A Pareto-based approach for CPU provisioning of scientific workflows on clouds

DOI:
10.1016/j.future.2018.12.004

Document Version
Accepted author manuscript

Citation for published version (APA):
Pietri, I., & Sakellariou, R. (2019). A Pareto-based approach for CPU provisioning of scientific workflows on clouds. *Future Generation Computer Systems*, *94*, 479 - 487. https://doi.org/10.1016/j.future.2018.12.004

Published in:
Future Generation Computer Systems

A Pareto-Based Approach for CPU Provisioning of Scientific Workflows on Clouds

Ilia Pietri[a], Rizos Sakellariou[b]

[a]*Intracom SA Telecom Solutions, Greece*
[b]*School of Computer Science, University of Manchester, UK*

Abstract

As of recently, cloud providers have started offering CPU resources that can be selected from a wide range of different CPU frequencies. CPU resources at higher frequencies have a higher price than CPU resources at lower frequencies that are available at a lower price. When executing applications, such as large scientific workflow applications, multiple CPU resources are required. In this case, this new pricing scheme allows users to choose from a large number of possible CPU configurations that may include relatively fast and relatively slow CPUs. However, such an option raises the problem of how to select appropriate CPU frequency configurations that strike a good balance between cost and execution time performance. As the search space is large with a wide range of choices that have different trade-offs, the problem becomes how to choose Pareto-efficient solutions with respect to execution time and (monetary) cost to use the (CPU) resources. This paper proposes an algorithm to efficiently explore alternative CPU configurations for a given number of resources and identify Pareto-efficient solutions for cost and execution time trade-offs. The algorithm is evaluated through simulation using three different pricing models to charge for CPU provisioning according to the allocated CPU frequency and four widely used scientific workflow applications.

Keywords: Cloud computing, Pareto-efficient scheduling, scientific workflows, resource provisioning, CPU frequency

1. Introduction

Several cloud providers now allow users to select computational capacity from a large number of options that charge CPU provisioning based on the selected CPU frequency of each resource. For example, providers such as ElasticHosts [1]

and CloudSigma [2] offer a large number of different CPU frequency and price combinations (ranging from tens to hundreds per core). More powerful and faster resources (which operate at higher frequencies) are charged at a higher price, which means that even a small increase of some MHz in the CPU frequency may incur a small but still higher monetary cost. With the emergence of this new pricing scheme, users can achieve varying (monetary) cost and execution time trade-offs for the execution of large applications such as scientific workflows [3, 4], which typically need to run on multiple cloud resources (multiple cores and/or servers). By simply opting for different CPU frequencies in the provisioned resources, applications can make use of configurations that contain both relatively cheap but less powerful resources (which operate at low CPU frequencies) and expensive and more powerful resources (which operate at high frequencies).

It should be noted that even if users provision less powerful resources (i.e., resources running at a low CPU frequency), which are offered at a lower price per unit of time, selecting low frequencies may not necessarily lead to an overall low cost for the user. This is because CPU frequency reduction may significantly affect overall execution time of the application. For example, a CPU-bound application can be very sensitive to CPU frequency changes, performing significantly worse at low frequencies. This means that the benefits of a lower cost per unit of time (because lower CPU frequencies are used) may be outweighed by the slowdown in overall application execution time, which may lead to an overall higher cost for the user. In other cases, lower frequencies may lead to an overall longer execution time for an application but at a much lower overall cost: some reasonable performance degradation may be tolerated in favor of high monetary savings. These remarks lead to the observation that underpins the work in this paper: different CPU frequency configurations for a given number of provisioned resources may result in different values in terms of cost and execution time. From the range of these different values, the pairs of values for cost and execution time that are not dominated by other pairs (i.e., pairs of values that are not inferior in terms of both cost and execution time compared to any other pair) form a *Pareto set* of cost-performance efficient configurations. Clearly, finding the Pareto set from all different configurations is a challenge, as the Pareto set may vary depending on: (i) the application; (ii) the number and type of resources used; and (iii) the pricing policy associated with the use of these resources.

This multi-objective optimization problem of identifying CPU frequencies for cost-performance efficient configurations becomes particularly challenging when the applications of interest consist of interdependent tasks, as it is the case with scientific workflow applications [3, 4]. In such applications, typically represented

by Directed Acyclic Graphs (DAGs), gaps of idle time may occur between the execution of two different tasks due to data or flow dependencies. In addition, as the different interdependent tasks of a scientific workflow may have different characteristics, some tasks may be predominantly CPU-bound (that benefit from running at high frequency CPUs) but some other tasks may be predominantly I/O-bound (that are not significantly affected if they run at low frequency CPUs). Both the presence of idle time and tasks of different characteristics suggest that resources running at different CPU frequencies may indeed provide cost-performance efficient configurations to run scientific workflows on the cloud.

This paper proposes an algorithm, the Pareto-efficient Stepwise Frequency Selection Algorithm (PSFS), to provision CPU frequencies for a predefined number of CPU resources that aims to find cost-efficient, Pareto-optimal solutions for different execution time and cost trade-offs. As the search space may potentially be very large, the idea is to start with a small number of CPU frequency configurations to develop an initial Pareto set for the selected number of resources and then to gradually improve this initial Pareto set by iteratively reducing the CPU frequency per resource when solutions with higher cost savings can be achieved. The proposed approach builds upon previous work in [5], which aimed to find the cheapest CPU configurations to complete execution within a deadline. In this paper, the proposed approach is extended to return a Pareto set that meets different cost-performance trade-offs, something that allows the user to select CPU frequency configurations according to different requirements.

The rest of the paper is organized as follows. Section 2 summarizes related work. A description of the problem and the assumptions made follow in Section 3. Section 4 presents the provisioning algorithm proposed, while Section 5 describes the experimental setup and provides a range of experimental results. Finally, Section 6 concludes the paper.

2. Related Work

Lots of work addresses the problem of optimizing scientific workflow execution in distributed environments, such as grids or clouds, considering different optimization objectives, such as execution time or monetary cost [6, 7, 8, 9, 10, 11]. In terms of optimizing execution time and finding an appropriate schedule of workflow tasks onto parallel resources, the problem is equivalent to the well-known problem of scheduling Directed Acyclic Graphs (DAGs), which has a long history and a significant amount of literature exists [6, 12, 13, 14]. Among the various DAG scheduling heuristics, HEFT [6] is a well-known heuristic that has

been widely applied to schedule scientific workflows. Its aim is to minimize overall workflow execution time following an approach that prioritizes workflow tasks and then uses this prioritization to assign each workflow task to the resource where it is estimated to have the earliest finish time.

Focusing on scientific workflows, there is a considerable body of work that formulates the problem of optimizing scientific workflow execution as a constrained bi-objective problem [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25], where commonly considered objectives include application deadlines or budget constraints. In other work, bi-objective optimization techniques combine conflicting objectives into some sort of a weighted function to generate a single execution plan that achieves a good trade-off between the different objectives [9, 10, 26]. Of particular interest is the work in [9, 10] which points out to the likelihood that fine-grained pricing models of resources may have to be considered. In [9], HEFT is extended to incorporate the monetary cost when making decisions to find slots with a balance between execution time and cost assuming a fine-grained pricing model in which cloud resources with a higher processing capability are offered at a higher price. The different combinations of VM capacities and prices are also considered in [10] to select proper configurations. Their algorithm makes an initial allocation of the workflow tasks favoring cheaper VMs and then modifies the execution schedule by reassigning non-critical tasks (tasks that are not in the critical path) to less expensive VMs to improve overall cost when task runtimes can be stretched without extending overall workflow execution time.

Pareto-based approaches that compute a set of solutions with different cost and execution time trade-offs, as in [27, 28, 29], are more directly related to the approach proposed in this paper. In [27], two single-objective approaches for cost and execution time minimization on clouds are combined into a Pareto-based approach to select non-dominated solutions. MOHEFT [28] extends HEFT by keeping a set of non-dominated solutions at each step rather than a single allocation. The problem addressed in our paper is similar to the aforementioned approaches but differs in that our objective is to select CPU frequency combinations for a given number of resources, assuming the user has already selected the number of CPU resources to provision based on some performance estimation model as, for example, the model described in [30]. It is noted that finding a Pareto set may be a computationally demanding exercise; as also noted in a useful early classification of multi-criteria workflow scheduling problems [31] Pareto sets may be large. However, as several Cloud providers nowadays offer users a large number of possible configuration choices and fine-grained pricing of their resources, it becomes essential to investigate such Pareto-based approaches to allow users to

make reasonably efficient and informed choices and avoid sub-optimal provisioning solutions.

Finally, a considerable body of work in service-oriented applications (SoA) deals with the problem of workflow scheduling as a service composition problem, where different service providers can be combined into a composite service. The challenge of selecting a proper set of services which meet the desired QoS level of users is addressed to develop self-adaptive mechanisms that can sustain the different requests of a single or different users [32, 33, 34, 35]. However, the selection of suitable service providers is mainly based on the QoS constraints of individual tasks (or services). In contrast, our work mainly targets specific QoS requirements of the whole workflow that interact with each other, also taking into account task constraints and their data dependencies which need to be respected during execution.

3. Preliminaries

3.1. Problem Description

In the problem considered, the user is interested in executing a scientific workflow on a given number of CPU resources that will have to be obtained from a cloud provider. Each CPU resource of the cloud provider can be offered at a range of CPU frequencies. The provider will charge for the use of each CPU resource on the basis of the selected CPU frequency, with high frequencies costing more than low frequencies. As a result, different choices made by the user are expected to vary both in terms of the overall (monetary) cost and the application execution time. Choosing the highest frequency for all CPUs may result in fastest execution time but at a high cost. Other choices may result in slower execution time but may also cost less. Clearly, many choices are expected to return suboptimal results, in the sense that they are dominated by at least one choice which delivers both better cost and better execution time. The results of some choices may lie on the Pareto front, meaning that such a choice strikes a good balance between cost and execution time. Results on the Pareto front are non-dominated, in the sense that no other choice achieves *both* better cost and better execution time at the same time. The problem considered in this paper boils down to choosing a CPU frequency for each resource so that the resulting overall cost and execution time lie on the Pareto front of the solution space formed from the multitude of all different CPU frequency choices (configurations).

The different trade-offs in terms of cost and execution time depend both on the (scientific workflow) application characteristics and the pricing model used.

Overall, using high CPU frequencies may result in good performance, in terms of execution time, but increased cost, as expensive CPU resources are used. Using a lower CPU frequency per resource, the cost may be reduced. However, this may not always be the case, as CPU frequency reduction may significantly impact application execution time. As a result, in some cases this increase in execution time may surpass the savings from the reduced price per time unit (by choosing cheap CPU resources at a lower frequency), leading to an increase of the overall cost. Hence, it is important to build an understanding of the whole solution space, as specified by the different trade-offs, in order to identify solutions that lie on the optimal Pareto front. Clearly, the Pareto front will differ for each scenario (e.g., type of scientific workflow application, pricing model, etc).

To illustrate the complexity of the problem, Figure 1 shows the cost and execution time performance achieved when executing a Montage workflow of 25 tasks on three resources. The results include all the alternative CPU frequency combinations when five CPU frequencies (1-3GHz with a step of 0.5GHz) are available for each resource (this makes 35 different combinations/configurations in total). The price for each resource changes linearly with frequency using the same linear pricing model that is described later in the experimentation in Section 5.1. What is important at this stage is to appreciate the trade-offs that may result even with a relatively small number of different configurations. Labels next to each result plotted in the graph show the mean frequency (in GHz) of the three CPU resources used in each case. It can be seen that even for resource configurations with the same mean frequency different trade-offs are achieved. For example, for a mean frequency of 2.17GHz, the best cost (£$3.9{\cdot}10^{-3}$) and execution time (99.9sec) are achieved when resources run at 2.5GHz, 2.5GHz and 1.5GHz. The solution also belongs to the optimal Pareto front as there is no other configuration with better performance for both cost and execution time. On the other hand, when running the resources at 3.0GHz, 2.0GHz and 1.5GHz (again with a mean frequency of 2.17GHz) a cost of £$4.1{\cdot}10^{-3}$ and an execution time of 103.4sec are achieved; hence, this configuration should not be preferred as it is dominated by better solutions. Finally, it is also noted that the Pareto front consists of only four points: essentially all solutions with a mean CPU frequency less than 2.17GHz are worse in terms of both execution time and cost than (at least) one of the four non-dominated solutions.

3.2. Models used

Application Model. The work in this paper focuses on scientific workflows which are modelled by a Directed Acyclic Graph (DAG) where the nodes are tasks for

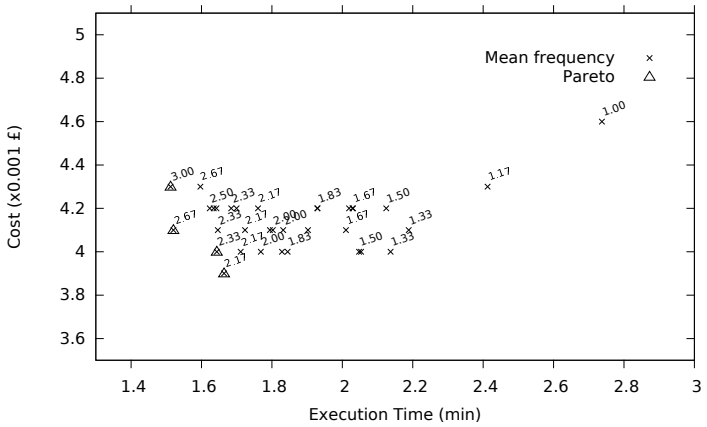


Figure 1: Execution of a Montage workflow of 25 tasks on 3 resources with different frequency configurations.

execution and the edges represent data dependencies between them. It is assumed that data transfer sizes are known. In addition, it is assumed that information about the task runtime characteristics when resources operate at maximum CPU frequency is known. Task runtimes, when running at a frequency f lower than the maximum, can be estimated using the performance model of Equation 1:

$$runtime_{t_f} = (\beta \cdot (\frac{f_{max}}{f} - 1) + 1) \cdot runtime_{t_{f_{max}}}, \quad (1)$$

where $runtime_{t_{f_{max}}}$ is the task runtime when running at the maximum CPU frequency f_{max}. The parameter β takes values between 0 and 1 and captures the task's CPU-boundedness [36, 37]. In brief, a CPU-bound task will have values of β close to 1, while an I/O-bound task will have values of β close to 0. CPU frequency reduction is expected to have a more significant impact on CPU-bound tasks rather than I/O-bound tasks. The value of β for any given task can be computed through profiling [37, 38].

Cloud Resources Model. Each CPU resource is provisioned from the time the workflow execution starts until the time the workflow execution finishes. Each task has exclusive access to the CPU resource slot where it runs. CPU provisioning is charged based on the CPU frequency allocated to each resource; resources can operate on a range of CPU frequencies uniformly distributed between a minimum and maximum frequency, f_{min} and f_{max} respectively, with a step $freqStep$. The cost of other (non-CPU) resources, such as disk and storage, is assumed to be fixed, as this paper focuses on the selection of CPU frequencies.

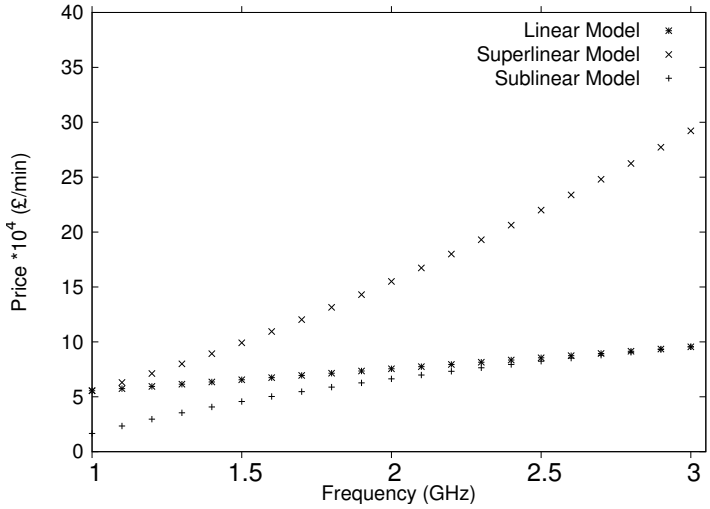


Figure 2: An example for each of the three types of pricing models used.

We consider three different pricing models: *linear*, *sublinear* and *superlinear*. An example of each pricing model is shown in Figure 2. In the linear pricing model, a model frequently used by Cloud providers providing fine-grained CPU pricing, the price is linearly proportional to the frequency. The curve of the two non-linear pricing models can be tuned depending on the provider [39].

In the linear model, the price decreases at the same rate for each pair of successive CPU frequencies while moving from a high to a low CPU frequency. The per unit of time cost, C_{f_r}, of resource r operating at frequency f_r under linear pricing is computed by:

$$C_{f_r} = C_{min} + C_{dif} \cdot (\frac{f_r - f_{min}}{f_{min}}), \tag{2}$$

where C_{min} is the price of the resource operating at minimum frequency, f_{min}, and C_{dif} is a coefficient used to generate the charge at each frequency.

In the superlinear model, the reduction in price is significant while the frequencies are still high and decreases for much lower frequencies. The per unit of time cost under superlinear pricing is given by:

$$C_{f_r} = C_{min} + C_{dif} \cdot ((1 + \frac{f_r - f_{min}}{f_{min}}) \cdot \ln(1 + \frac{f_r - f_{min}}{f_{min}})). \tag{3}$$

In the sublinear model the price increases abruptly while moving from low to higher frequencies while the rate of increase gets significantly smaller for much

higher frequencies. The per unit of time cost under sublinear pricing is computed as:

$$C_{f_r} = C_{min} + C_{dif} \cdot \ln(1 + \frac{f_r - f_{min}}{f_{min}}). \quad (4)$$

As can be observed from the above, each pricing model can be specified as a function of C_{min} and C_{dif}, using the corresponding equation.

Finally, the total user cost for the execution of the whole workflow is computed based on the cost for each resource as:

$$userCost = \sum_{\forall r \in plan} Cost_r, \quad (5)$$

where $Cost_r = C_{f_r} \cdot makespan_{plan}$ is the cost for using resource r at its assigned frequency f_r, during the whole application execution time, $makespan_{plan}$ (which is the same for all resources as all resources are provisioned for the same amount of time).

4. Algorithm Description

In this section, we propose an algorithm to select CPU frequency combinations for the execution of a scientific workflow on a predefined number of cloud resources identifying Pareto-efficient solutions with different cost and execution time trade-offs. The algorithm initially builds workflow execution schedules on the given number of resources for all different combinations of a small set of CPU frequencies (the size of this set will affect the efficiency of the algorithm) to compute an initial set of Pareto-efficient (non-dominated) solutions. Then, an attempt is made to improve the non-dominated solutions using the Cost-based Stepwise Frequency Selection from Heterogeneous Resources (CSFS-H) algorithm to check if better solutions can be achieved by lowering the CPU frequencies of selected resources. Any new solutions are compared with the initial set of solutions and a new Pareto front is computed. The algorithm returns the final set of Pareto-efficient solutions identified.

Pareto-efficient Stepwise Frequency Selection (PSFS) Algorithm. The algorithm (see 1) starts by selecting initially a set containing k equally distributed CPU frequencies (in this paper we choose $k = 3$ and take the minimum, median and maximum available CPU frequency modes) and computes all the possible CPU frequency combinations in the set (lines 2-5). For each frequency combination, the

Algorithm 1 Pareto-efficient Stepwise Frequency Selection.

Require: w: workflow, n: number of resources, k:number of initial frequencies

1: **procedure** PARETOCOMPUTATION(w, n, k)
2: **for** $i \in [0, k-1]$ **do**
3: $fmodes.add(fmode_{min} + \lceil \frac{i \cdot (fmode_{max} - fmode_{min})}{k-1} \rceil)$ ▷ initial CPU frequency modes to be used
4: **end for**
5: $freqSets \leftarrow$ all the different combinations of the initial CPU frequencies ($fmodes$)
6: **for** $freqs \in freqSets$ **do** ▷ $freqs$: the next combination from $freqSets$
7: **for** $r \in [1, n]$ **do**
8: $fmode_r = r \mod k$ ▷ round-robin assignment of frequencies to each resource r
9: $f_r \leftarrow freqsCur_{fmode_r}$ ▷ initial frequency configuration for each resource
10: **end for**
11: $plan_{cur} \leftarrow generateHEFTPlan(f_r)$
▷ Generate HEFT plan using a set of n resources with the frequency configurations in $fr[]$
12: $cost_{plan_{cur}}, makespan_{plan_{cur}} \leftarrow$ cost, makespan of $plan_{cur}$
13: $plans.add(plan_{cur})$
14: **end for**
15: $plans \leftarrow computePareto(plans)$ ▷ Compute the initial pareto front of the plans
16: **for** $plan_{cur} \in plans$ **do**
17: $plan_{new} \leftarrow CSFS-H(plan_{cur})$
▷ Call $CSFS-H$ (see Algorithm 2) for each plan at the pareto front
18: Compute $cost_{plan_{new}}$ and $makespanplan_{new}$ for $plan_{new}$
19: **if** $cost_{plan_{new}} < cost_{plan_{cur}}$ **then**
20: $plans.add(plan_{new})$
21: **end if**
22: **end for**
23: $paretoPlans \leftarrow computePareto(plans)$ ▷ Compute the final pareto front of the plans
return $paretoPlans$
24: **end procedure**

algorithm distributes in a round-robin fashion the CPU frequencies to the available number of resources (lines 7-10) and then generates a workflow execution schedule (plan) using HEFT [6]. The cost and execution time of the plan are computed and the plan is added to the list of potential solutions built (lines 12-13). Once all frequency combinations are considered, the Pareto front is computed and non-dominated solutions (solutions that are not outperformed by other solutions in both execution time and monetary cost) are kept. For each solution kept, CSFS-H (presented next) is used to modify the initial set of assigned CPU frequencies in the plan when reduction in CPU frequency can lead to cheaper configurations. The new plan built is added to the list of the solutions and the same procedure

Algorithm 2 Cost-based Stepwise Frequency Selection from Heterogeneous Resources (CSFS-H).

Require: *curPlan*
1: **procedure** CSFS-H(*curPlan*)
2: $curMode \leftarrow \max fmode_r, \forall r$ ▷ $fmode_r$: the frequency of resource r from $fmodes$
3: **while** $curMode > 0$ **do** ▷ 0 in the case of $f_{min_r}, \forall r$
4: $currentCost \leftarrow$ cost of $curPlan$ ▷ Eq. 5 for all resources and time
5: $curMode - -$ ▷ next available frequency mode
6: $newPlan = curPlan$
7: $candResources = \forall r \in newPlan$ ▷ candidate resources
8: **while** *candResources* not empty **do**
9: **for** $\forall r \in newPlan$ **do**
10: Compute $f = \max f_i \in [f_{r_{curMode}}, f_r)$ with $costSavings_r > 0$
11: **end for**
12: $candResources = \forall r \in newPlan$: $costSavings_r > 0$
13: Remove $r \in candResources$ with the largest $costSavings_r$
▷ for the frequency f selected for resource r in line 10
Update task runtimes for each task $t \in r$ using Eq. 1
Update start and finish times for each task $t \in w$ in the plan (*newPlan*)
14: **end while**
15: $newCost \leftarrow$ cost of $newPlan$
16: **if** $newCost >= currentCost$ **then** Reject *newPlan* and break
17: **end if**
18: Accept plan ($curPlan = newPlan$)
19: **end while**
20: **end procedure**

continues with the next plan in the initial Pareto front. The algorithm computes the new Pareto front and returns the final set of non-dominated solutions kept.

Cost-based Stepwise Frequency Selection from Heterogeneous Resources (CSFS-H) Algorithm. This algorithm (see Algorithm 2) is an extension of the Cost-based Stepwise Frequency Selection from Maximum Frequency (CSFS-Max) algorithm in [5] that adjusts the CPU frequencies of an initial execution plan trying to achieve cheaper configurations. CSFS-H takes as input an initial plan where the CPU frequency per resource may vary and iteratively reduces the frequency of each resource based on the impact of frequency reduction on overall cost savings. The procedure used in each iteration works as follows. The next mode of the currently lowest assigned frequency between the resources is used as a lower bound to iteratively modify the assigned configuration. For each resource the maximum frequency f between the lower bound ($f_{r_{curMode}}$) and the currently assigned fre-

quency that leads to overall cost savings for the workflow is kept. For a resource r overall cost savings from the transition to a lower frequency f are computed as:

$$costSavings_r = curCost_{f_r} - newCost_f, \tag{6}$$

where $curCost_{f_r}$ is the cost of the workflow when the resource operates at its currently assigned frequency, f_r, and $newCost_f$ is the cost at the resulted schedule where a lower frequency $f \in [f_{r_{curMode}}, f_r)$ is used. Resources where the reduction in frequency leads to cost savings are considered to be candidates to adjust their frequency. Starting with the most cost-efficient resource, the runtime of each task assigned to this resource is updated for the new selected frequency, f, and the slots of all the workflow tasks are adjusted to build a new plan. The procedure is repeated until there is no other candidate resource to adjust its configuration based on the current available frequency mode $curMode$. When there are no candidate resources in an iteration or the minimum frequency is reached for all resources, the algorithm terminates and returns the modified plan.

The time complexity of the algorithm for the computation of the initial Pareto front (Algorithm 1) depends on: (i) the number of the different combinations of frequencies $freqSets = \frac{(k+n-1)!}{k!\cdot(n-1)!}$, where k is the number of initial CPU frequencies per resource and n is the number of available resources; and (ii) the complexity of HEFT algorithm ($O(e \cdot n)$, where e is the number of edges in the DAG). In the second stage (Algorithm 2), the complexity to explore the available CPU frequencies iteratively and compute the final Pareto front is mainly influenced by the number of available resources n, the number of available CPU frequencies f_{modes}, the number of solutions obtained in the initial Pareto front s and the number of the workflow tasks $tasks$: $O(n \cdot fmodes \cdot s \cdot tasks)$. However, a key idea of the approach is that some frequency values may be skipped using a good sampling of the search space to reduce the complexity of the algorithm. Overall, in our experiments, the runtime of the algorithm took from several seconds (less time consuming runs) up to a few minutes (worst-case scenarios where a larger set of frequencies needed to be explored).

5. Evaluation

5.1. Methodology

The simulator in [40] was used to implement and evaluate the proposed approach. Resources (one-core VMs) that operate on a set of CPU frequencies equally distributed between a minimum and maximum frequency, $f_{min} = 1000MHz$

and $f_{max} = 3000MHz$, respectively, with a frequency step of $100MHz$ are assumed. A network of 1Gbps is used to compute the communication cost between tasks that do not run on the same VM.

Three different pricing models are used, as described by Equations 2, 3 and 4 (linear, superlinear and sublinear pricing, respectively). The following values for their key parameters, C_{min} and C_{dif} have been chosen:
$C_{min} = £9.24 * 10^{-6}$ and $C_{dif} = £3.33 * 10^{-6}$, for the linear pricing model;
$C_{min} = £9.24 * 10^{-6}$ and $C_{dif} = £4.44 * 10^{-6}$, for the superlinear pricing model;
$C_{min} = £2.78 * 10^{-6}$ and $C_{dif} = £1.2 * 10^{-5}$, for the sublinear pricing model.
The values were chosen to approximate roughly the monthly charges of ElasticHosts [1] for the provisioning of VMs, assuming time units in seconds.

Four real scientific workflows (LIGO [41], Montage [42], SIPHT [43] and Cybershake [44]) are used to evaluate the proposed approach and investigate how Pareto sets can be built for different applications. A size of 1000 was selected for each application, based on synthetic data obtained by the workflow generator in [45]. This size corresponds to 1000 tasks for LIGO, Montage and Cybershake and

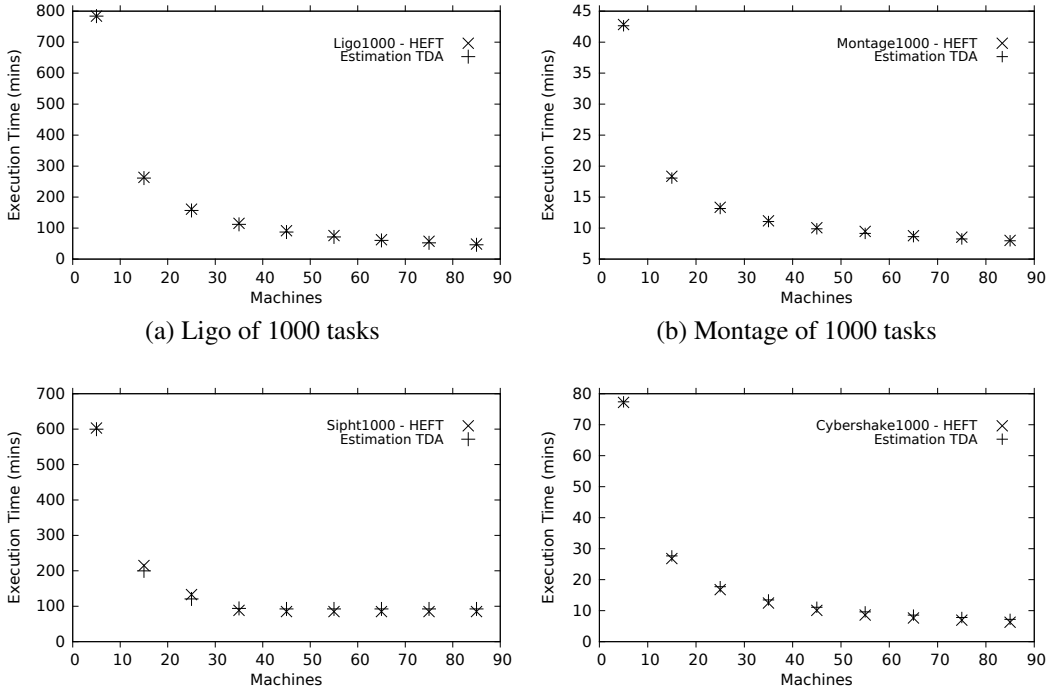
LIGO			
Job classes	#Tasks	β	runtime(sec)
TmpltBank	242	0.9894	18.12
Inspiral	497	0.8996	461.62
Thinca	6	0.4390	5.23
TrigBank	255	0.1744	5.09

SIPHT			
Job classes	#Tasks	β	runtime(sec)
Patser	584	0.8348	1.27
Patser_concate	32	0.1889	0.08
Transterm	32	0.9479	53.14
Findterm	32	0.9520	1401.20
RNAMotif	32	0.9505	35.43
Blast	32	0.9387	2288.88
SRNA	32	0.9348	370.66
FFN_Parse	32	0.8109	1.54
Blast_synteny	32	0.6101	33.00
Blast_candidate	32	0.4361	5.41
Blast_QRNA	32	0.8780	1404.97
Blast_paralogues	32	0.4430	4.79
SRNA_annotate	32	0.5596	1.76

Montage			
Job classes	#Tasks	β	runtime(sec)
mProjectPP	166	0.8696	13.63
mDiffFit	662	0.2839	10.59
mConcatFit	1	0.5317	52.96
mBgModel	1	0.9989	128.24
mBackground	166	0.0846	10.73
mImgTbl	1	0.0348	37.78
madd	1	0.0848	100.55
mShrink	1	0.0230	22.25
mJPEG	1	0.7714	5.73

Cybershake			
Job classes	#Tasks	β	runtime(sec)
ExtractSGT	6	0.6582	127.05
SeisSynthesis	496	0.9201	44.01
ZipSeis	1	0.0683	2.34
PeakVCOkaya	496	0.1689	1.09
ZipPSA	1	0.0289	5.31

Table 1: Workflow task characteristics.



(a) Ligo of 1000 tasks

(b) Montage of 1000 tasks

(c) Sipht of 1000 tasks

(d) Cybershake of 1000 tasks

Figure 3: Workflow performance for a different number of resources.

968 tasks for SIPHT. Information about task runtimes when resources operate at the maximum CPU frequency (as mentioned above, this is 3GHz in our case) is also provided. As the tasks of each workflow belong to one of several job classes with common characteristics, the parameter β for each workflow task (to use in Equation 1 to estimate task runtimes when running at a lower than the maximum CPU frequency) was set equal to the average CPU utilization of its job class using workflow profiling data from [46]. Information related to the characteristics of each job class for each workflow used is provided in Table 1 and further discussed next.

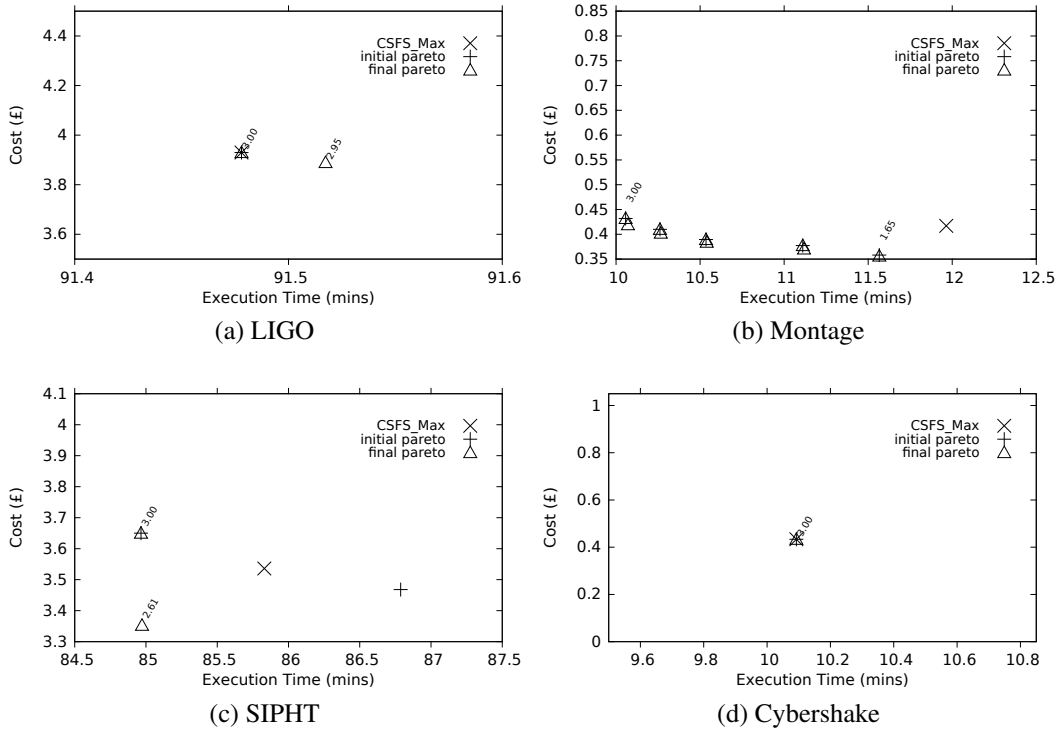
LIGO, a real scientific workflow used for the detection of gravitational waves in the universe, mainly consists of parallel CPU-bound jobs that process large amounts of data. Inspiral jobs with relatively high CPU utilization comprise the largest job class (almost half of the workflow tasks) and consume the largest part of the runtime and I/O usage. An important part of the total runtime and I/O usage is also spent on TmpltBank tasks which are the most computationally intensive

tasks reading significant amounts of data. The rest of the tasks have significantly shorter runtimes with low CPU and I/O usage. Montage is a workflow used for the creation of image mosaics of the sky. Overall, Montage spends most of the runtime on I/O operations. Several job classes have short runtimes and low CPU utilization such as the case of mDiffFit, mBackground and mShrink, while the single mBgModel task has the longest runtime and CPU utilization (99.89%). SIPHT is a CPU-intensive workflow used to search for sRNA encoding genes for bacterial replicons. Overall SIPHT consists of job classes with high CPU utilization and low I/O activity. Among them, the Blast, Findterm and Blast_QRNA classes have the tasks with the longest runtimes; only the Patser_concate tasks have low CPU usage (about 19%). Finally, Cybershake [44] is workflow that is used to characterize earthquake hazards at a site. The majority of the runtime is spent on the most computationally intensive tasks, the SeismogramSynthesis tasks, which also exhibit high memory requirements.

The number of cloud resources to provision was selected to be 45, based on workflow performance estimates obtained using the TDA modelling approach in [30]. The objective was to find a number of resources for the given workflows that allows scheduling and execution in a way that resource wastage is avoided. Figure 3 shows the estimated makespan for the four workflows when a varying number of resources (up to 90) that operate at maximum frequency is available. The makespan obtained for the same configurations using HEFT is also included to validate further the accuracy of the model. As can be seen, the curve converges towards some minimum execution time and tends to get flatter as the number of resources increases. This suggests that any performance improvement from the use of extra resources becomes less significant when the number of resources becomes larger. As with the SIPHT workflow the curve is flattened after 45 resources, this is the number of resources that was chosen for the evaluation.

5.2. Results

First we consider the linear pricing model. Figure 4 shows the results obtained for each workflow application. The results include the initial and final Pareto set generated using the proposed approach, PSFS, and the solution built using CSFS_Max proposed earlier in [5]; for the latter, a long deadline is assumed so that configurations at minimum CPU frequency can be reached. To avoid clutter, two points from the final Pareto set are annotated with labels in the graph. One point shows the mean CPU frequency of the configuration used by the fastest plan (which typically corresponds to the solution generated using HEFT when resources operate at maximum frequency), another point shows the mean CPU



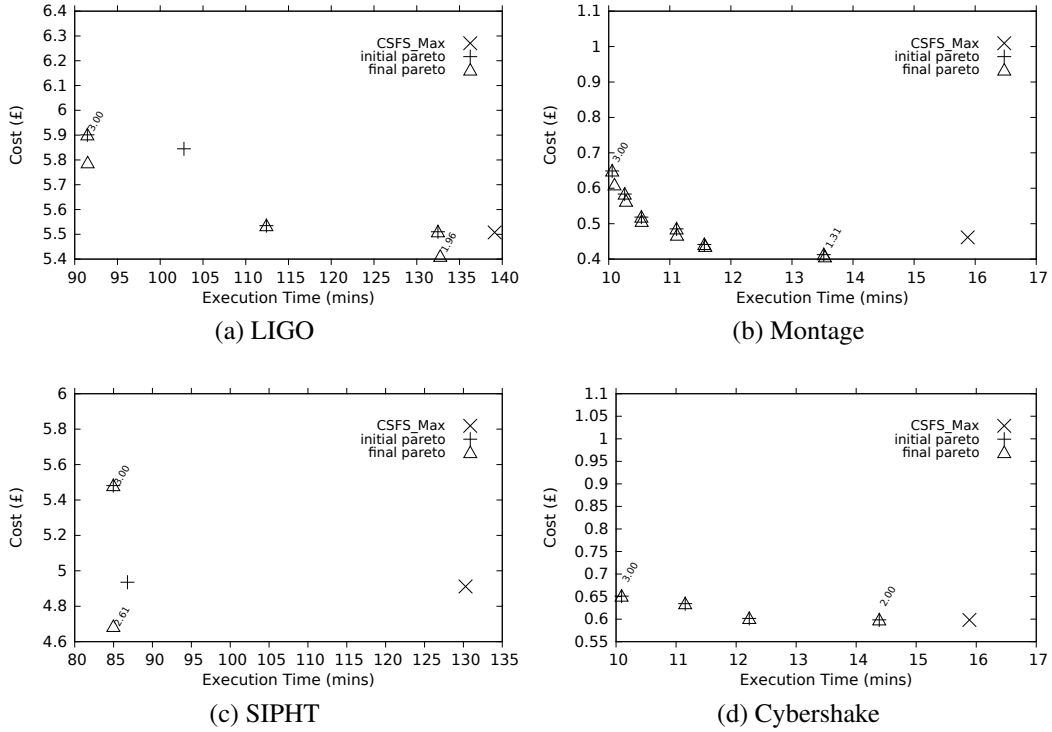
(a) LIGO (b) Montage

(c) SIPHT (d) Cybershake

Figure 4: Workflow execution on 45 resources using the linear pricing model.

frequency of the least expensive plan. Overall, it can be seen that the solutions obtained using the proposed approach, PSFS, outperform the solution obtained by CSFS_Max. In the case of LIGO, SIPHT and Cybershake the final Pareto set consists of only solutions with a relatively high mean CPU frequency. This may be because the impact of frequency reduction on task runtime and overall makespan may be significant, as most of the tasks have high CPU utilization. In the case of the I/O intensive Montage workflow, a more diverse set of Pareto-efficient solutions with different trade-offs between execution time and cost can be achieved by varying the CPU frequency of the resources.

In the case of superlinear pricing (Figure 5), the set of non-dominated solutions also includes configurations with a relatively lower mean CPU frequency compared to linear pricing. This is because, with superlinear pricing, the reduction in price decreases more abruptly with the reduction in frequency. Although overall execution time increases while moving to lower frequencies, the overall cost required for the workflow execution is reduced at a significant rate. Once



(a) LIGO (b) Montage

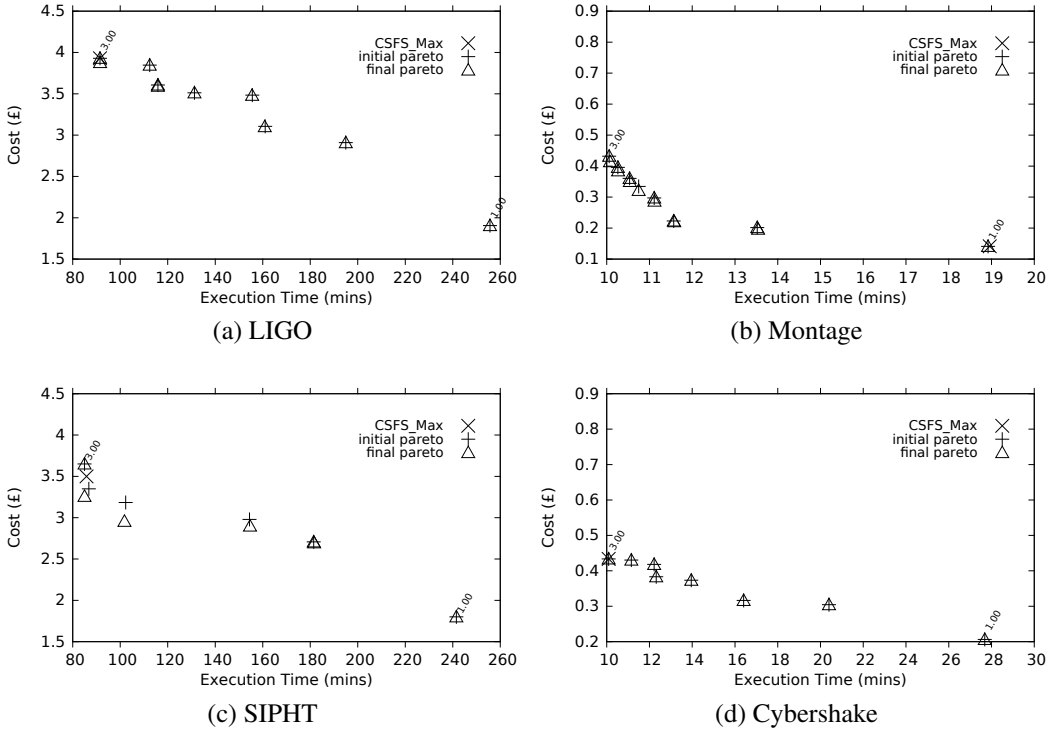
(c) SIPHT (d) Cybershake

Figure 5: Workflow execution on 45 resources using the superlinear pricing model.

again, Montage achieves a Pareto-efficient solution with the lowest mean CPU frequency (1.31GHz).

In the case of sublinear pricing (Figure 6), CSFS_Max fails to explore configurations with low CPU frequencies which are cost efficient, as the reduction in price is insignificant while frequencies are still high. In this case, PSFS produces a more diverse set of Pareto-efficient solutions, which include configurations with low mean CPU frequencies for all four workflows. It is interesting that cost and execution time between the two extreme points (the configurations with the highest and lowest mean CPU frequency in the Pareto set) vary more in the case of SIPHT and LIGO which are long, CPU-bound workflows.

In summary, the results suggest that the proposed algorithm, PSFS, can identify a set of Pareto-efficient configurations, which depends on both the application characteristics and the pricing policy. Using linear pricing, with I/O-intensive applications the Pareto set may include configurations with relatively low mean CPU frequencies but this is not the case with CPU-bound workflows (as such workflows



(a) LIGO (b) Montage

(c) SIPHT (d) Cybershake

Figure 6: Workflow execution on 45 resources using the sublinear pricing model.

are affected more by CPU frequency reduction and configurations with low mean CPU frequencies may result only in dominated solutions). This becomes easier to achieve with superlinear and particularly sublinear pricing, which help produce a diverse set of Pareto-efficient configurations.

6. Conclusion

This paper proposed an approach to select Pareto-efficient CPU frequency configurations for the execution of scientific workflows on a selected number of cloud resources. The main idea has been to prune the search space by carefully selecting a small set of initial CPU frequencies to build an initial Pareto front, which a subsequent algorithm attempts to improve. The proposed approach was evaluated through simulation for three different pricing models that charge resource provisioning based on the selected CPU frequencies. The key findings suggest that the outcome depends on both application characteristics and the pric-

ing model used.

Future work could investigate how to refine the initial set of CPU frequency configurations to take into account specific application characteristics and the provider's pricing model. This, for instance, may suggest that an initial, unevenly distributed set of CPU frequencies may enable the algorithm produce better results. In addition, given the sensitivity that heuristics such as HEFT may exhibit, different scheduling algorithms may be used to build execution schedules on resources and obtain the initial Pareto set. Further work could also assess the impact of the scheduling algorithm used for mapping tasks onto resources with respect to the quality of the final Pareto set achieved.

References

[1] ElasticHosts, `http://www.elastichosts.co.uk/`, Available online.

[2] CloudSigma, `https://www.cloudsigma.com/`, Available online.

[3] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-science: An overview of workflow system features and capabilities, Future Generation Computer Systems 25 (2009) 528 – 540.

[4] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, E. Deelman, A characterization of workflow management systems for extreme-scale applications, Future Generation Computer Systems 75 (2017) 228 – 238.

[5] I. Pietri, R. Sakellariou, Cost-efficient CPU provisioning for scientific workflows on clouds, in: Proceedings of the 12th International Conference on Economics, Grids, Clouds and Systems (GECON), Springer, 2015, pp. 49–64.

[6] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (2002) 260–274.

[7] T. T. Huu, J. Montagnat, Virtual Resources Allocation for Workflow-Based Applications Distribution on a Cloud Infrastructure, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID), IEEE, 2010, pp. 612–617.

[8] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, T. Fahringer, A multi-objective approach for workflow scheduling in heterogeneous environments, in: Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), IEEE, 2012, pp. 300–309.

[9] J. Li, S. Su, X. Cheng, Q. Huang, Z. Zhang, Cost-conscious scheduling for large graph processing in the cloud, in: Proceedings of the IEEE International Conference on High Performance Computing and Communications, IEEE, 2011, pp. 808–813.

[10] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large programs in the cloud, Parallel Computing 39 (2013) 177–188.

[11] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of Scientific Workflows in the ASKALON Grid Environment, SIGMOD Record 34 (2005) 56–62.

[12] L.-C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative evaluation of the robustness of DAG scheduling heuristics, in: Grid Computing: Achievements and Prospects, 2008.

[13] Y.-K. Kwok, I. Ahmad, Benchmarking and comparison of the task graph scheduling algorithms, Journal of Parallel and Distributed Computing 59 (1999) 381 – 422.

[14] R. Sakellariou, H. Zhao, A hybrid heuristic for DAG scheduling on heterogeneous systems, in: Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004, p. 111.

[15] S. Abrishami, M. Naghibzadeh, D. Epema, Cost-driven Scheduling of Grid Workflows Using Partial Critical Paths, IEEE Transactions on Parallel and Distributed Systems 23 (2012) 1400–1414.

[16] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Generation Computer Systems 27 (2011) 1011–1026.

[17] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC11), ACM, 2011, p. 49.

[18] M. Mao, M. Humphrey, Scaling and scheduling to maximize application performance within budget constraints in cloud workflows, in: Proceedings of the 27th International Symposium on Parallel & Distributed Processing (IPDPS), IEEE, 2013, pp. 67–78.

[19] H. Arabnejad, J. G. Barbosa, A budget constrained scheduling algorithm for workflow applications, Journal of Grid Computing 12 (2014) 665–679.

[20] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Scientific Programming 14 (2006) 217–230.

[21] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, Future Generation Computer Systems 48 (2015) 1 – 18.

[22] S. Abrishami, M. Naghibzadeh, D. H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Generation Computer Systems 29 (2013) 158 – 169.

[23] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, R. Sakellariou, Energy-constrained provisioning for scientific workflow ensembles, in: International Conference on Cloud and Green Computing, 2013, pp. 34–41.

[24] I. Pietri, R. Sakellariou, Energy-aware workflow scheduling using frequency scaling, in: 43rd International Conference on Parallel Processing Workshops, 2014, pp. 104–113.

[25] R. Sakellariou, H. Zhao, E. Tsiakkouri, M. D. Dikaiakos, Scheduling workflows with budget constraints, in: S. Gorlatch, M. Danelutto (Eds.), Integrated Research in GRID Computing: CoreGRID Integration Workshop 2005 (Selected Papers) November 28–30, Pisa, Italy, Springer US, Boston, MA, 2007, pp. 189–202.

[26] K. Lee, N. W. Paton, R. Sakellariou, A. A. A. Fernandes, Utility functions for adaptively executing concurrent workflows, Concurrency and Computation: Practice and Experience 23 (2011) 646–666.

[27] K. Bessai, S. Youcef, A. Oulamara, C. Godart, S. Nurcan, Bi-criteria workflow tasks allocation and scheduling in cloud computing environments, in: Proceedings of the 5th IEEE CLOUD, IEEE, 2012, pp. 638–645.

[28] J. J. Durillo, H. M. Fard, R. Prodan, MOHEFT: A multi-objective list-based method for workflow scheduling, in: Proceedings of the 4th IEEE CloudCom, IEEE, 2012, pp. 185–192.

[29] I. Pietri, Y. Chronis, Y. Ioannidis, Multi-objective optimization of scheduling dataflows on heterogeneous cloud resources, in: Proceedings of the 5th International Conference on Big Data, IEEE, 2017.

[30] I. Pietri, G. Juve, E. Deelman, R. Sakellariou, A performance model to estimate execution time of scientific workflows on the cloud, in: Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science (WORKS), IEEE, 2014, pp. 11–19.

[31] M. Wieczorek, A. Hoheisel, R. Prodan, Towards a general model of the multi-criteria workflow scheduling on the grid, Future Generation Computer Systems 25 (2009) 237 – 256.

[32] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, R. Mirandola, MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems, IEEE Transactions on Software Engineering 38 (2012) 1138–1159.

[33] V. Cardellini, V. D. Valerio, V. Grassi, S. Iannucci, F. L. Presti, A new approach to QoS driven service selection in service oriented architectures, in: Proceedings of the 6th IEEE International Symposium on Service Oriented System, 2011, pp. 102–113.

[34] D. Menasce, H. Gomaa, s. Malek, J. Sousa, SASSY: A framework for self-architecting service-oriented systems, IEEE Software 28 (2011) 78–85.

[35] Flexible service selection with user-specific QoS support in service-oriented architecture, Journal of Network and Computer Applications 35 (2012) 962 – 973.

[36] C.-H. Hsu, U. Kremer, The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction, ACM SIGPLAN Notices 38 (2003) 38–48.

[37] M. Etinski, J. Corbalan, J. Labarta, M. Valero, Optimizing job performance under a given power constraint in HPC centers, in: Proceedings of the International Green Computing Conference, IEEE, 2010, pp. 257–267.

[38] M. Etinski, J. Corbalan, J. Labarta, M. Valero, Understanding the future of energy-performance trade-off via DVFS in HPC environments, Journal of Parallel and Distributed Computing 72 (2012) 579–590.

[39] W. Shi, B. Hong, Towards Profitable Virtual Machine Placement in the Data Center, in: Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing, IEEE, 2011, pp. 138–145.

[40] Cloud Workflow Simulator, https://github.com/malawski/cloudworkflowsimulator, Available online.

[41] LIGO project, Laser interferometer gravitational wave observatory, http://www.ligo.caltech.edu/, Available online.

[42] D. S. Katz, J. C. Jacob, E. Deelman, C. Kesselman, G. Singh, M.-h. Su, G. Berriman, J. Good, A. Laity, T. A. Prince, A comparison of two methods for building astronomical image mosaics on a grid, in: Proceedings of the IEEE International Conference on Parallel Processing Workshops (ICPPW), IEEE, 2005, pp. 85–94.

[43] J. Livny, H. Teonadi, M. Livny, M. K. Waldor, High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs, PLoS One 3 (2008) e3197.

[44] P. Maechling, E. Deelman, L. Zhao, et al., SCEC CyberShake workflows—automating probabilistic seismic hazard analysis calculations, in: Workflows for e-Science: Scientific Workflows for Grids, Springer, 2007, pp. 143–163.

[45] Workflow Generator, https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, Available online.

[46] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Generation Computer Systems 29 (2013) 682–692.