Cost-efficient CPU Provisioning for Scientific Workflows on Clouds

Ilia Pietri and Rizos Sakellariou

School of Computer Science, University of Manchester, U.K

Abstract. Cloud providers now offer resources as combinations of CPU frequencies and prices, with faster resources (which operate at higher frequencies) charged at a higher monetary cost. With the emergence of this new pricing scheme, the problem of choosing cost-efficient configurations is becoming even more challenging for users. The frequencies required to achieve cost-efficient configurations may vary in different scenarios, depending on both the provider’s pricing model and the application characteristics. In this paper, two cost-aware algorithms that select low-cost CPU frequencies for each resource to complete a scientific workflow application within a deadline and at a minimum cost are presented. The proposed approaches are evaluated and compared through simulation using different pricing models that charge resource provisioning also based on the CPU frequency.

1 Introduction

Cloud providers now offer users a large number of configurations, charging CPU provisioning based on the selected frequency of each resource. For example, ElasticHosts [7] and CloudSigma [6] offer users the flexibility of choosing the computing capacity from a wide range of options with more powerful and faster resources (which operate at higher frequencies) costing more. In their pricing model, even 1MHz of CPU frequency more may still incur a small but still higher monetary cost. As users can choose between configurations with different cost and execution time performance, a challenge that arises is selecting the CPU frequency per resource so that cost-efficient configurations that meet the user’s application needs are achieved.

Although providers may charge a lower price for the provisioning of resources that operate at a lower frequency, the reduction in frequency may adversely affect the overall cost incurred by the user. This is because frequency reduction may impact application performance in different ways, depending on the application characteristics. For example, CPU-bound applications that are more sensitive to the allocated frequency show significantly poorer performance when executed at lower frequencies. As a result, a CPU-intensive application may need to run on fast resources to meet a specified deadline or incur a lower cost, as execution time may be greatly affected when a lower frequency is used. In other scenarios, running the resources at a lower speed sometimes with a slightly longer execution time, may be more cost-efficient, motivating users to choose configurations with lower speed performance. This problem may be more profound in applications such as scientific workflows [26], which consist of collections

of tasks with a different behaviour (some of them may be CPU-bound, some of them may be I/O-bound). The execution schedule of such applications may also include gaps (idle time) as a result of task dependencies that need to be satisfied. This gives further scope to take advantage of such gaps by choosing and provisioning CPU resources at different frequencies that minimize the cost, yet achieve a reasonable makespan within a deadline.

Assuming that users are interested in completing a workflow within a specified deadline, this paper contributes two algorithms that can be used to obtain cost-efficient resource provisioning for scientific workflows by choosing appropriate CPU frequencies. Cost-efficient provisioning is understood as provisioning that results in a low overall cost for the user. Typically a low overall cost is not achieved by choosing either high-frequency resources alone or low-frequency resources alone. Instead, cost-efficient provisioning is typically achieved by a combination of resources in between high and low frequencies. The two algorithms start with an initial allocation where all CPU resources are chosen at either high-frequency or low-frequency. The algorithm starting with highest frequencies (Cost-based Stepwise Frequency Selection from Maximum Frequency, CSFS-Max) is iteratively trying to reduce CPU frequencies for as long as the cost is minimized and prioritizing the biggest cost savings at the time (building on an idea suggested in an earlier poster in [19]). Conversely, the algorithm starting with lowest frequencies (Cost-based Stepwise Frequency Selection from Minimum Frequency, CSFS-Min) is iteratively trying to increase CPU frequencies by prioritizing the best makespan savings subject to an overall cost saving. The two algorithms are comprehensively evaluated using three different scientific workflow applications and three different pricing models for CPU frequency. These pricing models include a linear, a sublinear, and a superlinear pricing model, which capture three different ways of varying the price of resources based on CPU frequency. By appropriately choosing CPU frequencies, the two proposed algorithms achieve cost savings that allow users to meet their deadlines at a low cost.

The rest of the paper is organized as follows. Related work is described in Section 2. The model and assumptions made follow in Section 3. The algorithms developed are presented in Section 4. Finally, the evaluation and results are explained in Section 5, while Section 6 concludes the paper.

2 Related Work

Existing work on task scheduling and resource provisioning addresses various optimization goals in different computing systems such as grids [1, 9, 20, 22] and clouds [3, 11, 17]. Execution time and cost optimization is the focus in many studies [14, 25, 27]. The Heterogeneous Earliest Finish Time (HEFT) algorithm in [27] is a well known heuristic that deals with workflow scheduling on heterogeneous systems. HEFT schedules each task by selecting the slot with the earliest finish time from a number of available resources, also taking into account communication costs. In [14], HEFT is extended by introducing a cost-conscious factor to select slots with a balance between execution time and cost. In that way, cost-efficient schedules with an acceptable penalty on execution time are developed. The work in [25] considers different combinations of

Virtual Machine (VM) capacities and prices to reduce the monetary cost, while achieving good performance in terms of makespan. Their algorithm initially allocates tasks to the cheapest VMs and then reassigns non-critical tasks (tasks that are not in the critical path) to less expensive VMs stretching their execution time without impacting overall makespan.

A lot of work focuses on workflows with deadline constraints [2, 3, 11, 18, 28]. The work in [3] tries to determine the minimum resource capacity required to minimize the workflow cost within a deadline constraint to achieve elastic resource provisioning. This is done by dividing the execution of the workflow into time intervals, set equal to the time unit used in the pricing, and estimating the resource capacity required at each period. The algorithm in [28] schedules tasks to the cloud resources using Particle Swarm Optimization so that both execution time and cost are minimized. The algorithm also considers a budget constraint for the workflow. Similarly, in [18], a scalable environment to the user is provided by choosing proper VM instance types to dynamically schedule and execute the jobs within the deadline while reducing the cost.

In contrast to all related work, the two algorithms presented in this paper determine CPU frequencies of the resources to be provisioned in order to lead to cost-efficient execution within a specified deadline, taking also into account different pricing models to charge for the use of such CPU frequencies.

3 Problem Description and Assumptions

In the problem considered, the user submits a workflow for execution to be completed within a specified deadline on a number of available resources. The aim is to determine the CPU frequencies of the resources in order to achieve cost-efficient configurations. Each resource is provisioned from the time the workflow execution starts until the whole workflow finishes. CPU capacity is charged according to the CPU frequency selected for each resource, while the cost of other characteristics, such as disk and storage, is considered to be fixed, as this paper investigates only issues related to the selection of CPU frequencies.

Application Model The paper considers scientific workflows modelled as a Directed Acyclic Graph (DAG) assuming that information on task runtimes, when resources operate at maximum CPU frequency, and data transfer times is provided. Task runtimes, when running at a frequency lower than the maximum, can be estimated based on the task's CPU-boundedness [10, 8]. Hence, the runtime of a task t at a frequency f is given by:

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

where $runtime_{t_{f_{max}}}$ is the task runtime when running at maximum frequency, f_{max} and the parameter β captures the impact of the CPU frequency on job runtime, ranging between 0 and 1 for jobs with low or high CPU activity, respectively [8].

Cloud Resources Model Homogeneous resources which can work on different CPU frequencies, regularly distributed between a minimum and maximum frequency, f_{min}

and f_{max} respectively, with step $freqStep$ are assumed. Each resource is charged for the time provisioned according to the allocated frequency with each workflow task having exclusive control on the resource slot where it runs. The curve of the pricing model may be tuned according to the provider's needs and goals [24]. For example, the cost of resources may increase superlinearly for more capable (faster) resources [23], or it may increase sublinearly with the performance level as in [23, 24]. Similarly, in [14] the price increases exponentially with the speed of the VMs.

Hence, three different pricing models are investigated in this paper: *linear*, *sublinear* and *superlinear*; these are shown in Fig. 1.

In the case of the linear model, when the frequency is reduced, the price of provisioning a resource decreases at the same rate for each pair of successive frequencies. In the case of the superlinear model, the reduction in price is significant while the frequencies are still high, but gets smaller while approaching low frequencies. In contrast to superlinear pricing, when using a sublinear model the increase in price is significant while moving from low to higher frequencies and reduces while approaching high frequencies.

To compute the price charged for each frequency three different functions are used, one for each different pricing model. For linear pricing, the per unit of time cost, C_{f_r} , of resource r operating at frequency f_r is computed by

$$C_{f_r} = C_{min} + C_{dif} * \left(\frac{f_r - f_{min}}{f_{min}} \right), \quad (2)$$

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. The per unit of time resource cost under superlinear pricing is given by:

$$C_{f_r} = C_{min} + C_{dif} * \left(\left(1 + \frac{f_r - f_{min}}{f_{min}} \right) * \log \left(1 + \frac{f_r - f_{min}}{f_{min}} \right) \right), \quad (3)$$

Fig. 1: Pricing models used.

Fig. 2: Cost of CPU provisioning for 4 resources at the same frequency.

Fig. 3: Cost for different combinations of CPU frequencies (3 resources).

while the cost under sublinear pricing is computed as:

$$C_{f_r} = C_{min} + C_{dif} * \log\left(1 + \frac{f_r - f_{min}}{f_{min}}\right). \quad (4)$$

The overall cost required to run the workflow is computed by Eq. 5:

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

where $Cost_r = C_{f_r} \cdot makespan_{plan}$ is the cost of running each resource r in the schedule ($plan$) at its assigned frequency f_r for the whole scheduling period, $makespan_{plan}$.

Motivation Using the lowest possible frequencies to run a workflow within the specified deadline may not always lead to cost savings, as application performance degradation may be significant. This observation is an important aspect to help appreciate the contribution of the paper. To illustrate this, Fig. 2 shows the cost to execute the LIGO workflow [15] with 50 tasks on 4 resources. All resources run at the same CPU frequency but five different CPU frequencies are considered from low to high; the price charged for each resource changes linearly with the reduction in frequency. It can be seen that while moving from high to low frequencies the cost decreases until 2GHz is reached but then it increases as it approaches the lowest frequency (1GHz). This suggests that although the overall workflow execution time increases when moving from high to low frequencies, there is a point until which the cost can be minimized due to a more efficient resource utilization.

We also investigated the performance of the same workflow in terms of execution time (makespan) and cost for all the possible combinations of CPU frequencies in the range of 1-3GHz with a step of 0.5GHz. Fig. 3 shows the results when the workflow runs using 3 resources. The label shows the mean frequency for the three resources used in each run. Varying CPU frequencies results in different execution time and cost, even if the mean CPU frequency is still the same. For example, for a mean frequency of 2.17GHz, the best cost (£0.192) and makespan (4869.7sec) is achieved with resources running at 2.5GHz, 2GHz and 2GHz. At the same time, resources running at 2.5GHz,

2.5GHz, and 1.5GHz (same mean frequency) result in a cost of £0.209 and makespan of 5308.9 sec.

The two figures above suggest that different trade-offs between cost and execution time exist. In some sense, this work attempts to explore feasible nearby solutions of a Pareto frontier starting from an extreme point, e.g. maximum or minimum frequency, in order to achieve locally optimal solutions based on the application characteristics and the pricing model of the provider.

4 Algorithm Description

In this section two cost-aware algorithms to determine the frequencies per resource to be provisioned to execute a workflow at a minimum cost within a deadline are described. The algorithms can work for different pricing models with the per unit of time cost decreasing when a lower CPU frequency is used. The output of the two algorithms is an execution plan (or simply plan) that schedules tasks onto resources and determines an appropriate frequency for each resource.

Cost-based Stepwise Frequency Selection from Maximum Frequency (CSFS-Max). Initially, a schedule (plan) of the workflow tasks onto a number of resources is built, assuming that the resources operate at maximum frequency. This initial schedule is built using HEFT [27]. It is noted that the use of HEFT to build an initial schedule is not an

---

Algorithm 1 Cost-based Stepwise Frequency Selection from Maximum Frequency.

Require: w : workflow, $curPlan$: HEFT plan at maximum frequency, $deadline$: user deadline

```

1: procedure FREQUENCYSELECTIONMAX( $w, curPlan$ )
2:   while  $freq > f_{min}$  do                                     ▷ Starting with  $freq = f_{max}$ 
3:      $currentCost$  : cost of  $curPlan$                                ▷ Eq. 5 for all resources and time
4:      $freq- = freqStep$                                            ▷ next available lower frequency
5:      $newPlan = curPlan$ 
6:      $candResources = \forall r \in newPlan$                              ▷ candidate resources
7:     while  $candResources$  not empty do
8:       for  $\forall r \in newPlan$  do
9:         Compute  $costSavings_r$                                    ▷ 0 when deadline is exceeded
10:      end for
11:       $candResources = \forall r \in newPlan: costSavings_r > 0$ 
12:      Remove  $r \in candResources$  with largest  $costSavings_r$ 
13:      Update task runtimes for each task  $t \in r$  using Eq. 1
14:      Update start and finish times for each task  $t \in w$  in the plan ( $newPlan$ )
15:    end while
16:     $newCost$  : cost of  $newPlan$ 
17:    if  $newCost \geq currentCost$  then Reject  $newPlan$  and break
18:    end if
19:    Accept plan ( $curPlan = newPlan$ )
20:  end while
21: end procedure

```

---

intrinsic requirement of the proposed algorithm. In fact, such an initial schedule can be built by any DAG scheduling algorithm such as HBMCT [21] or any of the heuristics compared in [4].

The algorithm (shown in Alg. 1) iteratively lowers the frequencies of the resources following the range of the available frequencies to determine in each step whether cost savings can be achieved for each resource by provisioning a lower frequency. At the end of each iteration (for each available frequency) the impact on overall workflow cost from using the lower selected frequencies than the currently assigned frequencies to the resources is assessed to determine if the new plan will be accepted (lines 15-17).

To do so, in each iteration the next available frequency, $freq$, is used to lower the frequency of each resource when the cost for the resource decreases with the reduction in frequency, prioritizing resources with higher cost savings. The detailed steps repeated for each iteration (available frequency) are the following: The cost savings of each resource r from the transition to a lower frequency f are computed as:

$$costSavings_r = curCost_{f_r} - newCost_f, \quad (6)$$

where $curCost_{f_r}$ is the cost of running the resource at its currently assigned frequency, f_r , and $newCost_f$ is the cost of the resource at the resulted schedule where a lower frequency $f \in [freq, f_r)$ which allows the workflow execution within the deadline is used. Note that only resources where the reduction in frequency leads to cost savings without exceeding the deadline are considered to be candidate resources to adjust their frequency (line 11). Starting with the most cost-efficient resource (line 12), 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 same procedure (lines 8-12) is repeated until there is no other candidate resource to adjust its configuration based on the current available frequency $freq$. After each iteration (when there are no other candidate resources) the cost of the new plan is computed to determine if reducing the frequency further also leads to overall cost savings. When the overall cost is reduced the algorithm continues to the next iteration, repeating the same procedure until either the minimum frequency is reached or no cost savings are achieved for any resource by lowering the frequency further. When the overall cost does not decrease, the new plan is rejected and the algorithm terminates.

Cost-based Stepwise Frequency Selection from Minimum Frequency (CSFS-Min). Same as with CSFS-Max, the CSFS-Min algorithm (Alg. 2) also works in two phases to initially produce a schedule of the workflow tasks onto the resources and then determine the frequencies to be used per resource. The algorithm initially maps the workflow tasks onto the resources using HEFT again, but, in contrast to CSFS-Max, assuming that resources operate at minimum frequency. Then, the algorithm iteratively increases the frequencies of the resources following the range of the available frequencies for as long as the deadline is not met or cost savings are achieved. The idea of the algorithm is that as resources operating at lower frequencies are charged less, using lower frequencies may lead to less expensive configurations. However, using the lowest possible frequency may not always be cost-efficient due to the impact of frequency reduction on execution time. Thus, gradually higher frequencies are used in an attempt to reduce the overall workflow cost.

Algorithm 2 Cost-based Stepwise Frequency Selection from Minimum Frequency.

Require: w : workflow, $curPlan$: HEFT plan at minimum frequency, $deadline$: user deadline

```

1: procedure FREQUENCYSELECTIONMIN( $w, curPlan$ )
2:   while  $freq < f_{max}$  do                                ▷ Starting with  $freq = f_{min}$ 
3:      $currentCost, currentMakespan$ : overall cost, makespan of  $curPlan$ 
4:      $freq+ = freqStep$                                     ▷ next avail. frequency
5:      $newPlan = curPlan$ 
6:      $candResources : \forall r \in newPlan$                     ▷ candidate resources
7:     while  $candResources$  not empty do
8:       for  $r \in newPlan$  do
9:         Compute  $makespanSavings_r$ 
10:      end for
11:       $candResources : \forall r \in newPlan: makespanSavings_r > 0$ 
12:      Remove  $r \in candResources$  with largest  $makespanSavings_r$ 
13:      Update task runtimes for each task  $t \in r$  for the new frequency
14:      Update start and finish times for each task  $t \in w$  in the plan ( $newPlan$ )
15:    end while
16:     $newCost, newMakespan$ : overall cost and makespan of  $newPlan$ 
17:    if  $newCost \geq currentCost \ \&\& \ currentMakespan < deadline$  then
18:      Reject  $newPlan$  and break
19:    end if                                ▷ deadline already met in  $curplan$ , cost increases in  $newplan$ 
20:    Accept plan ( $curPlan = newPlan$ )
21:  end while
22:  if  $newMakespan > deadline$  then  $curPlan = generate\ HEFT\ plan\ at\ f_{max}$ 
23:  end if
24: end procedure

```

The algorithm tries the next higher available frequency in each iteration as long as the deadline is not met or the overall workflow cost is reduced. Thus, a higher frequency is assigned to each resource when the provisioning time for the resource (which is equal to the workflow makespan), is reduced. The makespan savings of each resource (the decrease in makespan when assigning the higher frequency to the resource) are computed as:

$$makespanSavings_r = makespanCur - makespanNew, \quad (7)$$

where $makespanCur$ is the makespan resulting from operating the resource at the currently assigned frequency and $makespanNew$ the makespan achieved by increasing the frequency to the higher possible frequency in the current iteration. Starting from the resource with the highest makespan savings, the runtimes of its assigned tasks are updated for the higher selected frequency f and the slots of all the workflow tasks in the new plan are adjusted. The same procedure (lines 9-12) to find the next candidate resource and increase its assigned frequency is repeated until there is no other candidate resource to adjust its configuration based on the current available frequency $freq$. Candidate resources (line 11) include only those for which the increase in frequency results in a smaller makespan ($makespanSavings_r$). This is because when no reduction in makespan is achieved, lower frequencies which are charged less are preferred. At the

end of each iteration, the makespan and cost of the new plan are computed to determine if increasing the frequency further is possible. If the new plan exceeds the deadline or it results in no cost savings then it is rejected and the algorithm terminates (lines 15-18). In some cases with strict deadlines the algorithm may not generate a plan that allows the workflow to be completed within the deadline, as the initial conditions start with a plan that exceeds the deadline; this may require successive frequency adjustments to make the schedule meet the deadline. If the algorithm does not succeed, it reverts to computing a schedule at maximum frequency using HEFT (lines 19-20).

5 Experimental Evaluation and Results

In this section the performance of the two proposed algorithms is evaluated and compared using three different pricing models and three workflows with the system and application characteristics described below.

Methodology The simulator in [5], originally developed in [17], was used to implement and evaluate the two proposed cost-aware algorithms. Resources that operate at a range of frequencies between a minimum and maximum frequency, $f_{min}=1000\text{MHz}$ and $f_{max}=3000\text{MHz}$, with a frequency step (f_{step}) of 100MHz are assumed. Each resource corresponds to one-core VM while task runtimes when using maximum frequency are considered to be known. In order to compute the price charged for the provisioning of each resource, the parameters C_{min} and C_{dif} for each pricing model – the linear, super-linear and sublinear pricing models of Eq. 2, 3 and 4, respectively – are the following:

$$\begin{aligned} C_{min} &= \pounds 9.24 * 10^{-6}, \pounds 9.24 * 10^{-6}, \pounds 2.78 * 10^{-6}, \\ C_{dif} &= \pounds 3.33 * 10^{-6}, \pounds 4.44 * 10^{-6}, \pounds 1.2 * 10^{-5}. \end{aligned}$$

The values were chosen to approximate roughly the monthly charges of ElasticHosts for the provisioning of VMs, assuming time units in seconds. A network of 1Gbps was assumed to compute the communication costs between tasks assigned to different resources.

Synthetic data of three real scientific workflows, namely LIGO [15], SIPHT [16] and Montage [13], were used by the workflow generator in [29] to generate a workflow of 1000 tasks for each application and investigate the performance of the algorithms for applications with different characteristics. LIGO, a scientific workflow for the detection of gravitational waves in the universe, can be characterized as a data-intensive application with many CPU-intensive parallel jobs processing large amounts of data. SIPHT, a workflow that searches for sRNA encoding genes for bacterial replicons mainly consists of CPU-intensive jobs with low I/O utilization and can be characterized as a CPU-intensive application. Montage, a workflow that generates image mosaics of the sky, can be characterized as I/O intensive with low CPU utilization for most jobs. The average CPU utilization of the jobs was computed for each application based on the workflow profiling data in [12] and was used to compute the slowdown from the reduction in frequency for each application setting the parameter β (see Eq. 1) equal to 0.4 and 0.8 and 0.36 for LIGO, SIPHT and Montage respectively.

Results In the evaluation, HEFT [27] was used as a baseline algorithm to generate schedules with CPU resources running at maximum frequency, which are compared with the performance of our two algorithms. In order to assess the performance of the algorithms with reasonable deadlines that would allow to stretch the execution of the workflow tasks using low CPU frequencies, the user deadline was set equal to $1.05 * M_{f_{min}}$ (where $M_{f_{min}}$ is the makespan obtained by HEFT when all resources operate at minimum CPU frequency). This allows the evaluation of the two algorithms with a wide range of different CPU frequencies as opposed to what would be the case if strict deadlines were chosen where mostly high CPU frequencies would be feasible. The results obtained are presented in Fig. 4-12, and include plots for each pricing model (linear, superlinear and sublinear model) and workflow used (LIGO, SIPHT and Montage). The performance of the algorithms is compared in terms of the cost (given in £) and time (given in *secs*) required for the workflow execution. The plots of the workflow makespan also show the deadline used in each scenario on top of the HEFT bar. The number of provisioned resources ranged between 5 to 35, as shown on the x-axis in the results, to create scenarios with a varied resource utilization.

With *linear pricing*, where the provisioning cost of each resource changes at the same rate between each pair of subsequent frequencies, when using LIGO and Montage, CSFS-Max and CSFS-Min achieve similar performance in terms of execution time and user cost (Fig. 4 and 6). However, in the case of the CPU-intensive SIPHT (Fig. 5), overall high frequencies are more cost-efficient, as the penalty in workflow execution time is significant when using lower frequencies exceeding the reduction in price. More specifically, CSFS-Max achieves performance similar to HEFT in terms of cost and execution time as using high frequencies is more cost-efficient, while CSFS-Min leads to increased execution time resulting in higher cost, due to the different assignment of the tasks to the resources and the lower frequencies used.

With *superlinear pricing* (Fig. 7-9), price reduction from high-frequencies to subsequently lower frequencies is significant and gets smaller while moving to lower frequencies. This means that CSFS-Max will iterate and reach low frequencies where the cost savings exceed the penalty on execution time. Also, CSFS-Min increases the frequency of a resource when the reduction in execution time exceeds the increase in price. As overall cost does not decrease while approaching high frequencies, low frequencies are chosen. As a result, both algorithms achieve similar performance for all three workflows, stretching the workflow execution time compared with the baseline HEFT algorithm to achieve cost-efficient configurations. Still, the benefits are less profound with SIPHT, which is CPU-intensive.

Finally, with *sublinear pricing* (Fig. 10-12) scaling the frequencies iteratively starting from maximum frequency may lead to more expensive configurations, such as the case of SIPHT. This is because the provisioning cost of each resource does not decrease greatly with a small reduction in frequency while the frequencies are high. As a result, CSFS-Max fails to reach lower frequencies where the cost savings become significant exceeding the penalty in execution time.

Overall, the results show that both algorithms may explore cost-efficient frequencies in most scenarios of different pricing and workflow characteristics. As their initial conditions and iterative procedure differ, the two algorithms complement each other. Start-

Fig. 4: LIGO of 1000 tasks under linear pricing.

Fig. 5: SIPHT of 1000 tasks under linear pricing.

Fig. 6: MONTAGE of 1000 tasks under linear pricing.

Fig. 7: LIGO of 1000 tasks under superlinear pricing.

Fig. 8: SIPHT of 1000 tasks under superlinear pricing.

Fig. 9: MONTAGE of 1000 tasks under superlinear pricing.

Fig. 10: LIGO of 1000 tasks under sublinear pricing.

Fig. 11: SIPHT of 1000 tasks under sublinear pricing.

Fig. 12: MONTAGE of 1000 tasks under sublinear pricing.

ing from maximum frequency ensures that execution time is stretched only if makespan increases lead to satisfactory cost savings. On the other hand, starting from minimum frequency may lead to difficulties to reduce execution time and obtain sufficient cost savings. This is visible with SIPHT, a CPU-intensive application, where high frequencies are more suitable to use to achieve a low cost for linear or superlinear pricing. However, starting from the minimum frequency appears to be more suitable for scenarios under sublinear pricing where low frequencies may be preferred. Sublinear pricing is the case where the two algorithms result in the most significant cost savings.

6 Conclusion

This work considered the problem of cost-aware resource provisioning for the execution of deadline-constrained scientific workflows, proposing algorithms that select a separate CPU frequency for each provisioned resource to reduce the overall user cost. The evaluation of the algorithms suggests that the algorithms can be beneficial under different circumstances, particularly in the case of a sublinear pricing model. Also, the workflows that appear to benefit more are data-intensive workflows as execution time is affected less when using lower CPU frequencies. Future work could try to develop and assess different algorithms (not necessarily iterative) to solve the problem. One issue to investigate could be how to combine the strengths of both proposed algorithms in order to decide what a good initial schedule is. Finally, the performance of the proposed algorithms could be evaluated through real experiments.

References

1. Abrishami, S., Naghibzadeh, M., Epema, D.: Cost-driven Scheduling of Grid Workflows Using Partial Critical Paths. *IEEE Transactions on Parallel and Distributed Systems* 23(8), 1400–1414 (2012)
2. Byun, E.K., Kee, Y.S., Kim, J.S., Deelman, E., Maeng, S.: Bts: Resource capacity estimate for time-targeted science workflows. *Journal of Parallel and Distributed Computing* 71(6), 848–862 (2011)
3. Byun, E.K., Kee, Y.S., Kim, J.S., Maeng, S.: Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems* 27(8), 1011–1026 (2011)
4. Canon, L.C., Jeannot, E., Sakellariou, R., Zheng, W.: Comparative evaluation of the robustness of dag scheduling heuristics. In: *Grid Computing*. pp. 73–84. Springer (2008)
5. Cloud Workflow Simulator: Available:<https://github.com/malawski/cloudworkflowsimulator>
6. CloudSigma: Available:<https://www.cloudsigma.com/>
7. ElasticHosts: Available:<http://www.elastichosts.co.uk/>
8. Etinski, M., Corbalan, J., Labarta, J., Valero, M.: Optimizing job performance under a given power constraint in HPC centers. In: *Proceedings of the IGCC*. pp. 257–267. IEEE (2010)
9. Fard, H.M., Prodan, R., Barrionuevo, J.J.D., Fahringer, T.: 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 (CCGrid2012)*. pp. 300–309. IEEE (2012)
10. Hsu, C.H., Kremer, U.: The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. *ACM SIGPLAN Notices* 38(5), 38–48 (2003)

11. Huu, T.T., Montagnat, J.: 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. pp. 612–617. IEEE (2010)
12. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. *Future Generation Computer Systems* 29(3), 682–692 (2013)
13. Katz, D.S., Jacob, J.C., Deelman, E., Kesselman, C., Singh, G., Su, M.h., Berriman, G., Good, J., Laity, A., Prince, T.A.: 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). pp. 85–94. IEEE (2005)
14. Li, J., Su, S., Cheng, X., Huang, Q., Zhang, Z.: Cost-conscious scheduling for large graph processing in the cloud. In: Proceedings of the IEEE International Conference on High Performance Computing and Communications. pp. 808–813. IEEE (2011)
15. LIGO project, L.i.g.w.o.: <http://www.ligo.caltech.edu/>
16. Livny, J., Teonadi, H., Livny, M., Waldor, M.K.: High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one* 3(9), e3197 (2008)
17. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds. In: Proceedings of the International Conference on Supercomputing. pp. 10–16. IEEE (2012)
18. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: Proceedings of the 11th IEEE/ACM International Conference on Grid Computing. pp. 41–48. IEEE (2010)
19. Pietri, I., Sakellariou, R.: Cost-efficient provisioning of cloud resources priced by CPU frequency. In: Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing. pp. 483–484. IEEE (2014)
20. Prodan, R., Wiczorek, M.: Bi-Criteria Scheduling of Scientific Grid Workflows. *IEEE Transactions on Automation Science and Engineering* 7(2), 364–376 (2010)
21. Sakellariou, R., Zhao, H.: A hybrid heuristic for dag scheduling on heterogeneous systems. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium. p. 111. IEEE (2004)
22. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: *Integrated Research in GRID Computing*, pp. 189–202. Springer (2007)
23. Sharma, U., Shenoy, P., Sahu, S., Shaikh, A.: A cost-aware elasticity provisioning system for the cloud. In: 31st International Conference on Distributed Computing Systems (ICDCS). pp. 559–570. IEEE (2011)
24. Shi, W., Hong, B.: Towards Profitable Virtual Machine Placement in the Data Center. In: Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing. pp. 138–145. IEEE (2011)
25. Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., Wang, J.: Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing* 39, 177–188 (2013)
26. Taylor, I.J., Deelman, E., Gannon, D., Shields, M.: *Workflows for e-Science*. Springer (2007)
27. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13(3), 260–274 (2002)
28. Verma, A., Kaushal, S.: Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud. In: *Recent Advances in Engineering and Computational Sciences (RAECS)*. pp. 1–6. IEEE (2014)
29. Workflow Generator: Available:<https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator>