Cost-efficient Provisioning of Cloud Resources Priced by CPU Frequency

Ilia Pietri, Rizos Sakellariou
School of Computer Science, University of Manchester, U.K

***Abstract*—Cloud providers now offer resources that operate in a range of CPU frequencies giving users a large number of resource configurations to choose for their application needs. As higher CPU frequencies incur a higher monetary cost, users face the challenge of selecting the CPU frequencies that lead to cost-efficient configurations and strike a good balance between cost and performance. In this paper, an algorithm that achieves cost-aware provisioning by selecting different CPU frequencies for each resource in order to execute scientific workflows within a deadline is presented.**

I. Introduction and Problem Description

Cloud providers start offering resources whose CPUs can operate at different frequencies, allowing users to choose from a wide range of possible configurations. For example, ElasticHosts charge for CPU provisioning depending on the frequency chosen, allowing the selection from a range of 391 different frequencies for a monthly cost ranging from £4.32 to £172.80 [1]; faster VMs (operating at a higher frequency) cost more. With such a wide range of possibilities users face the challenge of choosing cost-efficient configurations for their needs. Assuming that users set a deadline for the completion of their applications, the paper suggests an algorithm to minimize overall cost by choosing as low frequencies for different resources as possible in order to complete the application by the deadline. Cost-efficient provisioning for applications with precedence constraints is the subject of related work [1]. The distinguishing feature of our algorithm is that it focuses on choosing the operating frequency of the resources to achieve cost-efficient configurations.

In our model, the user is interested in executing a scientific workflow by a given deadline on a number of resources. Resources are provisioned from the start until the completion of the application and CPU capacity is charged on the basis of the frequency chosen for each resource. For simplicity, we assume the cost of other characteristics (e.g., disk, storage, etc) is fixed and we focus on the issues related to the possible CPU frequencies that can be used. Scientific workflows are modelled as a Directed Acyclic Graph (DAG) to describe computational tasks (nodes) and their data dependencies (edges). Information on task runtimes at maximum frequency and data transfers is provided. Then, task runtime when running at a frequency f can be estimated by

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

[1] http://www.elastichosts.co.uk/

where $runtime_{t_{f_{max}}}$ is the runtime of the task t when operating at maximum frequency, f_{max}. The parameter β_t shows the impact of the frequency to runtime and takes values between 0 when the frequency does not impact job runtime and 1 for jobs with high CPU activity [2]. We assume homogeneous resources that can operate at a number of frequency states, regularly distributed (with step $freqStep$) between a minimum and maximum frequency. A task has exclusive control of the resource where it runs.

The user is charged for each resource according to the allocated frequency using a linear pricing model. The per unit of time cost, C_{f_r}, of resource r operating at frequency f_r is computed by

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

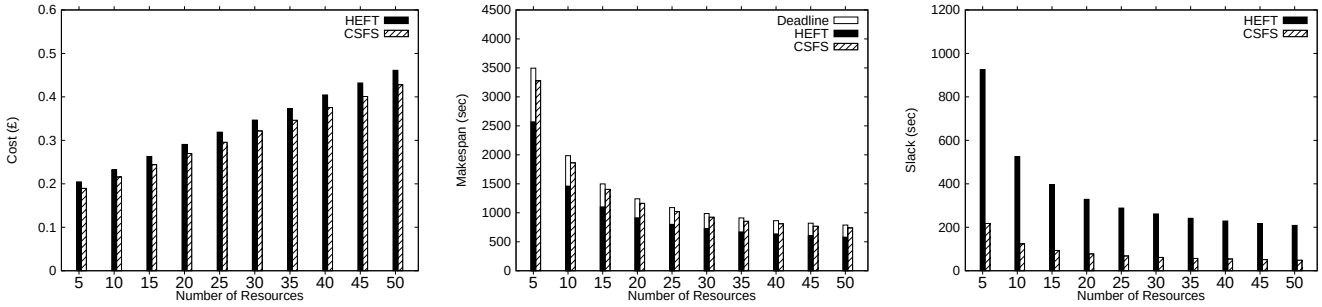
where C_{min} is the charge of the resource operating at minimum frequency, f_{min}, and C_{dif} is used to generate the price at each frequency. The overall cost for the workflow execution is computed from the cost of running each resource r at its assigned frequency f_r for the whole scheduling period.

II. The CSFS Algorithm

The Cost-based Stepwise Frequency Selection (CSFS) algorithm determines statically the frequencies that can be used for different resources to reduce the overall workflow cost and complete execution by a deadline. The algorithm

Algorithm 1
Cost-based Stepwise Frequency Selection - CSFS.

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



(a) Application Cost. (b) Execution time. (c) Slack until the deadline.

Figure 1: Montage workflow with 1000 tasks.

(see Alg. 1) roughly resembles the Energy-aware Stepwise Frequency Scaling Algorithm (ESFS) in [3], but it considers user cost to select a frequency per resource, as opposed to scaling the operating frequency of each task for energy savings.

The algorithm assumes that an initial mapping of tasks onto the available resources using HEFT [4] is built, with resources operating at maximum speed (highest frequency). Then, it iteratively lowers the frequency (step by step following the range of available frequencies), checks if cost savings for each resource can be obtained by using a lower frequency than the currently assigned to it (lines 7-13) and then assesses the impact that the cost savings from using this frequency have on the overall cost to execute the workflow (lines 14-17). The frequency of each resource is reduced when the cost of using the resource decreases and the resulting schedule is within the deadline. The loop continues until the minimum frequency is reached or further lowering the frequency does not bring cost savings for any resource.

III. Experimental Evaluation

The simulator in [5] was used to compare the performance of CSFS with the performance of HEFT [4], a standard workflow scheduling algorithm. Homogeneous resources that operate in steps of 100 MHz in the range of 1000-3000 MHz are assumed, connected by a 1 Gbps network. The parameters for the pricing model of Eq. 2 were set equal to $C_{min} = £9.24 * 10^{-6}$ and $C_{dif} = £3.33 * 10^{-6}$ based on the monthly charges of ElasticHosts for VMs, assuming time units in seconds. Synthetic data for Montage [6], a scientific application that generates image mosaics of the sky, were used to generate a workflow of 1000 tasks [7]. The parameter β was set equal to 0.36, the average CPU utilization of the jobs based on profiling data in [6]. Finally, the deadline was set equal to the mean of the makespans obtained by HEFT when the tasks are scheduled using the maximum and minimum frequency. This makes it possible to use a deadline which is not too tight (meaning resources running at the highest frequency are used) or too far ahead in the future (meaning that the cheapest resources running at the lowest frequency can still meet the deadline).

The number of resources used in each experiment (x-axis) was varied to create different utilization scenarios. Results for the cost required for the workflow execution ($£$), the achieved makespan ($secs$) and the slack time ($secs$), computed as the difference between the finish time of the workflow execution and the deadline, are included. The proposed algorithm, CSFS, reduces the cost compared with the initial schedule of HEFT (Fig. 1a), by lowering the frequency of the resources to produce cost-efficient configurations. As a result, CSFS results in longer execution time (Fig. 1b) but always within the deadline, making better use of the slack time (Fig. 1c).

IV. Conclusion

This paper considered the problem of cost-aware resource configuration for deadline-constrained scientific workflows. An algorithm that selects a CPU frequency for each provisioned resource to reduce the overall user cost without missing the deadline has been proposed and evaluated.

References

[1] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Computing*, vol. 39, pp. 177–188, 2013.

[2] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Optimizing job performance under a given power constraint in HPC centers," in *Proceedings of the IGCC*, 2010, pp. 257–267.

[3] I. Pietri and R. Sakellariou, "Energy-aware workflow scheduling using frequency scaling," in *Proceedings of the 43rd ICPPW*, 2014.

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE TPDS*, vol. 13, no. 3, pp. 260–274, 2002.

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

[6] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *FGCS*, vol. 29, no. 3, pp. 682–692, 2013.

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