Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 680271, 13 pages
http://dx.doi.org/10.1155/2015/680271

Research Article

Scheduling Multilevel Deadline-Constrained Scientific Workflows on Clouds Based on Cost Optimization

Maciej Malawski,[1] Kamil Figiela,[1] Marian Bubak,[1,2] Ewa Deelman,[3] and Jarek Nabrzyski[4]

[1] *Department of Computer Science, AGH University of Science and Technology, Aleja Mickiewicza 30, 30-059 Kraków, Poland*
[2] *ACC CYFRONET AGH, Ulica Nawojki 11, 30-950 Kraków, Poland*
[3] *USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA*
[4] *Center for Research Computing, University of Notre Dame, Notre Dame, IN 46556, USA*

Correspondence should be addressed to Kamil Figiela; kfigiela@agh.edu.pl

Received 15 May 2014; Accepted 22 November 2014

Academic Editor: Roman Wyrzykowski

This paper presents a cost optimization model for scheduling scientific workflows on IaaS clouds such as Amazon EC2 or RackSpace. We assume multiple IaaS clouds with heterogeneous virtual machine instances, with limited number of instances per cloud and hourly billing. Input and output data are stored on a cloud object store such as Amazon S3. Applications are scientific workflows modeled as DAGs as in the Pegasus Workflow Management System. We assume that tasks in the workflows are grouped into levels of identical tasks. Our model is specified using mathematical programming languages (AMPL and CMPL) and allows us to minimize the cost of workflow execution under deadline constraints. We present results obtained using our model and the benchmark workflows representing real scientific applications in a variety of domains. The data used for evaluation come from the synthetic workflows and from general purpose cloud benchmarks, as well as from the data measured in our own experiments with Montage, an astronomical application, executed on Amazon EC2 cloud. We indicate how this model can be used for scenarios that require resource planning for scientific workflows and their ensembles.

1. Introduction

Today, science requires processing of large amounts of data and use of hosted services for compute-intensive tasks [1]. Cloud services are used not only to provide resources, but also for hosting scientific datasets, as in the case of AWS public datasets [2]. Scientific applications that run on these clouds often have the structure of workflows or workflow ensembles that are groups of interrelated workflows [3]. Infrastructure as a service (IaaS) cloud providers offer services where virtual machine instances differ in performance and price [4]. Planning computational experiments requires optimization decisions that take into account both execution time and resource cost.

Research presented in this paper can be seen as a step towards developing a "cloud resource calculator" for scientific applications in the hosted science model [1]. Specifically, we address the cost optimization problem of large-scale scientific workflows running on multiple heterogeneous clouds, using mathematical modeling with AMPL [5] and CMPL [6], and mixed integer programming. This approach allows us to describe the model mathematically and use a set of available optimization solvers. On the other hand, an attempt to apply this method to the general problem of scheduling large-scale workflows on heterogeneous cloud resources would be impractical due to the problem complexity and therefore simplified models need to be analyzed. In our previous work [7], we used a similar technique to solve the problem where the application consists of tasks that either are identical or vary in size within a small range. As observed in [8, 9], large-scale scientific workflows often consist of multiple parallel stages or levels, each of which has a structure of set of tasks; that is, the tasks in each level are similar and independent of each other. In the case of large workflows, when the number of tasks in the level is high, it becomes more practical to optimize the execution of the whole level instead of looking

at each task individually, as many scheduling algorithms do [10]. Therefore, in this paper, we extend our model to deal with applications that are workflows represented as DAGs consisting of levels of uniform tasks.

The main contributions of this paper are summarized as follows.

(i) We define the problem of workflow scheduling on clouds as a cost optimization problem of assigning levels of tasks to virtual machine instances, under a deadline constraint.

(ii) We specify the application model, infrastructure model, and the scheduling model as mixed integer programming (MIP) problems using AMPL and CMPL modeling languages.

(iii) We discuss the alternative scheduling models for coarse-grained and fine-grained tasks.

(iv) We evaluate the models using infrastructure performance data: one obtained from CloudHarmony benchmarks, and the one based on our own experiments with Montage workflows on Amazon EC2 cloud.

This paper is an extension of our earlier conference publication [11]. The most important extension is a new scheduling model dedicated to fine-grained workflows with short deadlines. Moreover, for evaluation, we use more detailed cloud benchmark dataset, based on our recent experiments with Montage workflow on Amazon EC2.

After outlining the related work in Section 2, we introduce our methodology in Section 3. We describe the application and infrastructure model in Section 4. In Section 5, we provide the mathematical formulation of the problem, including the application model, the infrastructure model, and the scheduling models for coarse-grained and fine-grained workflows. Section 6 describes the datasets used for evaluation of our models. Finally, Section 7 describes the evaluation of our models on a set of benchmark workflows, while Section 8 gives conclusions and future work.

2. Related Work in Cloud Workflow Scheduling

Our work is related to heuristic algorithms for workflow scheduling on IaaS clouds. In [12], the model assumes that infrastructure is provided by only one provider. The cloud-targeted autoscaling solution [10] considers dynamic and unpredictable workloads containing workflows. In [13], a multiobjective list-based method for workflow scheduling (MOHEFT) is proposed and evaluated. The solution presented in [14] focuses on cloud bursting scenario, where a private cloud is combined with a public one, and the goal is to minimize the cost while maintaining the workflow deadline. Our work is different from these approaches in two aspects. First, in our infrastructure model we assume multiple heterogeneous clouds with object storage attached to them, instead of individual machines with peer-to-peer data transfers between them. Moreover, rather than scheduling each task individually, our method proposes a global optimization of placement of workflow tasks and data.

The deadline-constrained cost optimization of scientific workloads on heterogeneous IaaS described in [15] addresses multiple providers and data transfers between them, where the application is a set of tasks. The global cost minimization problem on clouds addressed in [16] focuses on data transfer costs and does not address workflows. Other approaches presented in [17, 18] consider unpredictable dynamic workloads on IaaS clouds and optimize the objectives, such as cost, runtime, or utility function, by autoscaling the resource pool at runtime.

Pipelined workflows consisting of stages are addressed in [19]. The processing model is a data flow and multiple instances of the same workflow are executed on the same set of cloud resources, whereas in our approach we focus on cost optimization instead of meeting the QoS constraints.

Integer linear programming (ILP) method is applied to scheduling workflows on hybrid clouds in [20]. The objective is to minimize monetary cost under a deadline constraint. The scheduler uses varying discretization of the schedule timeline to reduce the complexity of the problem so that the employed CPLEX solver can find acceptable solutions within a 10-minute limit. The evaluation, however, is performed on the Montage and random fork-join workflows of 30 tasks with randomly chosen runtimes, while we focus on larger scale workflows and we address the complexity by grouping tasks into levels.

3. Methodology Based on Mathematical Optimization

The core of our methodology (see Figure 1) is to use mathematical modeling languages that can be coupled with a set of solvers dedicated to linear, nonlinear, or mixed integer programming problems. As modeling languages we use AMPL [5], as it is one of the most advanced modeling languages, and CMPL [6], as its open source alternative. These languages provide interfaces to a wide set of solvers, both commercial, such as CPLEX [21], and open source, such as CBC [22].

The mathematical programming approach enables us to formally define optimization problem. AMPL (a mathematical programming language) and CMPL (COIN mathematical programming language) are algebraic mathematical modeling languages that resemble traditional mathematical notation to describe variables, objectives, and constraints. Algebraic modeling languages allow expressing a wide range of optimization problems: linear, nonlinear, and integer. The advantage of AMPL is that it is one of the most advanced mathematical programming languages, while CMPL is easier to use in open source projects. AMPL or CMPL enables us to separate model definition and instance specific data, usually into three files: model, data, and calling script. The model file defines abstract optimization model: sets and parameters, objective and constraints. The data file populates the sets and parameters with the numbers for the particular instance of the problem. Both model and data files are loaded from

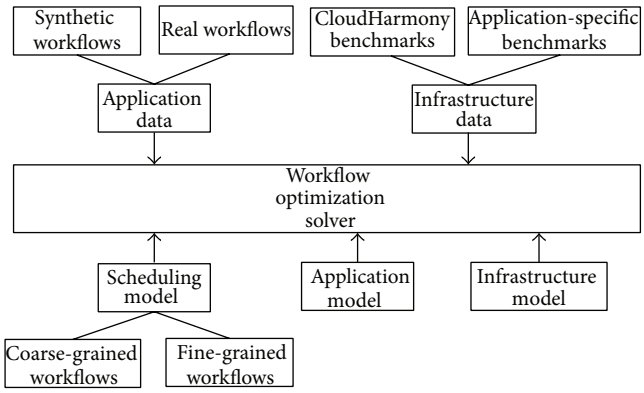


Figure 1: An overview of our approach to workflow scheduling. Mathematical models are input to the solver: application, infrastructure, and scheduling models, together with corresponding datasets.

calling script that may do some pre- or postprocessing. In addition, it is possible to import and export data and results into some external format such as YAML for analysis or integration with external programs.

The input to the solver has to be prepared in the form of a problem description. We separate the problem into an application model (in this case the leveled workflows) and infrastructure model (cloud consisting of compute sites running virtual machines and object storage such as Amazon S3). In addition, a scheduling model has to be defined, specifying how to calculate the objective and constraints using the application and infrastructure models. The challenge in the scheduling model is that it has to be developed to allow the solver to find a solution in a reasonable amount of time, so it must incorporate appropriate assumptions, constraints, and approximations. We discuss these assumptions in detail in Section 5.

The scheduling problems that we deal with in this paper are formulated as mixed integer programming (MIP) problems. This class of optimization problems has linear objective and constraints, while some or all of variables are integer-valued. Such problems are solved by using branch-and-bound approach that uses a linear solver to solve subproblems. Moreover, the solvers can relax the integrality of the variables in order to estimate the solution, since no integer solution can be better than the solution of the same problem in continuous domain. The difference between the best integer solution found and the noninteger bound can be used to estimate the accuracy of solution and to reduce the search time (see Section 7.1).

In this paper, we describe two alternative scheduling models: for workflows with fine-grained and coarse-grained tasks. This is motivated by the observation [11] that the granularity of the tasks in the workflows has significant influence on the results of the optimization. The best results can be obtained when the average runtime of the tasks is similar to the billing cycle of the cloud provider, such as 1 hour on Amazon EC2. To address this issue, we developed another scheduling model for fine-grained tasks and deadlines shorter than one hour, which corresponds to the real characteristics of the Montage workflow.

The scheduling models have to be provided with the actual values of parameters, consisting of the application data and infrastructure data. To evaluate our models, we use two sources of application data: synthetic workflows obtained from the workflow generator gallery [23] and real data obtained from our recent benchmarks performed on Amazon EC2. As infrastructure parameters, we use two sources: CloudHarmony benchmarks [24] that publish CPU performance of selected cloud providers and our own application-specific benchmark results. For research presented in this paper, we selected the Montage workflow and EC2 cloud as an example of a real workflow and infrastructure.

In the following sections, we describe the models and datasets used in more detail.

4. Application and Infrastructure Models

In this paper we focus on large-scale scientific workflows [23]. Examples of such workflows come from a wide variety of domains including bioinformatics (Epigenomics [25], SIPHT [26]), astronomy (Montage [27]), earthquake science (CyberShake [28]), and physics (LIGO [29]). Such workflows typically consist of a large number of computationally intensive tasks, processing large amounts of data.

We assume that each workflow may be represented with a directed acyclic graph (DAG) where nodes in the graph represent computational tasks, and the edges represent data- or control-flow dependencies between the tasks. Each task has a set of input and output files. We assume that the task and file sizes are known in advance.

Based on the characteristics of large-scale workflows, we assume that a workflow is divided into several levels that can be executed sequentially and tasks within one level not do depend on each other (see Figure 2). Each level represents a set of tasks that can be partitioned in several groups (A, B, etc.) that share computational cost and input/output size. We assume that only one task group is executed on a specific cloud instance. This forbids instance sharing between multiple levels, which means that each application may need its own specific VM template.

Similar to what is in [7], we assume multiple heterogeneous cloud IaaS infrastructures such as Amazon EC2, RackSpace, or ElasticHosts. Clouds have heterogeneous virtual machine instance types, with limits on the number of instances per cloud, for example, 20 for EC2 and 15 for RackSpace. Input and output data are stored on a cloud object store such as Amazon S3 or RackSpace CloudFiles. In our model, all virtual machine instances are billed per hour of usage, and there are fees associated with data transfer in/out of the cloud. In the application model, we also assume that there is a small constant cost of execution of a single task, which can correspond, for example, to the cost of a request to the queuing system such as Amazon SQS. The model allows us to include a private cloud where costs are set to 0.

For evaluation, we use synthetic workflows that were generated using historical data from real applications [23], as

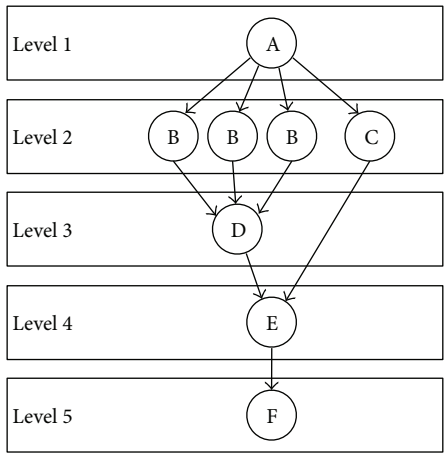


Figure 2: Example application structure.

well as the data from our own measurements. The synthetic workflows were generated using code developed in [30], with task runtimes based on distributions gathered from running real workflows. The experimental data come from execution of Montage workflow on Amazon EC2 using the HyperFlow workflow management system [31].

5. Formulation of the Scheduling Problem

In this section we give the mathematical formulation of the models, beginning with application and infrastructure models, and then describe the scheduling models for coarse-grained and fine-grained workflows. We have intentionally decided to present the problem in a form which is different from the routine statement of mathematical progrramming way. The main reason was to make it easily understood for reasearchers engaged in workflow execution optimization.

To perform optimization of the total cost of the workflow execution, mixed integer problem (MIP) is formulated and implemented using a mathematical programming language. First, we have implemented the optimization model using AMPL [5] and solved it with CPLEX solver, then we ported it to open source CMPL [6] and solved it with CBC solver. Both systems require to specify input datasets and variables to define the search space, as well as constraints and an objective function to be optimized.

5.1. Application and Infrastructure Model

Input Data. The formulation requires a number of input sets to represent the infrastructure model. This is a similar way to an approach presented in [7]. The infrasructure is described with the following sets:

(i) $S = \{s3, \text{cloudfiles}\}$: set of available cloud storage sites,

(ii) $P = \{\text{amazon}, \text{rackspace}, \ldots\}$: set of possible computing cloud providers,

(iii) $I = \{m1.\text{small}, \ldots, gg.1gb, \ldots\}$: set of instance types,

(iv) $PI_p \subset I$: set of instances that belong to provider $p \in P$,

(v) $LS_s \subset P$: set of compute cloud providers that are local to the storage platform $s \in S$,

(vi) $n_p^{P\max}$: upper limit of number of instances allowed by a cloud provider $p \in P$.

Introducing PI_p and LS_s enables one to describe the locality between compute and storage resources. This is an important aspect, since the cloud providers typically charge for the cost of data transfer out of a cloud site, while the transfers within the site are free.

Each instance type $i \in I$ is described with the following parameters:

(i) p_i^I: a fee (in US dollars) for running the instance of type i for one hour,

(ii) ccu_i^I: performance of instance of type i in CloudHarmony Compute Units (CCU),

(iii) cpu_i^I: number of virtual CPU cores assigned to an instance of type i,

(iv) $p_i^{I\text{out}}$, $p_i^{I\text{in}}$: price for nonlocal data transfer to and from an instance of type i in US dollars per MiB (1 MiB = $1024 \cdot 1024$ bytes),

(v) $n_i^{I\max}$: upper limit of the number of instances of type i, equal to $n_p^{P\max}$, where p is the provider of instance type i.

This instance model assumes the hourly billing cycle, which is the case for most of the cloud providers, notably for Amazon EC2.

Storage site $s \in S$ is characterized by

(i) $p_s^{S\text{out}}$ and $p_s^{S\text{in}}$: price in dollars per MiB for nonlocal data transfers.

Additionally, we need to provide data transfer rates $r_{i,s}$ between a given storage site s and instance i in MiB per second.

Our application model is different from that in [7] because it is designed for workflow scheduling where tasks are grouped into levels. This fact is described with the following characteristics:

(i) L: a set of levels the workflow is divided into,

(ii) G: a set of task groups (A, B, etc., in Figure 2); tasks in groups have the same computational cost and input/output size,

(iii) $G_l \subset G$: a set of task groups belonging to a level $l \in L$,

(iv) A_g^{tot}: number of tasks in a group $g \in G$,

(v) t_g^x: execution time in hours of a single task in a group g on a machine with the processor performance of 1 CloudHarmony Compute Unit (CCU) [32],

(vi) d_g^{in}, d_g^{out}: data size for input and output of a task in group g in MiB,

(vii) p^R: price per task for a queuing service, such as Amazon SQS,

(viii) t^D: total time allowed for completing workflow (deadline).

The application model assumes that the estimated execution time t_g^x is known in advance; that is, it is obtained using benchmarks or other estimation methods [33], such as regression or performance modelling. When using general purpose cloud benchmarks, such as CloudHarmony [24], which provide processor performance measured in CCU, the t_g^x depends only on a task in group g since we assume that the actual task execution time on a specific instance is inversely proportional to the processing speed of the instance expressed in the number of CCU. As it is not always the case, since different tasks may have different processing speeds on different instances, it is also possible to provide execution time predictions at instance level: $t_{g,i}^x$. The scheduling model can use such data if it is available. In Section 6.2 we provide an example of such a dataset for the Montage workflow on Amazon EC2.

5.2. Scheduling Model for Coarse-Grained Workflows. In this model, we schedule groups of tasks of the same type divided into levels. We do not schedule individual tasks as in [34] to keep MIP problem small, as one of the requirements is that optimization time is shorter than the workflow execution time. The coarse-grained workflows are such workflows where task execution times are in the order of one hour. This is important, as we assume the hourly billing cycle of the cloud, so the model has to optimize the task assignment in such a way that the hourly slots of allocated resources (VM instances) are as fully utilized as possible.

To keep this model in the MIP class, we had to take a different approach than in [7] and schedule each virtual machine instance separately. A drawback of this approach is that we need to increase the number of decision variables. We have also divided the search space by storage providers, solving the problem separately for each storage and selecting the best result. Additionally, the deadline becomes a variable with an upper bound, as it may happen that a shorter deadline may actually give a cheaper solution (see Figure 5 and its discussion).

Auxiliary Parameters. Based on the input parameters, in the scheduling model we derive a set of precomputed parameters that are used for expressing objectives and constraints. The transfer time is computed based on the input and output data size and the transfer rate between an instance and the storage. The time for processing a task is a sum of computing and data transfer time. The cost of data transfer is a sum of cost of input and output data, both including the transfer fees at the source and destination cloud site. The indexing of instances is introduced; for example, all m1.small instances are numbered $0, 1, 2, \ldots$, to distinguish between individual instances of a given type:

(i) $s \in S$: a selected storage site,

(ii) $t_{g,i,s}^{\text{net}} = (d_g^{\text{in}} + d_g^{\text{out}})/(r_{i,s} \cdot 3600)$: transfer time in hours, that is, time for data transfer between instances of type i and storage site s for a task in task group g,

(iii) $t_{g,i,s}^u = t_g^x/\text{ccu}_i^I + t_{g,i,s}^{\text{net}}$: time in hours for processing a task in group g on instance of type i using storage site s,

(iv) $c_{g,i,s}^T = (d_g^{\text{out}} \cdot (p_i^{I\text{out}} + p_s^{\text{Sin}}) + d_g^{\text{in}} \cdot (p_s^{\text{Sout}} + p_i^{I\text{in}}))$: a cost of data transfer between an instance of type i and a storage site s when processing task in group g,

(v) I_i^{idx}: a set of possible indices for instances of type i (from 0 to $n_i^{I\max} - 1$).

Variables. Variables of the optimization problem are

(i) $N_{g,i,k}$: 1 iff (if and only if) instance of type i with index $k \in I_i^{idx}$ is launched to process task group g, otherwise, 0 (binary);

(ii) $H_{g,i,k}$: for how many hours the instance of index k is launched (integer);

(iii) $T_{g,i,k}$: how many tasks of g are processed on that instance (integer);

(iv) D_l^t: actual computation time for level l (real);

(v) D_l: maximal number of hours (deadline) that instances are allowed to run at level l (integer).

The variables defined in this way allow the solver to search over the space of possible assignments of instances to task groups ($N_{g,i,k}$) with a varying number $H_{g,i,k}$ of hours each instance is launched and number $T_{g,i,k}$ of tasks processed on these instances. The deadline is divided into subdeadlines for each workflow level l, while the actual computation time D_l^t can be shorter than the deadline D_l.

Objective. The scheduling problem is represented as a cost minimization problem. The cost of running a single task is defined as follows:

$$\left(t_g^{\text{net}} + t_g^u\right) \cdot p_i^I + \tag{1}$$

$$d_g^{\text{in}} \cdot \left(p_s^{\text{Sout}} + p_i^{I\text{in}}\right) + \tag{2}$$

$$d_g^{\text{out}} \cdot \left(p_i^{I\text{out}} + p_s^{\text{Sin}}\right) + \tag{3}$$

$$p^R, \tag{4}$$

and it includes the cost of the computing time of instance (1), the cost of transfer of input data (2), that of output data (3), and request price (4).

The objective function C_{tot} represents the total cost which is a sum of task costs computed over all the task groups, all the instance types, and the individual instances. It is defined as

$$C_{\text{tot}} = \sum_{g \in G, i \in I, k \in I_i^{idx}} \left(p_i^I \cdot H_{g,i,k} + p^R + c_{g,i,s}^T\right) \cdot T_{g,i,k}. \tag{5}$$

To properly implement the assumptions we impose on the application, infrastructure, and scheduling models, the following *constraints* have to be introduced.

(1) $\sum_{l\in L} D_l \le t^D$ ensures that the sum of subdeadlines of all levels is not greater than the workflow deadline, that is, that the workflow finishes in the given deadline.

(2) To fix that the actual execution time of a level, rounded up to a full hour, gives us the level subdeadline ($D_l = \lceil D_l^t \rceil$), we require that $\forall_{l\in L} D_l^t \le D_l \le D_l^t + 1$.

(3) $\forall_{g\in G,i\in I,x\in I_i^{idx}} N_{g,i,k} \le H_{g,i,k} \le N_{g,i,k} \cdot \lceil t^D \rceil$ ensures that the number of computing hours of an instance $H_{g,i,k}$ may be nonzero only if instance is active ($N_{g,i,k}$ is 1), and it cannot exceed the deadline.

(4) $\forall_{g\in G,i\in I,k\in I_i^{idx}} N_{g,i,k} \le T_{g,i,k} \cdot N_{g,i,k} \cdot A_g^{\text{tot}}$ ensures that the computing tasks $T_{g,i,k}$ may be allocated to an instance only if the instance is active and that their number does not exceed the total number of tasks in group t.

(5) $\forall_{l\in L,g\in G_l,i\in I,k\in I_i^{idx}} H_{g,i,k} \le D_l$ enforces the level deadline on the actual runtimes of each instance.

(6) $\forall_{l\in L,g\in G_l,i\in I,k\in I_i^{idx}} T_{g,i,k} \cdot t_{g,i,s}^u \le D_l^t$ enforces that all the tasks allocated to the instance complete their work within the computing time of their level D_l^t.

(7) To make sure that all the instances run for enough time to process all tasks allocated to them we adjust $H_{g,i,k}$, respectively, to $T_{g,i,k}$: $\forall_{g\in G,i\in I,k\in I_i^{idx}} T_{g,i,k} \cdot t_{g,i,s}^u \le H_{g,i,k} \cdot T_{g,i,k} \cdot t_{g,i,s}^u + 1$.

(8) $\forall_{g\in G} \sum_{i\in I,k\in I_i^{idx}} T_{g,i,k} = A_g^{\text{tot}}$ ensures that all the tasks are processed.

(9) To reject symmetric solutions and thus to reduce the search space, we add three constraints:

(a) $\forall_{g\in G,i\in I,k\in\{1\cdots(n_i^{I\max}-1)\}} H_{g,i,k} \le H_{g,i,k-1}$,
(b) $\forall_{g\in G,i\in I,k\in\{1\cdots(n_i^{I\max}-1)\}} N_{g,i,k} \le N_{g,i,k-1}$,
(c) $\forall_{g\in G,i\in I,k\in\{1\cdots(n_i^{I\max}-1)\}} T_{g,i,k} \le T_{g,i,k-1}$.

(10) Finally, the constraint $\forall_{l\in L,p\in P} \sum_{i\in PI_p,g\in G_l,k\in I_i^{idx}} N_{g,i,k} \le n_p^{P\max}$ enforces instance limits per cloud.

The scheduling model presented above shows its advantages if the workflow tasks are about one hour long or larger, and the deadline exceeds one hour. For fine-grained workflows, such as Montage where most task execution times are in order of seconds and the whole workflow may be finished within an hour, a model can be simplified.

5.3. Scheduling Model for Fine-Grained Workflows. When scheduling workflows with many short tasks and with deadlines shorter than the cloud billing cycle (one hour), we do not need to use the $H_{g,i,k}$ variable that counts the number of hours the instance is running. Thus we can assume that each level completes its work in one hour. This assumption reduces the number of decision variables making the MIP problem faster to solve. We also add an assumption that only one instance type may be used for each task type, which also reduces the search space.

In addition to these assumptions, we changed the way how the data transfer time is computed. Since for short tasks the data access latency is important, in addition to transfer rate $r_{i,s}$ we also provide the latency parameter $r_{i,s}^{\text{lat}}$. The actual values come from linear regression of experimental data, where we run Montage workflow on Amazon S3. In the fine-grained scheduling model, we also use execution time predictions at instance level: $t_{g,i}^x$. The t^u is normalized by the number of CPU cores present on the VM if there are enough tasks to be processed in parallel. The modifications mentioned in this paragraph may also be applied to the coarse-grained model if needed.

Based on these modifications, the auxiliary parameters transfer time $t_{g,i,s}^{\text{net}}$ and unit time $t_{g,i,s}^u$ are computed as follows:

(i) $t_{g,i,s}^{\text{net}} = (d_g^{\text{in}} + d_g^{\text{out}})/(r_{i,s} \cdot 3600) + r_{i,s}^{\text{lat}}$;
(ii) $t_{g,i,s}^u = (t_{g,i}^x + t_{i,s}^{\text{net}})/\min(\text{cpu}_i^I, A_g^{\text{tot}})$.

The remaining part of the model has the following form.

Variables. Variables are similar to the ones in the coarse-grained model, but the problem has less dimensions, since there is no need to use $H_{g,i,k}$ and to distinguish instances by index x:

(i) $A_{g,i}$ tells if instances of type i are used to process task group t (binary);
(ii) $N_{g,i}$ tells how many instances of type i are launched to process task group g (integer);
(iii) $T_{g,i}$ tells how many tasks in group g are processed on instances of type i (integer);
(iv) D_l^t tells actual computation time for level l (real).

Objective. The cost function C_{tot} is computed in a similar way, by summing the costs of all the task groups over all of the instances, taking into account the task assignment $T_{g,i}$:

$$C_{\text{tot}} = \sum_{g\in G,i\in I} \left(p_i^I \cdot N_{g,i} + p^R + c_{g,i,s}^T\right) \cdot T_{g,i}. \tag{6}$$

Constraints. The constraints are as following:

(1) $\sum_{l\in L} D_l \le t^D$ ensures that workflow finishes before the given deadline;

(2) $\forall_{g\in G,i\in I} A_{g,i} \le N_{g,i} \le A_{g,i} \cdot n_i^{I\max}$ ensures that the number of active instances $N_{g,i}$ is consistent with the binary variable $A_{g,i}$ and does not exceed the instance limit;

(3) $\forall_{g\in G,i\in I} N_{g,i} \le T_{g,i} \le N_{g,i} \cdot A_g^{\text{tot}}$ ensures that there are no empty instances and that the number of assigned tasks does not exceed the total number of tasks;

(4) $\forall_{l\in L,g\in G_l,i\in I} T_{g,i} \cdot t^u_{g,i,s} \leq D^t_l \cdot N_{g,i}$ enforces that a level finishes work in D^t_l;

(5) $\forall_{g\in G} \sum_{i\in I} T_{g,i} = A^{\text{tot}}_g$ ensures that all tasks are processed;

(6) $\forall_{g\in G} \sum_{i\in I} A_{g,i} = 1$ ensures that only one instance type is used for a given task;

(7) $\forall_{g\in L,p\in P} \sum_{i\in PI_p,g\in G_l} N_{g,i} \leq n^{P\max}_p$ enforces instance limits per cloud, for each task group and instance type.

This scheduling model yields reasonable results only for the cases when it is actually possible to complete all the workflow tasks before the deadline. If not, the solver will not find any solution.

The optimization models introduced in this section were implemented using CMPL and AMPL effectively being workflow schedulers. The source code of the schedulers is available as an online supplement (https://github.com/kfigiela/optimization-models/tree/ppam-extended/workflows). The public repository on GitHub includes the model files, the data, and the scripts we used to run the solvers.

6. Application and Infrastructure Data Used for Evaluation

To perform optimization we need to provide optimization models defined in the previous section with data describing an application and an infrastructure. First, we used the generic infrastructure benchmarks obtained from CloudHarmony and the application data from the workflow generator gallery. Next, we performed our own experiments using the Montage workflows on Amazon EC2, which provided the application-specific performance benchmark of cloud resources together to obtain the real application data. The data gathered during experiments are inputs for the scheduler.

6.1. Data for Coarse-Grained Scheduler. To evaluate the coarse-grained scheduler on realistic data, we used CloudHarmony [24] benchmarks to parameterize the infrastructure model, and we used the workflow generator gallery workflows [23] as test applications. In the infrastructure model we assumed that we had 4 public cloud providers (Amazon EC2, RackSpace, GoGrid, and ElasticHosts) and a private cloud with 0 costs. The infrastructure had two storage sites: S3 which is local to EC2, and CloudFiles which is local to RackSpace, so data transfers between local virtual machines and storage sites are free.

We used the first generation of CloudHarmony CPU benchmarks described in [24]. CloudHarmony CPU benchmarks use CloudHarmony Compute Unit (CCU) as a unit for measuring CPU performance. It is calculated based on a set of general-purpose CPU benchmarks [32]. First generation benchmarks were calibrated relative to Amazon's `m1.small` instance and are now deprecated in favor of new benchmarks that are calibrated to nonvirtualized hardware. The new benchmark is compared to our benchmark data in Figure 3. Actual datasets are provided as an online supplement (https://github.com/kfigiela/optimization-models/tree/ppam-extended/workflows).

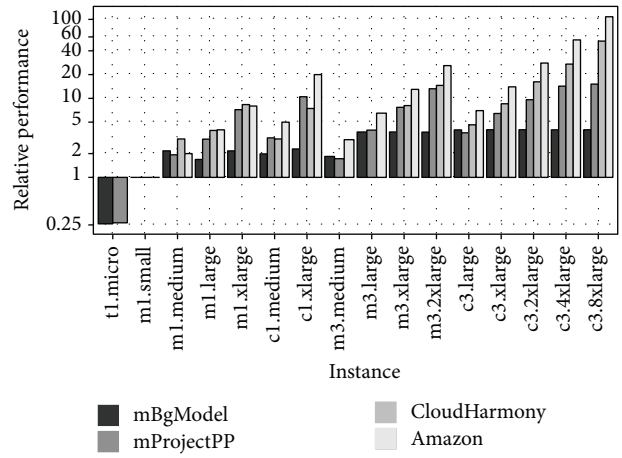


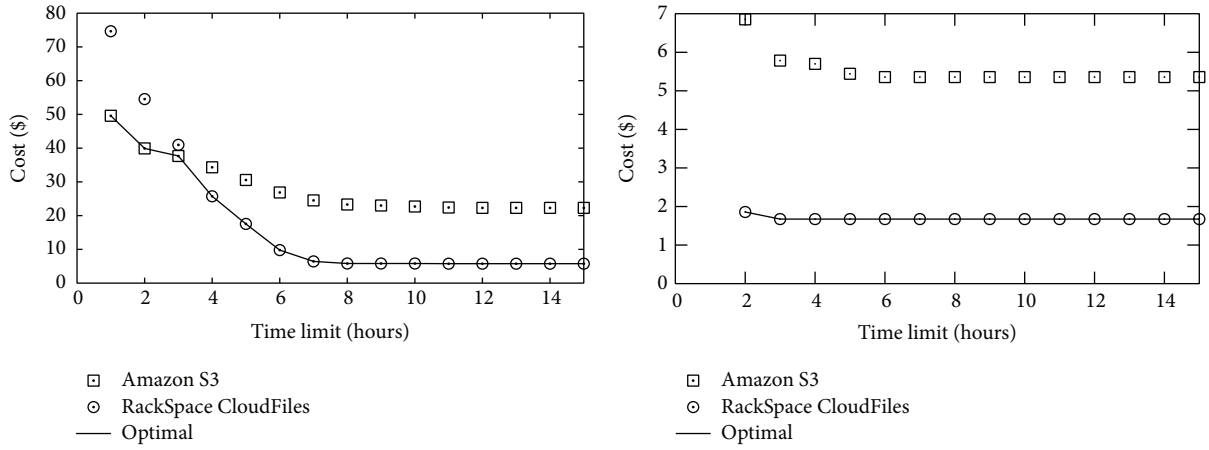
Figure 3: Amazon instance benchmarks for different tasks compared to the generic CloudHarmony benchmark and Amazon ECU. Data was normalized to `m1.small` instance type having relative performance of 1.0.

We tested the coarse-grained scheduler with all of the applications from the gallery: Montage, CyberShake, Epigenomics, LIGO, and SIPHT for all available workflow sizes (from 50 to 1000 tasks per workflow up to 5000 tasks in the case of SIPHT workflow). We varied the deadline from 1 to 30 hours with 1-hour increments. We solved the problem for two cases, depending on whether the data are stored on S3 or on CloudFiles.

6.2. Data for Fine-Grained Scheduler. Cloud benchmarks, such as CloudHarmony [24], are based on set of general-purpose benchmarks that do not necessarily represent scientific applications that are to be scheduled. In order to find out how it may differ, we run Montage workflow on several Amazon EC2 instance types. The workflow of 12700 tasks processing 8.5 GiB of photos rendered a mosaic of an 8×8 degree region at Orion Nebula from 2MASS survey.

Usually, benchmarks take into account the fact that instances provide multiple virtual cores that speed up multithreaded applications, but it has no impact on single threaded ones. Montage workflow tasks are single threaded and therefore in our experiment the number of execution threads running in parallel was equal to the number of virtual cores. We used the HyperFlow workflow engine [31] to drive workflow execution. In the experiment, we used EBS (elastic block storage) volume for data storage instead of S3 (simple storage service); however we measured the transfer times to and from S3 separately. EBS is different from S3 as it provides block level access (i.e., filesystem) to the data volume, while S3 is object store available as a service by REST API.

The data we gathered in experiments may be used to calculate application-specific performance metric of the instance (ECU-like). In Figure 3 we compare our results with CloudHarmony benchmarks. It shows that, for the



(a) 500 tasks, 4 GiB data size

(b) 400 tasks, 1 GiB data size (legend the same as above)

Figure 4: Result of coarse-grained scheduling for the Epigenomics application.

tasks forming the parallel levels of Montage workflow (such as `mProjectPP` [27]), the performance of the instances is proportional to the generic CPU benchmark. On the other hand, for the levels that are not parallel (e.g., `mBgModel`), there is no difference between cheaper `m3.large` and more expensive instance types (e.g., `c3.8xlarge`). Those instance types are deployed on the same generation of hardware, so their performance for single threaded applications is very similar. Additionally, as a reference we show the instance performance provided by Amazon in ECU (EC2 Compute Units).

The observation from this evaluation is that the benchmarks from CloudHarmony give better approximation to the task performance than the generic ECU value. Moreover, it is important to distinguish between parallel and sequential workflow levels when selecting the virtual machine instance type. The dataset obtained in this experiment was used for evaluation of fine-grained scheduling model in Section 7.2.

7. Evaluation of Optimization Models

In this section, we present the results of optimization, obtained by applying our schedulers to the application and infrastructure data. First, we show the results of using the coarse-grained scheduler applied to the generic CloudHarmony datasets. Next, we present the results of the fine-grained scheduler applied to the dataset obtained from our experiments with the Montage workflow on EC2.

7.1. Results for Coarse-Grained Scheduling. Figure 4 shows the cost of execution of the Epigenomics application with two workflows of sizes 400 and 500 tasks as a function of deadline. For longer deadlines (over 6 hours), the private cloud instances and the cheapest RackSpace instances are used so the cost is low when using CloudFiles. For shorter deadlines, the cost grows rapidly, since we reach the limit of instances per cloud and additional instances must be spawned on a different provider, thus making the transfer costs higher. This effect is amplified in Figure 4(a), which differs from Figure 4(b) not only by the number of tasks, but also by the data size of the most data-intensive level. This means that the transfer costs are growing more rapidly, so it becomes more economical to store the data on Amazon EC2 that provides more powerful instances required for short deadlines.

One interesting feature of our scheduler is that for longer deadlines it enables finding the cost-optimal solutions that have shorter workflow completion time than the requested deadline. This effect can be observed in Figure 5 and is caused by the fact that for long deadlines the simple solution is to run the application on a set of the least expensive machines.

Figures 6(a) to 7(b) show results obtained for Cybershake, LIGO, Montage, and SIPHT workflows. These workflows have relatively small execution time, so even for short deadlines the scheduler is able to schedule tasks on the cheapest instances on a single cloud, thus resulting in flat characteristics.

To investigate how the scheduler behaves for workflows with the same structure, but with much longer runtimes of tasks, we run the optimization for Montage workflow with tasks 1000x longer. This corresponds to the scenario where tasks are in the order of hours instead of seconds. The results in Figure 8 show how the cost increases very steeply with shorter deadlines, illustrating the trade-off between time and cost. The difference between Figures 7(a) and 8 illustrates that the scheduler is more useful for workflows when tasks are of granularity that is similar to the granularity of the (hourly) billing cycle of cloud providers. Additionally, Figure 8 shows how the optimal cost depends on available clouds.

The runtime of the optimization algorithm for workflows with up to 1000 tasks ranges from a few seconds up to 4 minutes using the CPLEX [21] solver running on a server with 4 16-core 2.3 GHz AMD Opteron processors (model 6276), with CPLEX limited artificially to use only 32 cores. Figure 9(a) shows that the time becomes much higher for shorter deadlines and increases slowly for very long deadlines. This is correlated with the size of search space: the

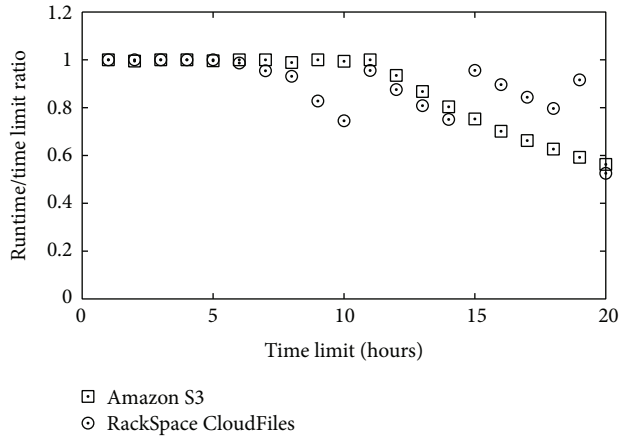
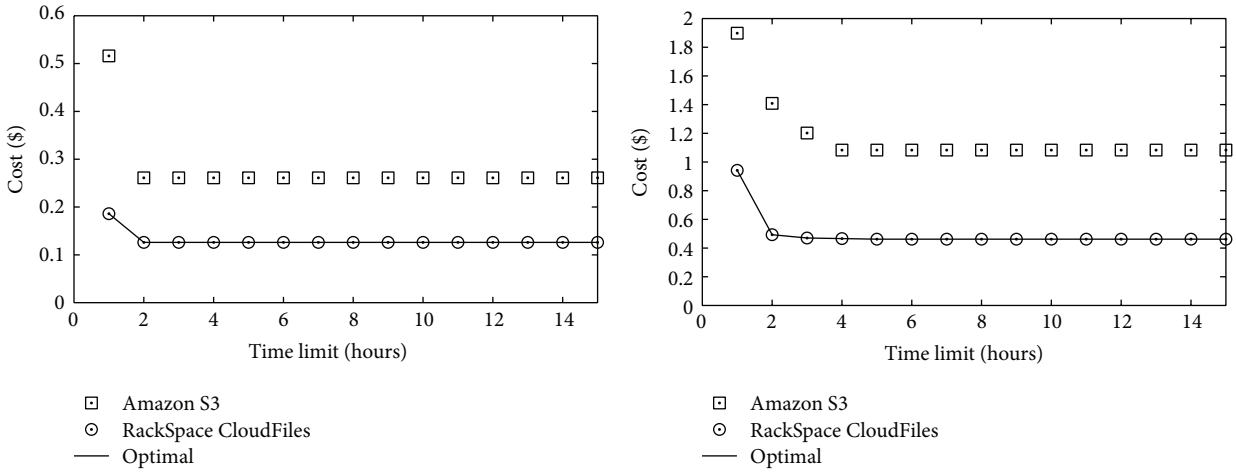


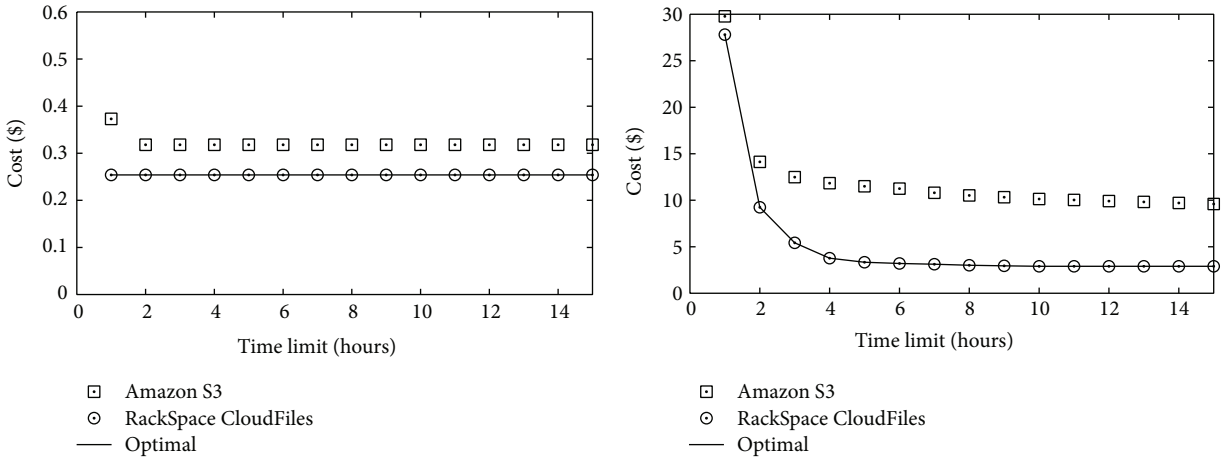
Figure 5: Ratio of the actual completion time to the deadline for the Epigenomics workflow with 500 tasks.



(a) CyberShake, 500 tasks

(b) LIGO, 500 tasks

Figure 6: Optimal cost found with the coarse-grained scheduling for CyberShake and LIGO applications.



(a) Montage, 500 tasks

(b) SIPHT, 5000 tasks

Figure 7: Optimal cost found by the scheduler for Montage and SIPHT applications.

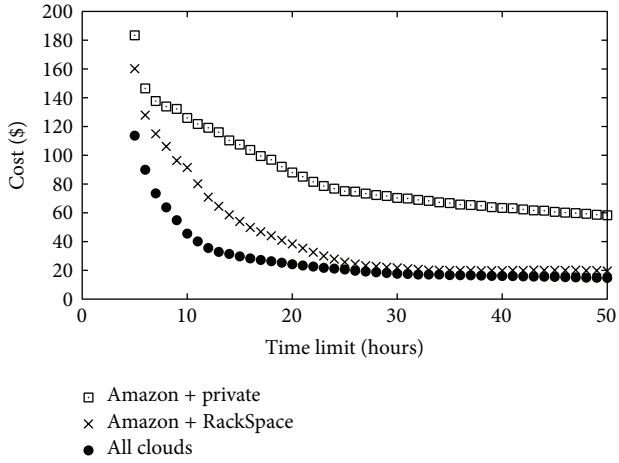
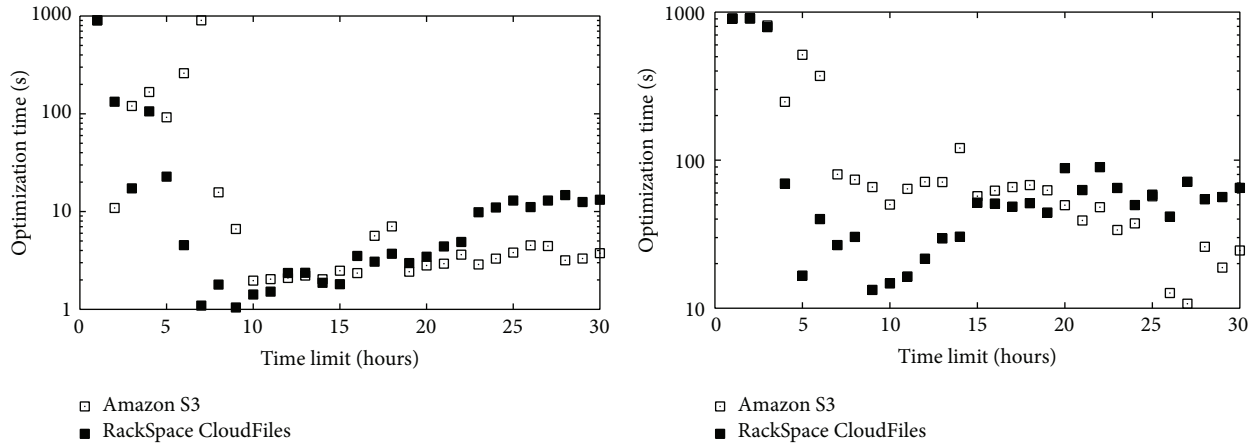


Figure 8: Optimal cost found by the coarse-grained scheduler for Montage workflow of 500 tasks with runtimes artificially multiplied by 1000 for different cloud infrastructures.



(a) Epigenomics, 600 tasks

(b) SIPHT, 5000 tasks

Figure 9: Optimization time of the solver.

longer the deadline, the larger the search space, while for shorter deadlines the problem has a very small set of acceptable solutions. The problem becomes more severe for bigger and more complex workflows like SIPHT as optimization time becomes very high (Figure 9(b)).

Figure 10 illustrates how the optimization time depends on MIP gap solver setting. The relative MIP gap is a relative diference between the best integer solution found by the solver and the possible optimal noninteger solution. The MIP gap value indicates to solver to stop when an integer feasible solution has been proved to be within a given percent of optimality [21]. Applying a relative MIP gap of 1% or 5% instead of default 0.01% shortens optimization time in orders of magnitude. Increasing the MIP gap to 5% did not decrease the quality of the result noticeably: the minimum cost obtained for the gap of 5% was higher only by 3.63% in the worst case.

7.2. Results for Fine-Grained Workflows and Short Deadlines. We performed optimization for deadlines ranging from 13 to 60 minutes, using the Amazon EC2 cloud, with S3 or local storage. When assuming that the storage is local, we set the $t^{\text{net}}_{g,i,s}$ fixed to 0, which may represent, for example, a very fast NFS storage when transfer times are negligible.

The results shown in Figure 11 have similar character to those we got in [7] and to the ones obtained using the coarse-grained scheduler and task runtimes artificially expanded (Figure 8). This observation leads to the conclusion that the granularity of the workflow tasks versus the granularity of the billing cycle of the cloud provider plays an important role in scheduling. In our case, we had to define two separate schedulers to address this issue. The problem, however, may be more complex when we assume more cloud providers with different billing cycles, such as hourly, 5-minute, or per-minute billing. This may be an interesting subject for further research.

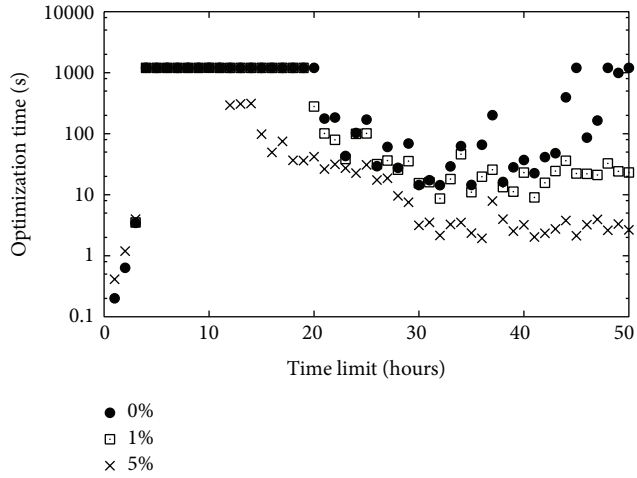


Figure 10: Solver runtime with different relative MIP gap (in percent), showing the relation between accuracy and runtime of the solver for the coarse-grained scheduler for Montage workflow of 500 tasks with runtimes artificially multiplied by 1000 for different cloud infrastructures.

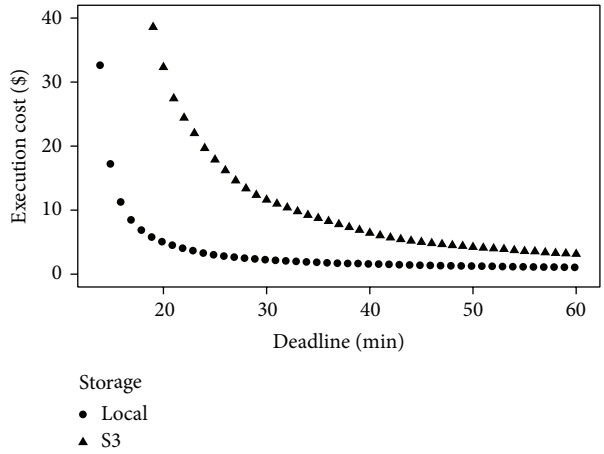


Figure 11: Montage workflow execution cost (8 × 8 degrees at M42) with S3 storage and local storage (i.e., very fast NFS).

8. Conclusions and Future Work

In this paper, we presented the schedulers using cost optimization for scientific workflows executing on multiple heterogeneous clouds. The models, formulated in AMPL and CMPL, allow us to find the optimal assignment of workflow tasks, grouped into levels, to cloud instances. We validated our models with a set of synthetic benchmark workflows as well as with the data of real astronomy workflow, and we observed that they gave useful solutions in a reasonable amount of computing time.

Based on our experiments with execution of Montage workflow on Amazon EC2 cloud and its characteristics, we developed separate scheduling models dedicated to coarse-grained workflows and to fine-grained workflows with short deadlines. We also compared the general-purpose cloud benchmarks, such as CloudHarmony, with our own measurements. The results underline the importance of application-specific cloud benchmarking, since the general purpose benchmarks can serve only as the rough approximation of the actual application performance. The observed relations between the granularity of the tasks and the performance of optimization models shows the influence of the cloud billing cycle on the cost optimizing workflow scheduling.

By solving the models for multiple deadlines, we can produce trade-off plots, showing how the cost depends on the deadline. We believe that such plots are a step towards a scientific cloud workflow calculator, supporting resource management decisions for both end-users and workflow-as-a-service providers.

In the future, we plan to apply this model to the problem of provisioning cloud resources for workflow ensembles [3], where the optimization of cost can drive the workflow admission decisions. We also plan to refine the models to better support smaller workflows by reusing instances between levels, to fine-tune the model, and to test different solver configurations to reduce the computing time, as well as to apply the optimization models to the problem of dynamic workflow scheduling in order to better handle the uncertainties in the infrastructure and the application.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was partially supported by the EC ICT VPH-Share Project (Contract 269978) and the KI AGH Grant 11.11.230.124. The work of K. Figiela was supported by the AGH Dean's Grant. E. Deelman acknowledges support of the National Science Foundation (Grant 1148515) and the Department of Energy (Grant ER26110). Access to Amazon EC2 was provided via the AWS in Education Grant. The authors would like to express their thanks to the reviewers for their constructive recommendations that helped them improve the paper.

References

[1] E. Deelman, G. Juve, M. Malawski, and J. Nabrzyski, "Hosted science: managing computational workflows in the cloud," *Parallel Processing Letters*, vol. 23, no. 2, Article ID 1340004, 2013.

[2] AWS, "AWS public datasets," 2013, http://aws.amazon.com/publicdatasets/.

[3] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of the 24th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)*, IEEE, November 2012.

[4] M. Bubak, M. Kasztelnik, M. Malawski, J. Meizner, P. Nowakowski, and S. Varma, "Evaluation of cloud providers for VPH applications," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-Grid '13)*, May 2013.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press, 2002.

[6] M. Steglich, "CMPL (Coin mathematical programming language)," 2014, https://projects.coin-or.org/Cmpl.

[7] M. Malawski, K. Figiela, and J. Nabrzyski, "Cost minimization for computational applications on hybrid cloud infrastructures," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1786–1794, 2013.

[8] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS '08)*, pp. 1–10, IEEE, November 2008.

[9] R. Duan, R. Prodan, and X. Li, "A sequential cooperative game theoretic approach to storage-aware scheduling of multiple large-scale workflow applications in grids," in *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing (Grid '12)*, pp. 31–39, IEEE, September 2012.

[10] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, ACM, New York, NY, USA, November 2011.

[11] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Cost optimization of execution of multi-level deadline-constrained scientific workows on clouds," in *Parallel Processing and Applied Mathematics—10th International Conference, PPAM 2013, Warsaw, Poland, September 8–11, 2013, Revised Selected Papers, Part I*, vol. 8384 of *Lecture Notes in Computer Science*, pp. 251–260, Springer, Berlin, Germany, 2014.

[12] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[13] J. J. Durillo, H. M. Fard, and R. Prodan, "MOHEFT: a multi-objective list-based method for workflow scheduling," in *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom '12)*, pp. 185–192, Taipei, Taiwan, December 2012.

[14] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[15] R. van den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973–985, 2013.

[16] S. Pandey, A. Barker, K. K. Gupta, and R. Buyya, "Minimizing execution costs when using globally distributed Cloud services," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 222–229, April 2010.

[17] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC '11)*, pp. 229–238, ACM, San Jose, Calif, USA, 2011.

[18] H. Kim, Y. El-Khamra, I. Rodero, S. Jha, and M. Parashar, "Autonomic management of application workflows on hybrid computing infrastructure," *Scientific Programming*, vol. 19, no. 2-3, pp. 75–89, 2011.

[19] R. Tolosana-Calasanz, J. Á. Bañares, C. Pham, and O. F. Rana, "Enforcing QoS in scientific workflow systems enacted over Cloud infrastructures," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1300–1315, 2012.

[20] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Using time discretization to schedule scientific workflows in multiple cloud providers," in *Proceedings of the IEEE 6th International Conference on Cloud Computing (CLOUD '13)*, pp. 123–130, Santa Clara, Calif, USA, July 2013.

[21] IBM, "IBM ILOG CPLEX Optimization Studio CPLEX User's Manual," 2013, http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r5/topic/ilog.odms.studio.help/pdf/usrcplex.pdf.

[22] J. Forrest, "Cbc (coin-or branch and cut) open-source mixed integer programming solver," 2012, https://projects.coin-or.org/Cbc.

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

[24] CloudHarmony, "Benchmarks," 2014, http://cloudharmony.com/benchmarks.

[25] USC epigenome center, http://epigenome.usc.edu.

[26] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs," *PLoS ONE*, vol. 3, no. 9, Article ID e3197, 2008.

[27] G. B. Berriman, E. Deelman, J. C. Good et al., "Montage: a grid enabled engine for delivering custom science-grade mosaics on demand," in *Optimizing Scientific Return for Astronomy through Information Technologies*, vol. 5493 of *Proceedings of SPIE*, pp. 221–232, June 2004.

[28] P. Maechling, E. Deelman, L. Zhao et al., "SCEC cyber-shake workows—automating probabilistic seismic hazard analysis calculations," in *Workows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds., pp. 143–163, Springer, London, UK, 2007.

[29] A. Abramovici, W. E. Althouse, R. W. P. Drever et al., "LIGO: the laser interferometer gravitational-wave observatory," *Science*, vol. 256, no. 5055, pp. 325–333, 1992.

[30] Workflow Generator, 2014, https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

[31] B. Balis, "Hypermedia workflow: a new approach to Data-Driven scientific workflows," in *Proceedings of the SC Companion: High Performance Computing, Networking Storage and Analysis (SCC '12)*, pp. 100–107, November 2012.

[32] Cloud Harmony, "What is ECU? CPU benchmarking in Cloud," 2010, http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html.

[33] R. F. da Silva, G. Juve, E. Deelman et al., "Toward fine-grained online task characteristics estimation in scientific workflows," in *Proceedings of the 8th Workshop on Workows in Support of Large-Scale Science (WORKS '13)*, pp. 58–67, ACM, Denver, Colo, USA, November 2013.

[34] R. Van Den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD '10)*, pp. 228–235, Miami, Fla, USA, July 2010.