Adaptive Multi-level Workflow Scheduling with Uncertain Task Estimates

Tomasz Dziok, Kamil Figiela, and Maciej Malawski(✉)

Department of Computer Science, AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{kfigiela,malawski}@agh.edu.pl

Abstract. Scheduling of scientific workflows in IaaS clouds with pay-per-use pricing model and multiple types of virtual machines is an important challenge. Most static scheduling algorithms assume that the estimates of task runtimes are known in advance, while in reality the actual runtime may vary. To address this problem, we propose an adaptive scheduling algorithm for deadline constrained workflows consisting of multiple levels. The algorithm produces a global approximate plan for the whole workflow in a first phase, and a local detailed schedule for the current level of the workflow. By applying this procedure iteratively after each level completes, the algorithm is able to adjust to the runtime variation. For each phase we propose optimization models that are solved using Mixed Integer Programming (MIP) method. The preliminary simulation results using data from Amazon infrastructure, and both synthetic and Montage workflows, show that the adaptive approach has advantages over a static one.

Keywords: Cloud · Workflow · Scheduling · Optimization · Adaptive algorithm

1 Introduction

Scientific workflow is a widely accepted method for automation of complex computational processes on distributed computing infrastructures, including IaaS clouds [7]. When using clouds and their pay-per-use pricing model with multiple types of virtual machine (VM) resources, usually called instances, the problem of scheduling and cost optimization becomes a challenge. The specific problem we address in this paper is that most static scheduling algorithms assume that the estimates of task runtimes are known in advance, while in reality these estimates may be inaccurate. These discrepancies may be a result of inherent uncertainty in performance models of the application, or may be caused by unexpected dynamic behavior of the infrastructure. On the other hand, dynamic scheduling approaches that adapt to such uncertainties cannot be easily used for scheduling

M. Malawski—This work is partially supported by EU FP7-ICT project PaaSage (317715), Polish grant 3033/7PR/2014/2 and AGH grant 11.11.230.124.

R. Wyrzykowski et al. (Eds.): PPAM 2015, Part II, LNCS 9574, pp. 90–100, 2016.
DOI: 10.1007/978-3-319-32152-3_9

under deadline or budget constraints, since meeting a constraint requires some form of advance planning based on estimates.

In this paper, we propose an adaptive scheduling algorithm for deadline constrained workflows that consist of multiple levels. Such levels are present in real scientific workflows and they often have up to 1 000 000 tasks [7,13]. The main idea behind the algorithm is to produce a global approximate plan for the whole workflow in a first phase, and a local detailed schedule for the current level of the workflow. The algorithm is then invoked iteratively after each level completes the execution, in this way being able to adjust to the runtime variation from the estimated execution times. Another advantage of this approach is that we can reduce the complexity of scheduling of the whole workflow by reducing it into two smaller problems that can be solved using Mixed Integer Programming (MIP). The algorithm has been evaluated by simulation using data from Amazon infrastructure and workflows from Pegasus Workflow Gallery [13].

This paper is organized as follows: in Sect. 2 we discuss other scheduling models and algorithms for workflows. Section 3 contains detailed description of the algorithm proposed in this paper, and its illustration on a simple example is given in Sect. 4. In Sect. 5 we outline the optimization models used. Then in Sect. 6 we show results for real workflows. Finally, in Sect. 7 we present conclusions and future work.

2 Related Work

Mathematical programming has been applied to the problem of workflow scheduling in clouds. The model presented in [12] is applied to scheduling small-scale workflows on hybrid clouds using time discretization. Large-scale bag-of-task applications on hybrid clouds are addressed in [4]. The cloud bursting scenario described in [3], where a private cloud is combined with a public one, also addresses workflows. None of these approaches addresses the problem of inaccurate estimates of actual task runtimes.

Adaptive approach is known from engineering systems [1]. Dynamic algorithms for workflow scheduling in clouds have been proposed e.g. in [17], where they assume the dynamic stream of workflows. In [9] the goal is to minimize makespan and monetary cost, assuming an auction model, which differs from our approach where we assume a cloud pricing model of Amazon EC2.

In our earlier work [14], we also used the MIP approach to schedule multi-level workflows, but the dynamic nature of cloud is not considered. We have also analyzed the impact of uncertainties of runtime estimations on the quality of scheduling for bag-of-task in [15] and workflow ensembles in [16], with the conclusion that these uncertainties cannot be always neglected.

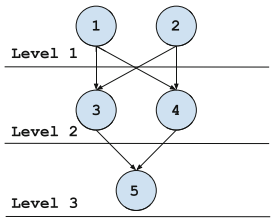


Fig. 1. DAG example

Task estimation for workflow scheduling is a non-trivial problem, but several approaches exist,

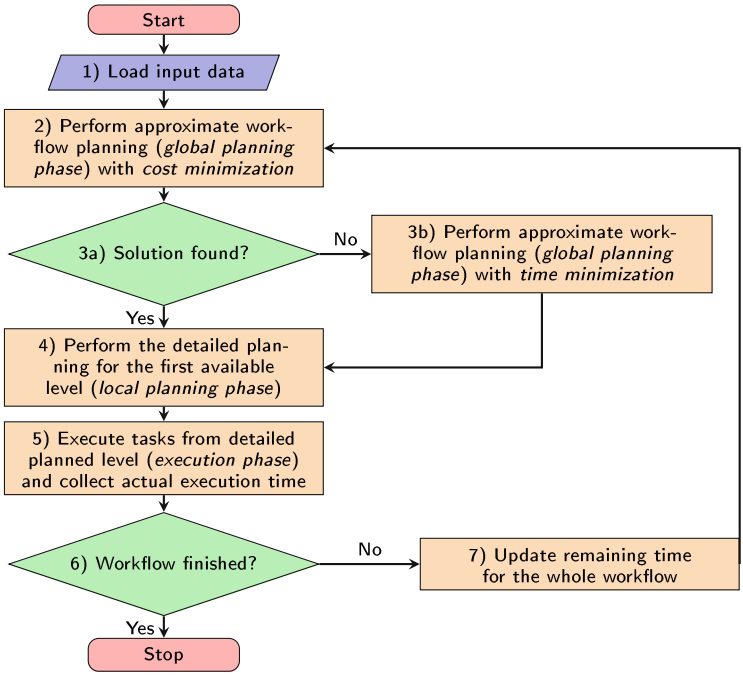


Fig. 2. High level flow of scheduling algorithm.

e.g. those based on stochastic modeling and workflow reductions [5]. It is also possible to create performance models to estimate workflow execution time using application and system parameters, as proposed in [18]. The error of these estimates is less than 20 % for most cases, which gives a hint on the size of possible uncertainties.

3 Adaptive Scheduling Algorithm

Our algorithm provides an adaptive method for optimizing cost of workflow execution in IaaS clouds, under a deadline constraint. We assume that the workflow tasks can be divided by their levels, where a level of a task is a length of the longest path from an entry node. Tasks from one level can have different estimates of execution time. It can be considered as a hybrid between static and dynamic scheduling algorithms.

The algorithm requires: (a) workflow (see Fig. 1) represented as directed acyclic graph (DAG), where nodes represent tasks and edges dependencies between them; (b) information about available infrastructure, i.e. the performance and cost of available VM instance types; and (c) global deadline for the whole workflow. We assume that (a) all tasks in each level are independent and can be executed in parallel on multiple VMs; (b) each VM has price per hour

and a performance metric called CCU (which is a result of a benchmark, as in Cloud Harmony Compute Units [6]); (c) each task has estimated size which is execution time on a VM with performance of 1 CCU; (d) tasks in one level could have different estimated size; and (e) execution time of a task on given VM is inversely proportional to VM performance expressed in CCU.

The objective of the algorithm is to minimize the execution cost under a deadline constraint. The algorithm is run before each level of tasks begins its execution. Each time it consists of two phases. In the first *global planning phase*, the algorithm uses an approximation that tasks in each level are uniform, and finds assignments between the tasks and VMs for the whole workflow. In the second *local planning phase*, a detailed plan is prepared for the closest level of individual tasks. After a level completes, the algorithm takes into account the real execution time of already completed tasks, and based on that updates the remaining time. Thanks to that it is able to adjust to differences between an estimated and actual execution time.

The algorithm is shown in Fig. 2, and consists of the following steps.

1. First, the information about workflow, available infrastructure (list of VMs) and global deadline are loaded.
2. In this step (*global planning phase*) algorithm assigns VMs to levels. For each level, we calculate average estimated task execution time and we pass it as input. The aim of this optimization is to find assignments between VMs and levels with minimal total cost under global deadline. As a result, the algorithm returns information which VMs are assigned to each level and also how many tasks should be executed on each VM. It also returns estimated execution time and cost for levels.
3. If the solver does not find a solution, the optimization is run again without deadline constraint, but with time minimization as an objective. This may be the case when the deadline is too short. We then fallback to minimization of deadline overrun and we ignore the cost objective.
4. Next, we perform *local planning phase* that assigns individual tasks to VMs in the current level. It uses the results from step 2 as an input: VMs assigned to this level and number of tasks which should be executed on each VM. The objective of optimization is to minimize the total execution time. Total cost is not taken into account, because the VMs are already chosen and the estimated execution time for each one is known – so the cost does not change. As a result the algorithm returns information on which VM task will be executed.
5. Then we execute tasks on VMs assigned in *local planning phase* and collect the actual task execution time. Tasks may be executed on real VMs instances or in a cloud simulator (which allows to test many scenarios easily).
6. The algorithm finishes if there are no remaining levels to be scheduled.
7. We update remaining total time with actual execution time and perform planning for remaining part of the workflow, repeating process from step 2.

4 Illustrative Example

To illustrate the operation of our algorithm, we prepared an example using the simple workflow from Fig. 1. The input is provided in Table 1. The workflow consists of 3 levels, so the algorithm is executed in three iterations, as shown in Table 2. The resulting execution times and costs in all iterations are presented and commented in Fig. 3.

Table 1. Example input to the algorithm: estimated task sizes, VM performance and costs for the workflow shown in Fig. 1. We assume the global deadline is 15.

TaskID	T1	T2	T3	T4	T5
Est. Task Size	22	18	10	10	20

VM ID	Performance (CCU)	Cost per Time Unit
A	5	10
B	10	25

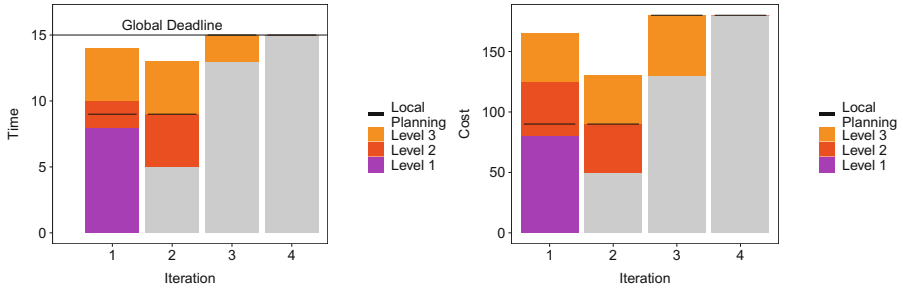


Fig. 3. Execution time and cost of the algorithm, shown level by level. In the first iteration, the global planning phase estimates the completion time of level 1 is 8 (purple bar) and the local planning estimates it to be 9 (solid line). In iteration 2, it turns out that the level L1 finished at time 5 (grey bar). Both global and local planning for level 2 (red bar and solid line) predict the finish time for time 9. The actual execution of level 2 completes in time 13 (grey bar), so in iteration 3 both global and local phases plan the execution of level 3 (orange bar) to complete just within the deadline). The execution in iteration 4 shows that the level 3 actually completed as planned (Color figure online).

5 Optimization Models

We use three optimization models in the algorithm: the first one for global planning phase, the second one in the case when deadline cannot be met, and the third one for the local planning. Since the domain is discrete, each model belongs to a mixed-integer programming (MIP) class. In all three models we assume for simplicity that VMs start immediately and have no latency. Thanks to that the problems are solved quicker. On the other hand, we assume that all possible delays are included in the error of estimates, which is taken into account in step 7 of the algorithm. Here we outline the main features of the models, and for the details we refer to the source code in the public repository [8].

Table 2. Planning and execution flow for illustrative example. The assignments of tasks to VMs change whenever the actual execution time differs from the estimated one.

Level	# Tasks	Avg. Task Exec. Time	Planned Time	Planned Cost	Assigned VMs (number of tasks)
L1	2	20	8	80	A (2)
L2	2	10	2	45	A (1), B (1)
L3	1	20	4	40	A (1)

Task	VM	Planned Time	Planned Cost	Actual Time	Actual Cost
T1	A	5	50	3	30
T2	A	4	40	2	20

(a) 1^{st} iteration, *global planning* (left): estimated cost of executing the workflow is 165 and the total time is 14. Algorithm plans to use all the available time, selects cheaper instance and minimizes the total cost. *Local planning* and execution (right): task runtimes from level L1 were overestimated, the remaining time is 10.

Level	# Tasks	Avg. Task Exec. Time	Planned Time	Planned Cost	Assigned VMs (number of tasks)
L2	2	10	4	40	A (2)
L3	1	20	4	40	A (1)

Task	VM	Planned Time	Planned Cost	Actual Time	Actual Cost
T3	A	2	20	4	40
T4	A	2	20	4	40

(b) 2^{nd} iteration, *global planning* (left): Due to more time than expected, the algorithm assigned all the tasks to the cheapest VM – to minimize the total cost. *Local planning* and execution (right): tasks from level L2 were underestimated. Remaining time is 2, current total cost is 130.

Level	# Tasks	Avg. Task Exec. Time	Planned Time	Planned Cost	Assigned VMs (number of tasks)
L3	1	20	2	50	B (1)

Task	VM	Planned Time	Planned Cost	Actual Time	Actual Cost
T5	B	2	50	2	50

(c) 3^{rd} iteration, *global planning* (left): the algorithm selects the more powerful VM B to keep the deadline constraint. *Local planning* and execution (right): total time is 15, total cost is 180 and the workflow completed in the given deadline.

Model used in *global planning phase* assigns VMs and sub-deadlines to each level, but instead of scheduling individual tasks, it uses an approximation of average task runtimes. For each level, it calculates an average task size, and based on this, an estimated cost of executing its tasks on a given VM. As a result, it is known which VMs should be used for each level and how many tasks should be executed on selected VM. The objective is to minimize total cost of the whole workflow execution.

Input to this approximate planning is defined with the following data: m is number of VMs, n is number of levels, V is a set of VMs, L is a set of levels, d is global deadline, L_l is number of tasks in level l, $T^a_{l,v}$ is average estimated execution time of task from level l on VM v, p_v is cost of running VM v for one time unit, $C_{l,v} = p_v T^a_{l,v}$ is average estimated cost of executing task from level l on VM v.

The search space is defined with the following variables: $A_{l,v}$ is binary matrix which tells if VM v will execute at least one task from level l, $Q_{l,v}$ is integer matrix which tells how many tasks from level l will be executed on VM v, T^e_l is vector of real numbers which stores execution time for level l (estimated sub-deadlines), $T^v_{l,v}$ is matrix which stores execution time for VM v on level l. $A_{l,v}$ is used as an auxiliary variable to simplify defining constraints.

The objective is to minimize total cost: Minimize: $\sum_{l}^{L}\sum_{v}^{V} C_{l,v} * Q_{l,v}$. We constrain the search space to keep the total execution time below the deadline, to divide the deadline into sub-deadlines and to enforce them, and to ensure that all the tasks from all the levels are executed.

Model used in *global planning phase* when deadline cannot be met is used when searching for solution using the first model fails. It can happen e.g. when real execution time of previous level takes much more time than expected. Comparing to the previous model, the algorithm ignores global deadline constraint and the objective function minimizes total time of workflow execution: Minimize: $\sum_{l}^{L} T_l^e$.

Model used in *local planning phase* assigns VMs to each task from a single level. The goal is to minimize time of level execution, which is equal to the time of the longest working VM. The input to this optimization problem is defined with the following data: m is number of VMs, k is number of tasks in current level, K is a set of tasks, V is a set of VMs (only VMs assigned to current level – results from *global planning phase*), $T^e_{k,v}$ is an estimated execution time of task k on VM v, N_v is a number of tasks which will be executed on VM v (results from *global planning phase*).

Search space is defined with the following variables: $A_{k,v}$ is binary matrix which tells if task k will be executed on VM v, T^r_v is vector of real numbers which tells how long does each VM v work, w is helper variable which stores the longest working time for VMs from V.

The objective is to minimize time of the longest working VM: Minimize: $max(T^r_v | v \in V)$ that is implemented as Minimize: w. We constrain the search space to ensure that all the tasks are executed, to assign given number of tasks on each VM, and to assign the correct value to w which is the longest working VM.

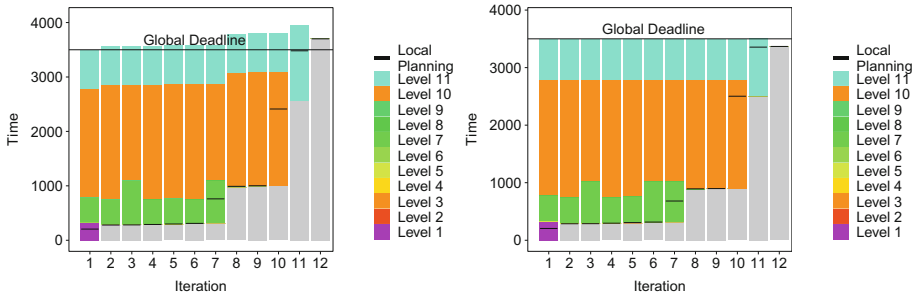
Implementation of Algorithm and Models. Optimization models are implemented in CMPL modeling language [19]. As a solver we use CBC [11]. Input data is loaded from DAG files (workflows) and JSON files (infrastructure). The simulator which executes the tasks and introduces the runtime variations is implemented in Java. Source code (including optimization models) is available in the repository [8].

6 Evaluation Using Synthetic Workflows

For evaluation of the algorithm we implemented a simple simulator. Its goal is to execute one level of tasks on the assigned VMs and to introduce the runtime variation of task execution times to simulate the behavior of the real infrastructure.

We present here the results of our adaptive algorithm obtained using Montage workflow [13] representing astronomical image processing, consisting of 5000 tasks. As estimates of task sizes we used data from the logs of our earlier runs performed on Amazon EC2 [10]. We used the m3.large as a reference VM type

and for performance estimation of other instance types we used the ECU value as provided by Amazon [2]. As the error of estimates we introduced a normal distribution with the standard deviation of 0.25. Since the real Montage workflow consists of very small tasks (having execution time in the order of seconds), we artificially extended them by multiplying their execution time by 3600. The deadline was set to 3500 time units (hours).



(a) Static Algorithm (b) Adaptive Algorithm

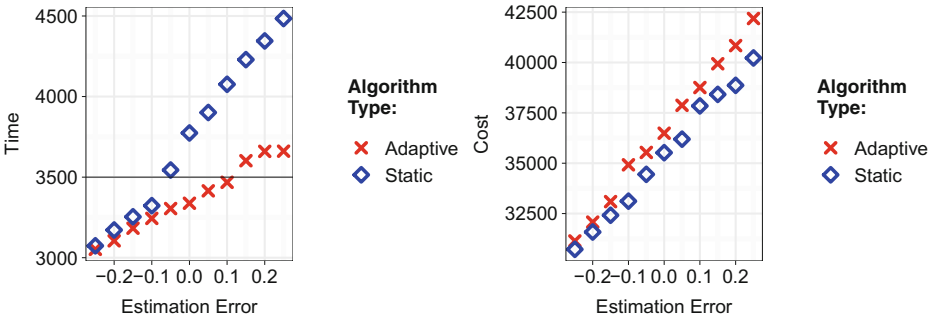
Fig. 4. Execution time plot for Montage 5000 workflow with random errors of estimates.

We compared our adaptive algorithm to its static scheduling variant as a baseline. The static scheduling works in the same way as our algorithm, but it plans all the levels in advance. This means it does not update the global and local planning phases after execution of each level, so it does not adjust to the runtime variations.

Figure 4 shows the results of the static and adaptive scheduling algorithms, presented in the same convention as in the illustrative example (Fig. 3). We can observe that in the plot (b) the adaptive algorithm adjusts to the actual execution time after each level, while the static algorithm (a) does not, which leads to the deadline overrun.

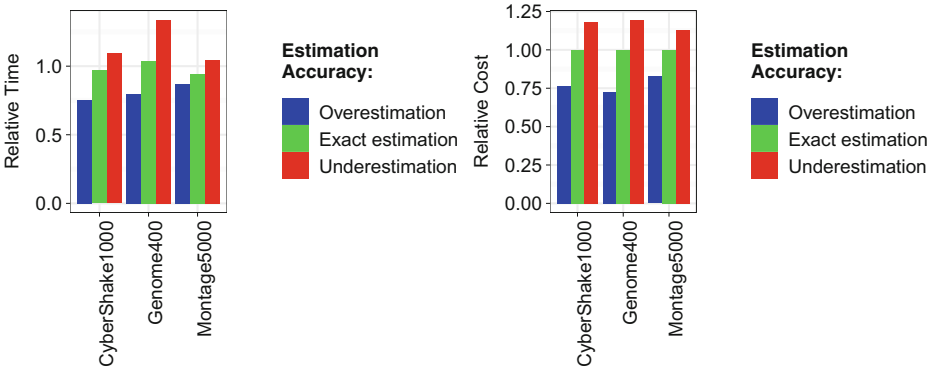
Figure 5 presents how the completion time and total cost depend on the varying estimation error μ. The errors were generated using the normal distribution with the standard deviation of 0.25 and the mean of μ, with μ from -0.25 (overestimation) to 0.25 (underestimation). In plot (a) we observe that our adaptive algorithm succeeds to meet the deadline in more cases than the static algorithm. Even for the largest error ($\mu = 0.25$) the deadline overrun is only 5 %, while for the static algorithm it is over 25 %. On the other hand, plot (b) shows that the adaptation costs more, i.e. in most cases the cost is higher for adaptive algorithm, but never more than by 5 %. This is explained by the need to choose more expensive VMs to complete the workflow before the deadline.

In addition to Montage, we tested our algorithms using other workflows from the gallery [13]. Generally, we observed similar behavior as in the case of Montage. Sample results are shown in Fig. 6, where overestimation and underestimation represent error distribution shifted by -0.25 and 0.25, respectively. Relative



(a) Workflow execution time. (b) Workflow execution cost.

Fig. 5. Workflow execution time/cost depending on the estimation error.



(a) Workflow execution time. (b) Workflow execution cost.

Fig. 6. Plots with normalized execution time/cost for other workflows.

execution time is normalized to the deadline, while the relative cost is normalized to the cost of execution with exact estimates (errors with $\mu = 0$ and standard deviation of 0.25). The deadline overrun for large errors is caused by the fact that when the task runtimes are underestimated in the final level, the algorithm cannot adjust to them. Improving the algorithm would require adding a learning capability to predict the estimation error based on previous levels, which will be the subject of future work.

7 Conclusions and Future Work

In this paper we presented the adaptive algorithm for scheduling workflows in clouds with inaccurate estimates of run times. The preliminary evaluation results have shown that the implemented algorithm works as designed, and is able to meet the given deadline while minimizing the cost.

The algorithm adapts to the actual situation at runtime: when tasks execute quicker than estimated – the algorithm selects slower (and cheaper) VMs, and minimizes the total cost. When tasks execute slower than estimated – the algorithm selects faster (and more expensive) VMs, which increases total cost, but allows not exceeding the deadline for the whole workflow. When deadline is exceeded (or it is not possible to plan execution under deadline) then the algorithm minimizes the total time regardless of cost. When estimated execution time for tasks from the same levels has a big variation, then there are visible differences between estimated time in *global planning phase* and *local planning phase*. When execution of tasks is longer is final levels (which is the worst case scenario) then the total cost increases, but this is general problem for all adaptive scheduling algorithms.

During implementation and evaluation we found out a few ways that could be enhanced in future work. They include improvement of pricing in optimization models by e.g. reusing already assigned VMs, extending models with data transfer time and cost, or splitting levels with many tasks on smaller ones on 'logic' independent levels. It would be also interesting to improve task estimation (i.e. take into account multi-core CPUs) or use machine learning in estimating task execution time. After more systematic testing, we plan to use this algorithm as a part of engine to executing workflows in computing clouds.

References

1. Abdelzaher, T., Diao, Y., Hellerstein, J.L., Lu, C., Zhu, X.: Introduction to control theory and its application to computing systems. In: Liu, Z., Xia, C.H. (eds.) Performance Modeling and Engineering, pp. 185–215. Springer, Heidelberg (2008)
2. Amazon: AWS pricing (2015). http://aws.amazon.com/ec2/pricing/
3. Bittencourt, L.F., Madeira, E.R.M.: Hcoc: A cost optimization algorithm for workflow scheduling in hybrid clouds. J. Internet Serv. Appl. **2**(3), 207–227 (2011)
4. den Bossche, R.V., Vanmechelen, K., Broeckhove, J.: Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Gener. Comput. Syst. **29**(4), 973–985 (2013)
5. Chirkin, A.M., Belloum, A.S.Z., Kovalchuk, S.V., Makkes, M.X.: Execution time estimation for workflow scheduling. In: 2014 9th Workshop on Workflows in Support of Large-Scale Science, pp. 1–10. IEEE, November 2014
6. CloudHarmony: What is ECU? CPU benchmarking in Cloud (2010). http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html
7. Deelman, E., et al.: Pegasus, a workflow management system for science automation. Future Gener. Comput. Syst. **46**, 17–35 (2015)
8. Dziok, T.: Repository with optimization models (2015). https://bitbucket.org/tdziok/mgr-cloudplanner
9. Fard, H.M., Prodan, R., Fahringer, T.: A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1203–1212 (2013)
10. Figiela, K., Malawski, M.: Modeling, optimization and performance evaluation of scientific workflows in clouds. In: 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, p. 280. IEEE, December 2014

11. Forrest, J.: Cbc (coin-or branch and cut) open-source mixed integer programming-solver (2012). https://projects.coin-or.org/Cbc
12. Genez, T.A.L., Bittencourt, L.F., Madeira, E.R.M.: Using time discretization to schedule scientific workflows in multiple cloud providers. In: 2013 IEEE Sixth International Conference on Cloud Computing, pp. 123–130. IEEE, June 2013
13. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Gener. Comput. Syst. **29**(3), 682–692 (2013)
14. Malawski, M., Figiela, K., Bubak, M., Deelman, E., Nabrzyski, J.: Scheduling Multilevel Deadline-Constrained Scientific Workflows on Clouds Based on Cost Optimization. Scientific Programming, New York (2015)
15. Malawski, M., Figiela, K., Nabrzyski, J.: Cost minimization for computational applications on hybrid cloud infrastructures. Future Gener. Comput. Syst. **29**(7), 1786–1794 (2013)
16. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Algorithms for cost and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. Future Gener. Comput. Syst. **48**, 1–18 (2015)
17. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: SC 2011. SC 2011, ACM, Seattle, Washington (2011)
18. Pietri, I., Juve, G., Deelman, E., Sakellariou, R.: 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, pp. 11–19. WORKS 2014, IEEE Press, Piscataway, NJ, USA (2014)
19. Steglich, M.: CMPL (Coin mathematical programming language) (2015). https://projects.coin-or.org/Cmpl