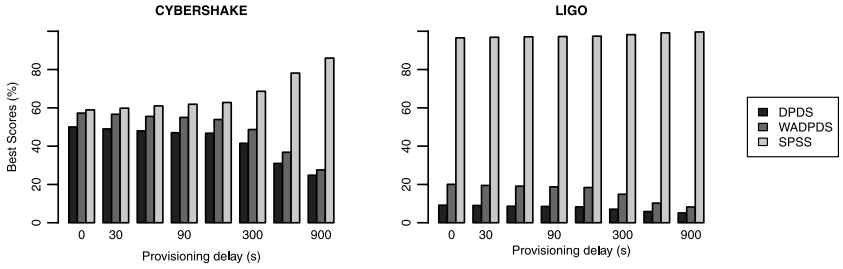


Fig. 14. Percentage of high scores achieved by each algorithm on CyberShake and LIGO ensembles when provisioning delay varies from 0 s to 15 min.

than the dynamic algorithms. In the previous section we showed how the relative performance of the algorithms remains the same when a uniform error is applied to the runtime constraints. In that case, the dynamic algorithms adapted to the error by rearranging the schedule to accomplish the same amount of work given the same deadline and budget. In this case, the dynamic algorithms adapt by performing less work (executing fewer workflows) to remain within the constraints while accounting for the delays. In both cases, the static algorithm completed the same amount of work, but did so by exceeding the constraints.

This experiment suggests that SPSS is too sensitive to provisioning delays in its current form to be of practical use in real systems. It is possible that modifying the SPSS algorithm to account for provisioning delays would improve its performance on this experiment. In fact, since provisioning operations are infrequent (because all the algorithms tend to provision resources for a long time), it is likely that the performance of SPSS could be improved significantly by simply adding an estimate of the provisioning delay to its scheduling function. Such an estimate may not have to be particularly accurate to get good results, and developing an estimate from historical data should be relatively simple. Testing this idea is left for future work.

6.6. Task failures

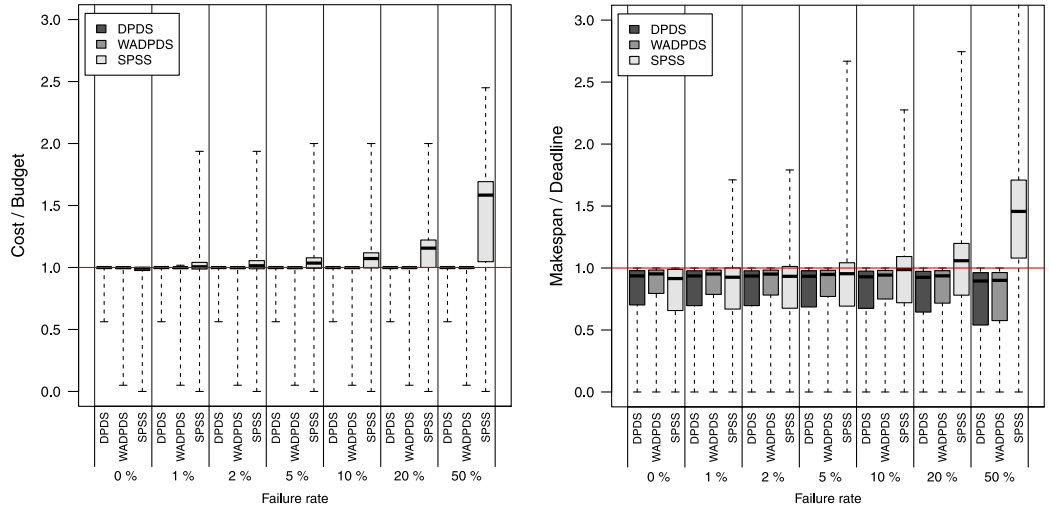
Running workflows consisting of large numbers of tasks on distributed systems often results in failures. The goal of the next experiment was to assess the behavior of the algorithms in the presence of task execution failures by introducing a failure model into the simulator. The model is characterized by a failure rate f that defines the probability that a task will fail. The failure time of the task is determined by randomly sampling a value between the task start time and finish time. If the task fails, it is reported to the workflow engine, which retries the task until it succeeds. The dynamic algorithms re-add the task to the priority queue (queue P in Algorithm 2) so that it can be resubmitted by the scheduler. The SPSS algorithm immediately resubmits the failed task to the same VM that was selected in the plan (to minimize the disruption to the overall plan).

Fig. 15 shows the ratios of simulated values to constraints when the failures are introduced. The figure summarizes a suite of 525,000 simulations (10 ensembles of 50 workflows × 5 applications × 5 distributions × 10 budgets × 10 deadlines × 7 failure rates × 3 algorithms). As in the previous experiments, these results show that high failure rates can degrade the performance of the static algorithm considerably, while the dynamic algorithms are able to adapt.

Fig. 16 shows the relative performance as a percentage of the number of high scores achieved by each algorithm as the failure rate increases. The results are similar to the ones of Fig. 14. Comparing Fig. 15 to Figs. 11 and 13 one may conclude that failures are worse than provisioning delays and runtime estimate errors, since their impact is larger. However, we consider higher failure rates as rare events that suggest a significant system malfunction or invalid selection of resources.

6.7. SPSS planning time

Because SPSS involves more complicated logic than the dynamic algorithms and makes its decisions before execution, it is



(a) Ratio of simulated ensemble cost to budget.

(b) Ratio of simulated ensemble makespan to deadline.

Fig. 15. Boxplots for budget and deadline ratios when failure rate varies from 0% to 50% for all three algorithms. Values greater than 1 indicate that the budget/deadline constraint was exceeded.

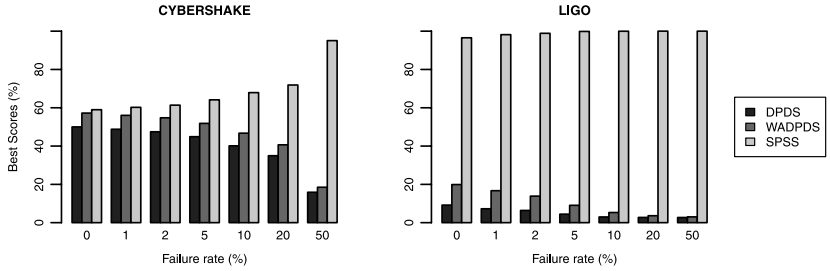


Fig. 16. Percentage of high scores achieved by each algorithm on CyberShake and LIGO ensembles when failure rate varies 0% to 50%.

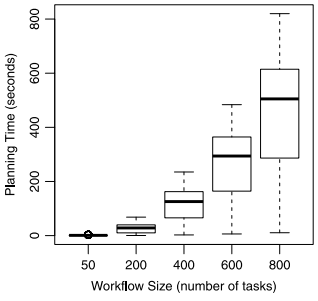


Fig. 17. Planning time of SPSS algorithm for ensembles of 100 workflows and different workflow sizes.

important to understand what impact planning time has on the overall execution time.

Fig. 17 shows the SPSS planning time for ensembles of 100 workflows with five different workflow sizes: 50, 200, 400, 600, and 800 tasks. The ensembles were generated using a constant distribution equal to the workflow size desired. Two different applications were used: SIPHT and CyberShake. Each box summarizes the results of 2000 simulations (2 applications $\times$ 10 ensembles $\times$ 10 budgets $\times$ 10 deadlines).

Fig. 17 shows that, for small workflows, the SPSS planning time is reasonable, taking on the order of tens of seconds to a few minutes. For larger ensembles of large workflows, however, the SPSS planning time can easily reach 10 min. Considering that the largest workflows used in this experiment are still relatively small (maximum of 800 tasks), and that real workflows are often much larger (workflows with tens of thousands of tasks are common, and even workflows with millions of tasks are possible), it is unlikely that SPSS will be practical for ensembles of very large workflows.

SPSS considers scheduling each task on the cheapest available slot, which involves scanning all of the available slots on all of the VMs. Since the number of available slots, in the worst case, is proportional to the number of tasks scheduled (because scheduling a task splits an existing slot into at most two slots: one before the task, and one after), the complexity of SPSS is $O(n^2)$, where n is the number of tasks in the ensemble. In comparison, the dynamic algorithms all have a more scalable complexity of $O(n)$. DPDS only examines the tasks in the workflow once when they are scheduled, and WA-DPDS does it twice: once in the admission algorithm, and once when they are scheduled. This makes the dynamic algorithms a better fit for larger workflows and ensembles even though in some cases they may not produce as good results as SPSS.

It may be possible to optimize SPSS to reduce its runtime by, for example, clustering the workflow to increase task granularity, which would decrease the ratio of planning time to ensemble makespan. It may also be possible to reduce the complexity of SPSS by employing more sophisticated data structures to store the available slots. Investigating these topics is left for future work.

7. Conclusions and future work

In this paper we addressed the interesting and important new problem of scheduling and resource provisioning for scientific workflow ensembles on IaaS clouds. The goal of this work is to maximize the number of user-prioritized workflows that can be completed given budget and deadline constraints.

This problem differs from previous work on grid and utility grid scheduling in that cloud infrastructures provide more control over provisioning, so that the number of resources can be adjusted according to the requirements of the application. Therefore the problem space becomes larger; it requires not only an efficient mapping of tasks to available resources, but also the selection of the best resource provisioning plan.

Formulating the problem as a maximization of the number of prioritized workflows completed from the ensemble is also novel and requires workflows to be admitted or rejected based on their estimated resource demands. We believe that this bi-constrained problem is highly relevant because such constraints are commonly imposed on many real-world projects. The approach is also directly applicable to grid environments that provide resource reservations and charge service units for resource use.

We developed three algorithms to solve this problem: two dynamic algorithms, DPDS and WA-DPDS, and one static algorithm, SPSS. The algorithms were evaluated via simulation on ensembles of synthetic workflows, which were generated based on statistics from real scientific applications.

The results of our simulation studies indicate that the two algorithms that take into account the structure of the workflow and task runtime estimates (WA-DPDS and SPSS) yield better results than the simple priority-based scheduling strategy (DPDS), which makes provisioning decisions based purely on resource utilization. This underscores the importance of viewing workflow ensembles as a whole rather than as individual tasks or individual workflows.

In cases where there are no provisioning delays, task runtime estimates are good, and failures are rare, we found that SPSS performs significantly better than both dynamic algorithms. However, when conditions are less than perfect, the static plans produced by SPSS are disrupted and it frequently exceeds the budget and deadline constraints. In comparison, the dynamic algorithms are able to adapt to a wide variety of conditions, and rarely exceed the constraints even with long delays, poor estimates, and high failure rates. However, this comes at the cost of not being able to complete as many workflows as SPSS in ideal conditions.

In addition, we found that SPSS tends to perform better on coarse-grained workflows than the dynamic algorithms. For wide, fine-grained workflows, such as CyberShake and Montage, however, the dynamic algorithms frequently produce performance that is as good or better than SPSS, because they are better able to pack the smaller tasks onto idle VMs close to the deadline than SPSS, which distributes deadlines to tasks in a way that prevents them from starting late.

For very large workflows and ensembles, we found that the planning time of the SPSS algorithm is prohibitive. SPSS often took 10 min or more to plan ensembles of 100 workflows with 800 tasks each. This suggests that ensembles of workflows with tens of thousands of tasks, which are commonly encountered in real workflow applications, would take many hours to plan using SPSS. This makes the dynamic algorithms much more attractive for large scale problems.

This study suggests several areas for future work. Our current approach models data access as part of the task execution time and does not explicitly consider data storage and transfer costs. In the future we plan to extend the application and infrastructure model to include the various data storage options available on clouds. A previous experimental study [51] suggests that the data demands of scientific workflows have a large impact on not only the execution time, but also on the cost of workflows in commercial clouds. In dynamic algorithms, it will be interesting to extend our utilization-based autoscaling with more advanced workflow-aware strategies based on feedback control. We also plan to investigate heterogeneous environments that include multiple VM types and cloud providers, including private and community clouds, which will make the problem even more complex and challenging.

The results of this study can be applied to develop tools that assist researchers in planning their large-scale computational experiments. The estimates of cost, runtime, and number of workflows completed that can be obtained from both the static algorithms and from the simulation runs, constitute valuable hints for planning ensembles and evaluating the associated trade-offs.

Acknowledgments

This work was supported by the U.S. Department of Energy Office of Science under award number ER26110, by the National Science Foundation ACI SI2-SSI program award number 1148515, by the AGH grant 11.11.230.124 and by the EU FP7 project VPH-Share (269978).

References

[1] S. Callaghan, P. Maechling, P. Small, K. Milner, G. Juve, T. Jordan, E. Deelman, G. Mehta, K. Vahi, D. Gunter, K. Beattie, C.X. Brooks, Metrics for heterogeneous scientific workflows: a case study of an earthquake science application, Int. J. High Perform. Comput. Appl. 25 (3) (2011) 274–285.
[2] CyberShake Study 13.4. URL http://scec.usc.edu/scecpedia/CyberShake_Study_13.4.
[3] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the montage example, in: 2008 ACM/IEEE Conference on Supercomputing, SC 08, 2008.
[4] Montage galactic plane. URL http://pegasus.isi.edu/applications/galactic-plane.
[5] J. Vöckler, G. Juve, E. Deelman, M. Rynge, G.B. Berriman, Experiences using cloud computing for a scientific workflow application, in: 2nd Workshop on Scientific Cloud Computing, ScienceCloud'11, 2011.
[6] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 cloud computing services for scientific computing, in: O. Akan, et al. (Eds.), Cloud Computing, in: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 34, Springer, Berlin, Heidelberg, 2010, pp. 115–131.
[7] K. Keahey, M. Tsugawa, A. Matsunaga, J. Fortes, Sky computing, IEEE Internet Comput. 13 (5) (2009) 43–51.
[8] D. Durkee, Why cloud computing will never be free, Commun. ACM 53 (5) (2010) 62–69.
[9] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (1) (2011) 23–50.
[10] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC'12, IEEE Computer Society Press, 2012, URL http://portal.acm.org/citation.cfm?id=2389026.
[11] Amazon auto scaling. URL http://aws.amazon.com/autoscaling.
[12] RightScale. URL http://www.rightscale.com.
[13] P. Marshall, K. Keahey, T. Freeman, Elastic site: using clouds to elastically extend site resources, in: 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2010, 2010.
[14] H. Kim, Y. el-Khamra, I. Rodero, S. Jha, M. Parashar, Autonomic management of application workflows on hybrid computing infrastructure, Sci. Program. 19 (2011) 75–89.
[15] J. Chen, C. Wang, B.B. Zhou, L. Sun, Y.C. Lee, A.Y. Zomaya, Tradeoffs between profit and customer satisfaction for service provisioning in the cloud, in: Proceedings of the 20th International Symposium on High Performance Distributed Computing, HPDC'11, ACM, New York, NY, USA, 2011, pp. 229–238.
[16] J. Yu, R. Buyya, C. Tham, Cost-based scheduling of scientific workflow application on utility grids, in: First International Conference on e-Science and Grid Computing, 2005.
[17] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169. http://dx.doi.org/10.1016/j.future.2012.05.004. URL http://www.sciencedirect.com/science/article/pii/S0167739X12001008.
[18] Y.C. Lee, A.Y. Zomaya, Stretch out and compact: workflow scheduling with resource abundance, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid, IEEE, 2013, pp. 219–226. http://dx.doi.org/10.1109/ccgrid.2013.55.
[19] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: 2011 ACM/IEEE Conference on Supercomputing, SC'11, 2011.

[20] H. Arabnejad, J. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694. http://dx.doi.org/10.1109/TPDS.2013.57.
[21] M. Wieczorek, A. Hoheisel, R. Prodan, Towards a general model of the multi-criteria workflow scheduling on the grid, Future Gener. Comput. Syst. 25 (3) (2009) 237–256.
[22] R. Prodan, M. Wieczorek, Bi-criteria scheduling of scientific grid workflows, IEEE Trans. Autom. Sci. Engrg. 7 (2) (2010) 364–376.
[23] J.J. Dongarra, E. Jeannot, E. Saule, Z. Shi, Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems, in: 19th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'07, 2007.
[24] J.J. Durillo, V. Nae, R. Prodan, Multi-objective workflow scheduling: an analysis of the energy efficiency and makespan tradeoff, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid, IEEE, 2013, pp. 203–210. http://dx.doi.org/10.1109/ccgrid.2013.62.
[25] J.J. Durillo, H.M. Fard, R. Prodan, Moheft: a multi-objective list-based method for workflow scheduling, in: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, December 3–6, 2012, pp. 185–192.
[26] R. Sakellariou, H. Zhao, E. Tsiakkouri, M.D. Dikaiakos, Scheduling workflows with budget constraints, in: Integrated Research in GRID Computing, in: COREGrid Series, Springer-Verlag, 2007.
[27] W. Zheng, R. Sakellariou, Stochastic DAG scheduling using a Monte Carlo approach, J. Parallel Distrib. Comput. 73 (12) (2013) 1673–1689. http://dx.doi.org/10.1016/j.jpdc.2013.07.019. Heterogeneity in parallel and distributed computing.
URL http://www.sciencedirect.com/science/article/pii/S0743731513001573.
[28] H. Zhao, R. Sakellariou, Scheduling multiple dags onto heterogeneous systems, in: 20th International Parallel and Distributed Processing Symposium, 2006, IPDPS, 2006, p. 14. http://dx.doi.org/10.1109/IPDPS.2006.1639387.
[29] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J. González-García, T. Röblitz, J. Ramírez-Alcaraz, Multiple workflow scheduling strategies with user run time estimates on a grid, J. Grid Comput. 10 (2) (2012) 325–346. http://dx.doi.org/10.1007/s10723-012-9215-6.
[30] R. Duan, R. Prodan, X. Li, A sequential cooperative game theoretic approach to Storage-Aware scheduling of multiple Large-Scale workflow applications in grids, in: 2012 ACM/IEEE 13th International Conference on Grid Computing (GRID), IEEE, 2012, pp. 31–39. http://dx.doi.org/10.1109/Grid.2012.14.
[31] R. Duan, R. Prodan, X. Li, Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds, IEEE Trans. Cloud Comput. 2 (1) (2014) 29–42. http://dx.doi.org/10.1109/TCC.2014.2303077.
[32] R. Tolosana-Calasanz, J.A. Banares, C. Pham, O.F. Rana, Enforcing qos in scientific workflow systems enacted over cloud infrastructures, J. Comput. System Sci. 78 (5) (2012) 1300–1315. http://dx.doi.org/10.1016/j.jcss.2011.12.015.
URL http://www.sciencedirect.com/science/article/pii/S0022000011001607.
[33] W. Zheng, R. Sakellariou, Budget-deadline constrained workflow planning for admission control, J. Grid Comput. 11 (4) (2013) 633–651. http://dx.doi.org/10.1007/s10723-013-9257-4.
[34] A.K.M.K.A. Talukder, M. Kirley, R. Buyya, Multiobjective differential evolution for scheduling workflow applications on global grids, Concurr. Comput. Pract. Exp. 21 (13) (2009) 1742–1756.
[35] S. Pandey, L. Wu, S.M. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in: International Conference on Advanced Information Networking and Applications, 2010.
[36] J. Durillo, R. Prodan, Multi-objective workflow scheduling in Amazon EC2, Clust. Comput. 17 (2) (2014) 169–189. http://dx.doi.org/10.1007/s10586-013-0325-0.
[37] T.A.L. Genez, L.F. Bittencourt, E.R.M. Madeira, Using time discretization to schedule scientific workflows in multiple cloud providers, in: 2013 IEEE Sixth International Conference on Cloud Computing, IEEE, 2013, pp. 123–130. http://dx.doi.org/10.1109/CLOUD.2013.141.
URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6676686.
[38] R. Van den Bossche, K. Vanmechelen, J. Broeckhove, Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads, in: 2010 IEEE 3rd International Conference on Cloud Computing, IEEE, 2010, pp. 228–235. http://dx.doi.org/10.1109/CLOUD.2010.58. URL
http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5557990.
[39] R.V. den Bossche, K. Vanmechelen, J. Broeckhove, Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds, Future Gener. Comput. Syst. 29 (4) (2013) 973–985.
http://dx.doi.org/10.1016/j.future.2012.12.012.
URL http://www.sciencedirect.com/science/article/pii/S0167739X12002324.
[40] L.F. Bittencourt, E.R.M. Madeira, Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds, J. Internet Serv. Appl. 2 (3) (2011) 207–227.
[41] M. Malawski, K. Figiela, M. Bubak, E. Deelman, J. Nabrzyski, Cost optimization of execution of multi-level deadline-constrained scientific workflows on clouds, in: Parallel Processing and Applied Mathematics — 10th International Conference, PPAM 2013, Warsaw, Poland, September 8–11, in: Lecture Notes in Computer Science, vol. 8384, Springer, 2014, pp. 251–260. Revised selected papers, Part I.
[42] I. Sfiligoi, glideinWMS—a generic pilot-based workload management system, J. Phys. Conf. Ser. 119 (6) (2008) 062044. URL http://stacks.iop.org/1742-6596/119/i=6/a=062044.
[43] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, P. Maechling, Scientific workflow applications on amazon EC2, in: 2009 5th IEEE International Conference on E-Science Workshops, 2009.
[44] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, K. Vahi, Characterization of scientific workflows, in: 3rd Workshop on Workflows in Support of Large Scale Science, WORKS 08, 2008.
[45] Workflow generator. URL https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.
[46] Cloud workflow simulator.
URL https://github.com/malawski/cloudworkflowsimulator.
[47] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, D.S. Katz, Pegasus: a framework for mapping complex scientific workflows onto distributed systems, Sci. Program. 13 (3) (2005) 219–237.
[48] R.F. da Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling research in distributed scientific workflows, in: 10th IEEE International Conference on e-Science, eScience 2014, 2014.
[49] M. Malawski, K. Figiela, J. Nabrzyski, Cost minimization for computational applications on hybrid cloud infrastructures, Future Gener. Comput. Syst. http://dx.doi.org/10.1016/j.future.2013.01.004.
URL http://dx.doi.org/10.1016/j.future.2013.01.004.
[50] W. Chen, E. Deelman, Fault tolerant clustering in scientific workflows, in: Eighth IEEE World Congress on Services, SERVICES 2012, Honolulu, HI, USA, June 24–29, 2012, IEEE, 2012, pp. 9–16.
[51] G. Juve, E. Deelman, K. Vahi, G. Mehta, B.P. Berman, B. Berriman, P. Maechling, Data sharing options for scientific workflows on amazon EC2, in: 2010 ACM/IEEE conference on Supercomputing, SC10, 2010.
[52] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems, UCSB Computer Science Technical Report 2008-10, 2008.
[53] G. Juve, E. Deelman, Automating application deployment in infrastructure clouds, in: 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011, 2011.
[54] National Energy Research Scientific Computing Center, NERSC, Magellan. URL http://magellan.nersc.gov.
[55] FutureGrid. URL http://futuregrid.org/.

Maciej Malawski holds Ph.D. in Computer Science, M.Sc. in Computer Science and in Physics. He is an assistant professor and a researcher at the Department of Computer Science AGH and at ACC Cyfronet AGH, Krakow, Poland. In 2011–13 he was a postdoc and a visiting faculty at Center for Research Computing, University of Notre Dame, USA.

He is coauthor of over 50 international publications including journal and conference papers, and book chapters. He participated in EU ICT Cross-Grid, ViroLab, CoreGrid and VPH-Share projects. His scientific interests include parallel computing, grid and cloud systems, distributed service- and component-based computing, and scientific applications.

Gideon Juve is a Computer Scientist at the USC Information Sciences Institute. He earned B.S., M.S., and Ph.D. degrees in Computer Science from the University of Southern California. His research focuses on resource management techniques for scientific workflows in clusters, grids, and clouds.

Ewa Deelman is a Research Associate Professor at the USC Computer Science Department and a Project Leader at the USC Information Sciences Institute. Dr. Deelman's research interests include the design and exploration of collaborative, distributed scientific environments, with particular emphasis on workflow management as well as the management of large amounts of data and metadata. At ISI, Dr. Deelman is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale applications running in distributed environments. Pegasus is being used today in a number of scientific disciplines, enabling researches to formulate complex computations in a declarative way. Over the years, Dr. Deelman worked with a number of application

domains including astronomy, bioinformatics, earthquake science, gravitational-wave physics, and others.

As part of these collaborations, new advances in computer science and in the domain sciences were made. For example, the data intensive workflows in LIGO (gravitational-wave physics) motivated new workflow analysis algorithms that minimize workflow data footprint during execution. On the other hand, improvements in the scalability of workflows enabled SCEC scientists (earthquake science) to develop new physics-based seismic hazard maps of Southern California. In 2007, Dr. Deelman edited a book on workflow research: "Workflows in e-Science: Scientific Workflows for Grids", published by Springer. She is also the founder of the annual Workshop on Workflows in Support of Large-Scale Science, which is held in conjunction with the Super Computing conference. In 1997 Dr. Deelman received her Ph.D. in Computer Science from the Rensselaer Polytechnic Institute.

Jarek Nabrzyski received his Ph.D. and M.Sc. in Computer Science and Engineering from Poznan University of Technology, Poland. Currently he is director of the Center for Research Computing and concurrent associate professor at the Department of Computer Science and Engineering at the University of Notre Dame. He co-authored over 50 publications in international journals and conferences, one co-edited book and several conference proceedings. His scientific interests include operations research, resource management in distributed and cloud computing systems, scientific applications and multi-criteria decision support systems. Nabrzyski manages a multidisciplinary team of 35 research staff consisting of computational scientists, HPC engineers and research programmers.