AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

Multi-stage optimization of workflow execution in clouds

Wieloetapowa optymalizacja wykonania grafów zadań w chmurze obliczeniowej

Autor:	Tomasz Dziok
Kierunek studiów:	Informatyka
Opiekun pracy:	dr inż. Maciej Malawski

Kraków, 2016

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): "Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) "Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem "żeńskim samorządu studenckiego, zwanym dalej "sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Foremost, I would like to express my sincere gratitude to my supervisor Maciej Malawski, for the continuous support of my M.Sc. study, motivation, enthusiasm, and immense knowledge. His guidance helped me a lot during my research and writing of this thesis. Besides my supervisor, I would also like to thank Kamil Figiela for valuable discussions and consultancy. Finally I would like to thank my wife and parents for their endless love and support.

Contents

List of Figures

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, there is proposal of an adaptive scheduling algorithm for deadline constrained workflows consisting of multiple levels. The algorithm provides a global approximate plan for the whole workflow in a first phase, and a local detailed schedule for the current level of the workflow in the second one. By applying this procedure iteratively after each level completes, the algorithm is able to adjust to the runtime variation. Each phase uses 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.

This work contains:

1. Introduction to presented topics [chapters 1. Introduction and 2. State of the Art Overview].
2. Overview of existing solutions [chapter 2. State of the Art Overview].
3. Detailed description of presented algorithm [chapter 3. Adaptive Algorithm].
4. Implementation details [chapter 4. Implementation].
5. Results of performed experiments [chapter 5. Experiments and Results].
6. Conclusions and future work [chapter 6. Conclusions and Future Work].
7. Appendix with file formats and outputs [appendix A. File Formats and Outputs].
8. Appendix with article presented in Parallel Processing and Applied Mathematics with presented algorithm [appendix B. Publication].

1. Introduction

This chapter introduces topics covered in this work and presents its goal. Section 1.1 describes motivation why this work has been started. Next two ones (1.2 and 1.3) quickly introduce cloud computing [19] [20] and scientific workflows [21] [22]. Then section 1.4 contains problem statement and in 1.5 there is a goal of thesis with listed tasks which should be completed.

1.1. Motivation

Every year cloud computing becomes more and more popular. There are a lot of applications of clouds. Among them are scientific workflows which are used in almost all areas of science. They become more complex and time consuming (even up to a few days). Efficient usage of available resources in clouds are crucial point when taking into account the cost of scientific research. In other words effective usage allows to save money.

There are many scheduling algorithms, but unfortunately not many of them are dedicated to workflows which address uncertainties. These uncertainties are big issue because they have influence on real execution time which affects costs and given deadline. Additionally they come from various, independent sources. Handling them during scheduling workflows is an interesting challenge.

The problems described above are topics of my interests and they constitute the reasons for preparing this work.

1.2. Cloud Computing

Cloud computing is another episode of dynamic development of computer science. There are a few types of clouds. In this work, IaaS (Infrasturcture as a Service) [5] is in an area of interest. Basically it means that user can request on demand specified number of various types of preconfigured virtual machines. Generally, each type is characterised by performance and price. Besides virtual machines it is possible to request other resources like for example, storage. What is important, whole management of available resources is performed remotely by dedicated software or webservices. Cloud resources are available for all - not only for chosen companies or organisations. In other words, everybody can gain access and this is why popularity of the cloud computing is increasing.

1.3. Scientific Workflows

Scientific Workflow is a series of computional tasks which must be executed to achieve the final result of experiment or simulation. They are widely used in science world and become more and more popular every year. Workflows are used e.g. for generating mosaic of the sky from multiple images (Montage, is presented on Fig. 1.2), in genome sequence processing (Epigenomic) or for characterizing earthquakes (Cybershake). Execution of complex workflows could take even up to a few days. Basic workflow is presented on Fig. 1.1. On Pegasus website [26] the workflows gallery is available.

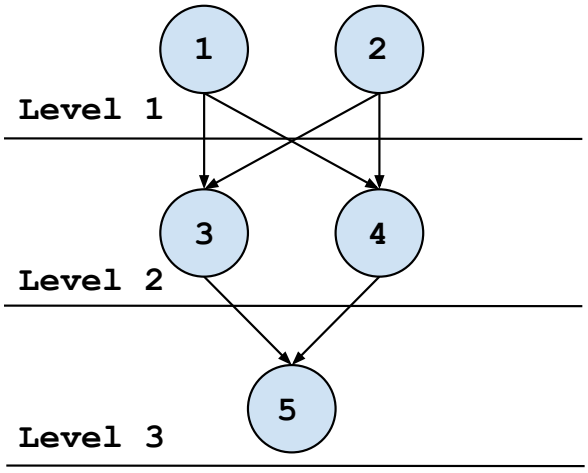


Figure 1.1: DAG example

1.4. Problem Statement

Problem is characterized as follows: there is a workflow (set of tasks) which must be executed on available cloud infrastructure with minimized total cost under given deadline constraint.

There is an information about dependencies between tasks (that one must be executed before another one). Each task has estimated required resources. Infrastructure is a set of available virtual machines characterized by performance (CPU, RAM, etc.) and price per time unit.

As a result there is an expectation to have information on which virtual machine each task should be executed to keep total time under deadline and minimize total cost.

1.5. Goal of Thesis

The goal of thesis is to design, implement and verify an algorithm that optimizes the workflow execution in the cloud by minimizing total cost under the given deadline with uncertain tasks estimates.

To achieve the above goal, the following tasks are defined:

1. Analyze problem.

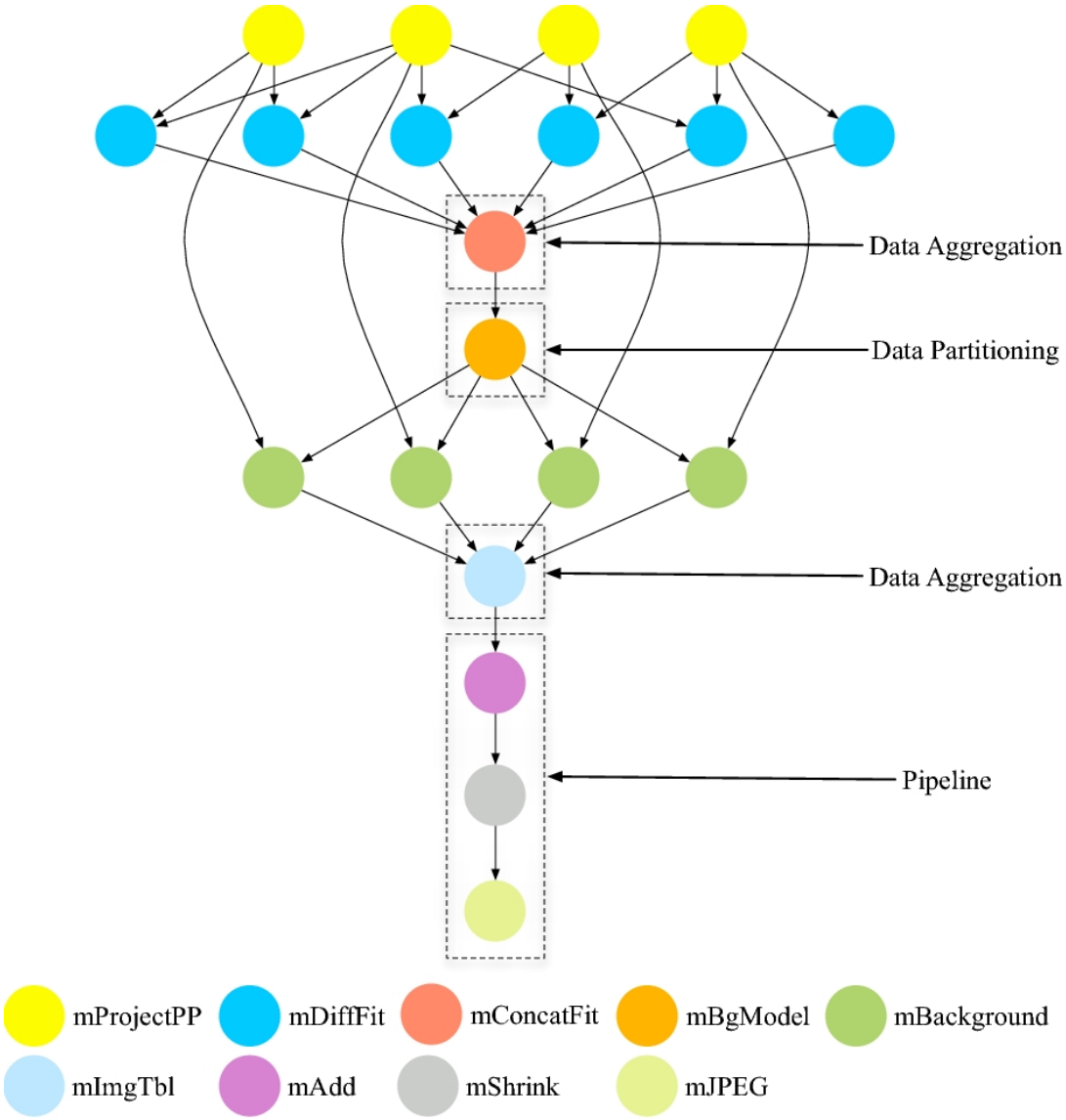


Figure 1.2: Montage workflow structure. Each node represents up to a hundreds of tasks. (Source: [26])

2. Review existing solutions.
3. Design adaptive algorithm.
4. Implement adaptive algorithm.
5. Verify adaptive algorithm.
6. Summarize results and define future work.

1.6. Summary

This chapter provides motivation to conduct this work followed by a short introduction to cloud computing and scientific workflows. After that there is a problem statement and goal of thesis with defined tasks provided.

2. State of the Art Overview

This chapter introduces scientific workflows (section 2.1) which are the subject of this work. Then there is a a description of linear programming (section 2.2) used in work to model and solve goal of thesis. At the end (section 2.3) there are presented related works in this area.

2.1. Workflow Overview

As mentioned in introduction section, workflows are basically series of tasks. From formal point of view, workflow can be presented as a *Directed Acyclic Graph* where node represents one task and edge dependency between tasks. On the basis of dependencies between tasks it is possible to group them into levels. Each level contains independent tasks which can be executed in parallel. This fact gives more opportunities to optimize workflow execution by executing in parallel tasks from the same level.

Such workflows can be classified according to tasks number (even up to 1,000,000 number of tasks), width (in wide workflow there are sets of tasks which can be executed in parallel, in narrow one by one) or tasks size in relation to pricing unit time: fine-grained (task execution time is lower then pricing unit time) or coarse-grained (task execution time is longer than unit time).

2.2. Linear Programming

2.2.1. Definition

Linear programming is a optimization method which allows to achieve the best solution (minimum or maximum) represented in a mathematical model as linear relationships.

Linear programming consists of linear function to be maximized (or minimized) known as *an objective function*, problem constraints and non-negative variables.

Linear programs (are problems that) can be expressed in canonical form as:
maximize $c^T x$
subject to $Ax \leq b$
and $x \geq 0$

where:
x represents the vector of variables (to be determined)
c and b are vectors of known coefficients

A is a known matrix of coefficients
and $(.)^T$ is the matrix transpose
$c^T x$ is a *objective function*
inequalities are the constraints which specify the search space over which the objective function is to be optimized.

Linear programming can be applied in many areas of science and engineering.

Mixed integer linear programming (MIP) is a variant of linear programming in which only some of the variables are constrained to be integers, while other variables are allowed to be non-integers.

2.2.2. Example

One of the popular examples of usage is a planning production in a factory to minimize total cost under given deadline.

A Factory which produces three different products: A, B and C will be provided as an example. They can be produced on two assembly lines: I (faster and more expensive) and II (slower and cheaper). There is an order for N_A A products, N_B B and N_C C ones. Client gives deadline T to complete the order. From factory point of view crucial information is a plan describing which (and how many) products should be produced on each assembly line to minimize total cost under given deadline.

Assembly line	I	II
Cost/hour	C_I	C_{II}

Product/Assembly line	I	II
A	E_I^A	E_{II}^A
B	E_I^B	E_{II}^B
C	E_I^C	E_{II}^C

(a) The left table presents cost per hour C_l for line for l assembly line, on the right there is an efficiency per hour E_l^p for l line and p product

Product	Order amount
A	N_A
B	N_B
C	B_C

Product/Assembly line	I	II
A	X_I^A	X_{II}^A
B	X_I^B	X_{II}^B
C	X_I^C	X_{II}^C

(b) On left side there is a product order, on right production plan - variable X_l^p represents how many products p will be produced on line l

Figure 2.1: Tables above represent available assembly lines and performance, then ordered products and unknown variables.

The objective is to minimize total cost:
$min \sum_{l}^{L} \sum_{p}^{P} C_l * X_{l,p}$
Constraints are:
to meet deadline: $\sum_{l}^{L} \sum_{p}^{P} E_{l,p} * X_{l,p} <$ T
amount of the ordered products: $\forall p \in P \sum_{l}^{L} X_l^p = N_p, \quad \text{where} \quad P = \{A, B, C\} \quad \text{and} \quad L = \{I, II\}$

and X_l^p variables non-negative: $\forall x \in X : x \geq 0$

There are a few ways of solving the problem defined above. But this is out of scope of this work. Worth to comment is that such solution may not exist at all. Basically it could be impossible to complete the order under specified deadline.

2.3. Related Work

Mathematical programming has been applied to the problem of workflow scheduling in clouds. The model presented in [11] 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 [12]. The cloud bursting scenario described in [13], where a private cloud is combined with a public one, also addresses workflows. None of these approaches address the problem of inaccurate estimates of actual task runtimes.

Adaptive approach is known in engineering systems [14]. Dynamic algorithms for workflow scheduling in clouds have been proposed e.g. in [17], where there is an assumption of the dynamic stream of workflows. In [15] the goal is to minimize makespan and monetary cost, assuming an auction model, which differs from this approach where there is an assumption of cloud pricing model of Amazon EC2. In [6] there are presented scheduling workloads of workflows with unknown task runtimes - here also workflows are treated as stream.

MIP approach to schedule multi-level workflows is used in work [1], but the dynamic nature of cloud is not considered. In other works there were analysis of impact of uncertainties of runtime estimations on the quality of scheduling bag-of-task in [3] and workflow ensembles in [2], with the conclusion that these uncertainties cannot be always neglected.

Uncertainties which may come almost from all sides are described in details in [7].

Task estimation for workflow scheduling is a non-trivial problem, but several approaches exist, e.g. those based on stochastic modeling and workflow reductions [8]. It is also possible to create performance models to estimate workflow execution time using application and system parameters, as proposed in [16]. The error of these estimates is less than 20% for most cases, which gives a hint on the size of possible uncertainties.

2.4. Summary

In this chapter there is information about scientific workflows and then a short introduction to linear programming with an example. At the end related work papers are listed.

3. Adaptive Algorithm

This chapter describes adaptive algorithm for multi-stage optimization of workflow executions in IaaS clouds under a deadline constraint. At the beginning there is a general description (sections 3.1 and 3.2), then input and output are described (section 3.3). After that there is a presentation of high-level flow with a detailed description of each phase (sections 3.4 and 3.5). This is followed by execution of sample workflow with comments to each step (section 3.6). At the end optimization models used in algorithm (section 3.7) are presented .

3.1. Introduction

Algorithm provides an adaptive method for minimizing cost of workflow execution on IaaS clouds under a deadline constraint. Algorithm is run many times during workflow execution. Each execution is just one iteration which consists of two phases: *global planning phase* and *local planning phase*. In first phase the algorithm approximately assigns VMs to whole workflow under deadline constraint. As a result of a second phase it returns detailed assignments between VM and each task - but only for the first available subset of independent tasks (level). Then tasks from this subset are executed. After that algorithm updates remaining deadline with real execution time and starts next iteration. Thanks to that it is able to adjust to differences between an estimated and actual execution time. This approach can be considered as a hybrid between static and dynamic scheduling algorithms.

For example, if we have deadline for 3 days, but in runtime it shows a possibility to finish in 2 days (e.g. when tasks were overestimated) algorithm will minimize total cost by choosing slower and cheaper VMs. And as a result workflow will be executing for 3 days. In second scenario when tasks are underestimated algorithm tries to keep deadline and selects more powerful VMs.

3.2. Assumptions

The main assumption is that workflow can be divided into levels. Tasks from each level are independent and can be executed simultaneously on multiple VMs. Level of task is a length of the longest path from an entry node. What is important, tasks from one level can have different estimates of execution time.

3.3. Input/Output

The algorithm requires:

1. Information about available infrastructure (VMs)
   (a) The performance expressed in metric called CCU (which is a result of a benchmark, as in Cloud Harmony Compute Units [18])unit time (e.g. hour)
   (b) List of available VM instances
2. Workflow (see Fig. 1.1) represented as directed acyclic graph (DAG):
   (a) Nodes represent tasks
   (b) Edges represent dependencies between tasks
   (c) Each task has estimated execution time on a VM with performance of 1 CCU
3. Global deadline for the whole workflow.

3.4. High Level Flow

The algorithm is shown in Fig. 3.1 and consists of the following steps described below.

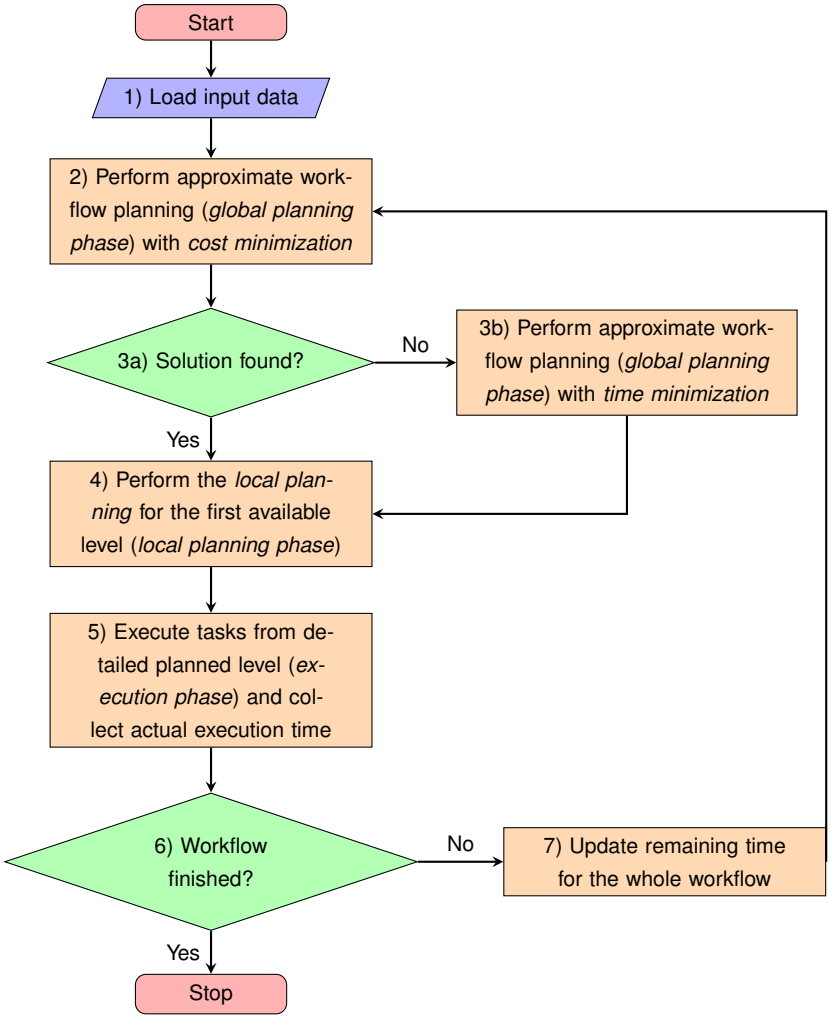


Figure 3.1: High level flow of scheduling algorithm.

3.5. Description of Each Algorithm Step

1. In this step all required data is loaded: information about workflow, available infrastructure (list of VMs) and global deadline. Detailed input is described above in section 3.3.

2. In *global planning phase* algorithm assigns VMs to all not finished levels. Tasks estimated execution time could have different values in the same level which makes calculation complex and time consuming. To simplify optimization model used in this phase, algorithm calculates average tasks execution time for each level and uses as input. The goal of this phase is to find assignments with minimal total execution cost under global deadline. As a result algorithm returns which VMs will execute tasks from given level and also how many tasks should be executed on each VM. Additionally algorithm returns information about estimated execution time and cost for each level.

3. Next step is introduced to check if the solver finds a solution. If yes, then algorithm goes to the 4th step. If solution is not found (e.g. global deadline is too short), the optimization is run again without deadline constraint, but with time minimization as an objective. Execution cost is skipped in this model. This is one of the possible ways of handling this situation. Returned results are the same as in the previous model with global deadline constraint.

4. *Local planning phase* assigns individual tasks to VMs in the current level. It uses the results from previous step as an input: VMs assigned to this level and number of tasks which should be executed on each VM. In this phase there are used exact task estimates, so tasks could have different estimates (in *global planning phase* algorithm calculates average estimates). 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 level is known – so the cost does not change. As a result the algorithm returns information on which VM task will be executed.

5. In this step tasks from *local planning phase* are executed on assigned VMs. Actual task execution time is collected and used in next iterations. Tasks may be executed on real VMs instances or in a cloud simulator (which allows to easily test many scenarios).

6. The algorithm finishes if there are no remaining levels to be scheduled (all tasks are executed).

7. Remaining total time is decreased by real execution time and then algorithm performs planning for remaining part of the workflow, repeating process from step 2. Thanks to such iterative planning the algorithm adjusts to the current situation of workflow execution.

3.6. Illustrative Example

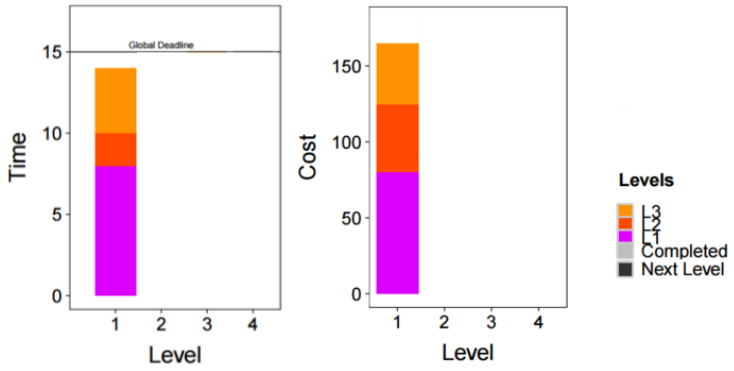
To illustrate the operation of described algorithm, an example below has been prepared using the simple workflow from Fig. 1.1. The input is provided in Table 3.2a. The workflow consists of 3 levels, so the algorithm is executed in three iterations, as shown in Fig. 3.2, Fig. 3.3 and Fig. 3.4. The results are presented and commented in Fig. 3.5.

Task ID	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

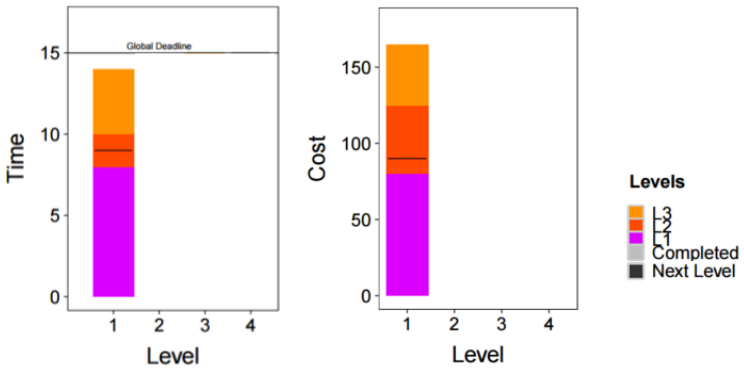
(a) Input to the illustrative example algorithm execution: estimated task sizes, VM performance and costs for the workflow shown in Fig. 1.1. Global deadline is 15.

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



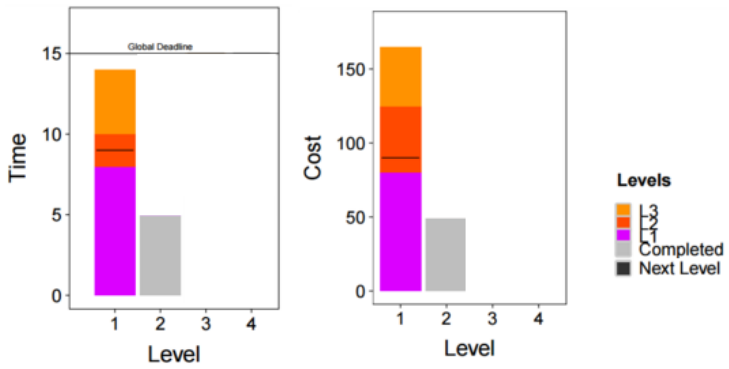
(b) Iteration 1, *Global planning phase*. Estimated cost of executing workflow is 165 and the total time is 14. Algorithm plans to use almost all of the available time, selects cheaper instance (A) and minimizes the total cost.

Task ID / Level	T1	T2	L1
VM	A	A	
Planned Time	5	4	9
Planned Cost	50	40	90



(c) Iteration 1, *Local planning phase*. In this phase algorithm uses exact values of task estimates - because of that total execution time and cost is different from global phase for this level (8 vs. 9). Planned time for level is 9, planned cost is 90.

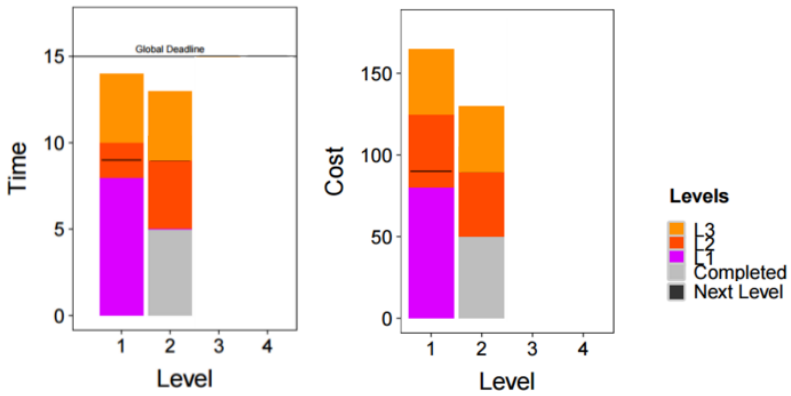
Task ID / Level	T1	T2	L1
VM	A	A	
Actual Time	3	2	5
Δ Time	-2	-2	-4
Actual Cost	30	20	50
Δ Cost	-20	-20	-40



(d) Iteration 1, *Execution phase*. In this case tasks from level 1 were overestimated - execution time was shorter than estimated (planned 9, actual 5).

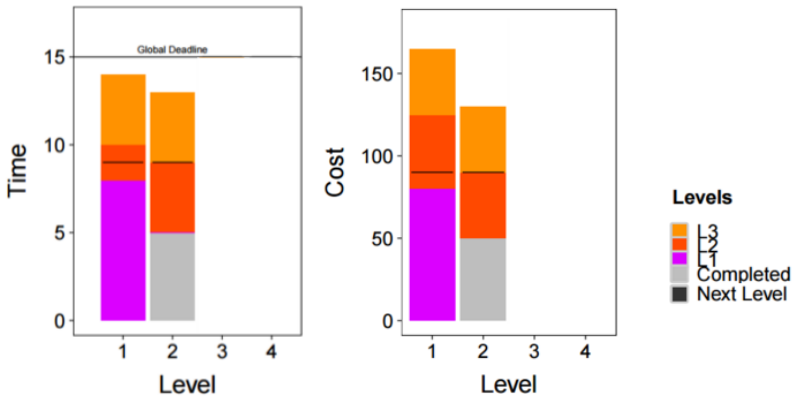
Figure 3.2: 1^{st} iteration. In *Global planning phase* there are all 3 levels used, in *local planning phase* the first one - L1. Execution took less time than estimated. Algorithm updates remaining time with actual value (now 10 time units remained) and current total cost is 50.

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



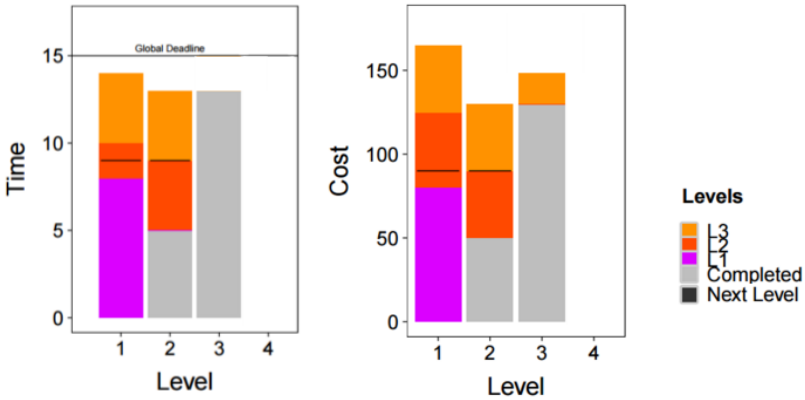
(a) Iteration 2, *Global planning phase*. Due to more time available than expected, algorithm assigned all tasks to the cheapest VM - to minimize the total cost. Previously there was selected VM A and B to level L2, now VM A is assigned to both tasks from level L2.

Task ID	T3	T4	L2
VM	A	A	
Planned Time	2	2	4
Planned Cost	20	20	40



(b) Iteration 2, *Local planning phase*. Planned time for level is 4, planned cost is 40.

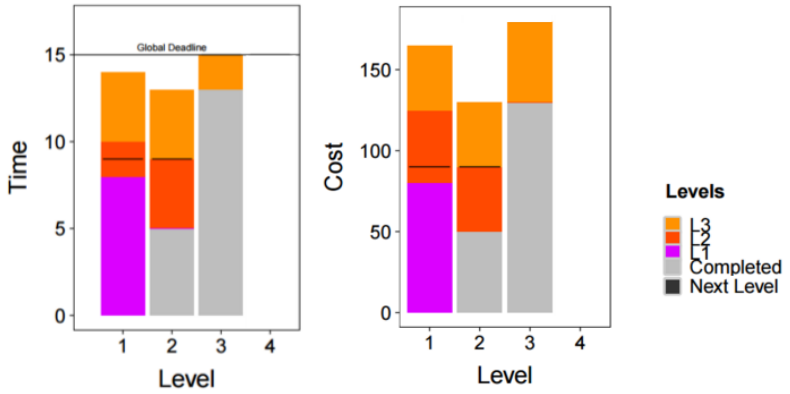
Task ID	T1	T2	L2
VM	A	A	
Actual Time	4	4	8
Δ Time	+2	+2	+4
Actual Cost	40	40	80
Δ Cost	+20	+20	+40



(c) Iteration 2, *Execution phase*. Tasks from level L2 were underestimated - execution time was longer than expected. Actual level execution time is 8 and cost 80.

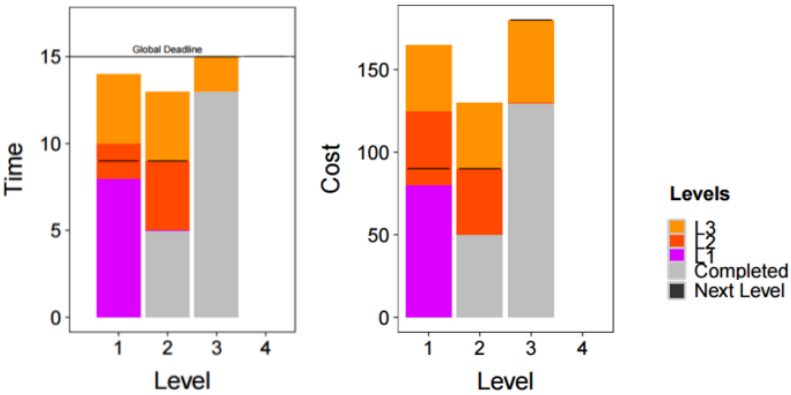
Figure 3.3: 2^{nd} iteration. In the first phase algorithm selects cheaper VMs due to more available time comparing to the first iteration (L1 was overestimated). *Execution phase* shows that L2 was underestimted. Total cost is 130 and remaining time is 2.

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



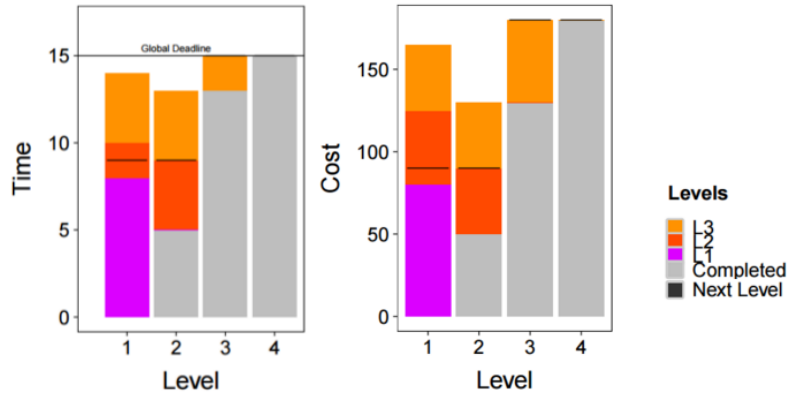
(a) Iteration 3, *Global planning phase* - algorithm selects the most powerful VM B to keep the deadline constraint.

Task ID	T5
VM	B
Planned Time	2
Planned Cost	50



(b) Iteration 3, *Local planning phase*. Level L3 has only one task so level execution time equals to the task execution time.

Task ID	T5
VM	B
Actual Time	2
Δ Time	-
Actual Cost	50
Δ Cost	–



(c) Iteration 3, *Execution phase* - in this case estimate for T5 was exact.

Figure 3.4: 3^{rd} iteration. Before this iteration L3 was assigned to VM A, but due to less available time for L3 then before (due to underestimated L2) in *global planning phase* the algorithm selects the most powerful VM B to keep the deadline constraint. Total time is 15, total cost is 180 and the workflow completed in the given deadline.

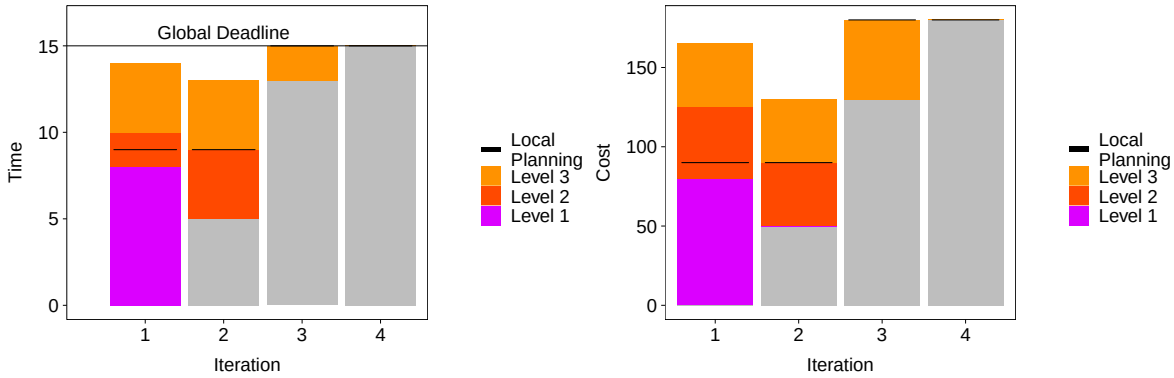


Figure 3.5: Execution time and cost of the algorithm, shown level by level. In the first iteration, the global planning phase estimates the completion time of level 1 as 8 (purple bar) and the local planning estimates it to be 9 (solid line). In iteration 2, it turns out that the level L1 finished at time 5 (grey bar). Both global and local planning for level 2 (red bar and solid line) predict the finish time to be 9. The actual execution of level 2 completes at 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. As we see the assignments of tasks to VMs change whenever the actual execution time differs from the estimated one.

3.7. Optimization Models

Algorithm uses three optimization models: 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 phase*. Since the domain is discrete, each model belongs to a mixed-integer programming (MIP) class. In all three models there is an assumption of simplicity that VMs start immediately and have no latency. Thanks to that the problems are solved quicker. However, there is an assumption that all possible delays are included in the error of estimates, which is taken into account in step 7 of the algorithm. Below there is a description of the models with reference to the source code in the public repository [29]

3.7.1. Global Planning Phase Model

Model used in *global planning phase* assigns VMs and sub-deadlines to each level. Sub-deadlines are evaluated during computation based on global deadline for the whole workflow. 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 given VM. The objective is to minimize total cost of the whole workflow execution.

Input to this *global planning phase* is defined by 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 by the following variables:

- $Q_{l,v}$ is integer matrix which provides 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: $\sum_{l}^{L}\sum_{v}^{V} C_{l,v} * Q_{l,v}$

Search space is constrained to keep the total execution time below the deadline, divide the deadline into sub-deadlines and to enforce them, and to ensure that all the tasks from each level are executed. This is achieved by following constraints:

- enforce workflow deadline:
  $\sum_{l}^{L} T_l^e \leq d$
- determine VM working time for each level:
  $\forall l \in L,\ \forall v \in V : T_{l,v}^v = Q_{l,v} * T_{l,v}^a$
- determine time for level (sub-deadline) - which is equal to the longest working vm time:
  $\forall l \in L,\ \forall v \in V : T_l^e \geq T_{l,v}^v$
- make sure that execution time for vm in level is less or equal to level execution time:
  $\forall l \in L : T_l^e \leq \sum_{v}^{V} T_{l,v}^v$
- enforce that all tasks from level are executed:
  $\forall l \in L : \sum_{v}^{V} Q_{l,v} = L_l$
- enforce that values from matrix Q are not negative:
  $\forall l \in L,\ v \in V : Q_{l,v} \geq 0$

3.7.2. Global Planning Phase Alternative Model

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 and global deadline for workflow can not be met. In comparison to the previous model, the algorithm ignores global deadline constraint and the objective function minimizes total time of workflow execution:

Minimize: $\sum_{l}^{L} T_l^e$

Defined constraints are almost the same as in *global planning phase* - all are used without the first one (enforce global deadline)

3.7.3. Local Planning Phase Model

Model used in *local planning phase* assigns one VM (from VMs assigned to level) 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.

Input to this optimization problem is defined by 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 by the following variables:

– $A_{k,v}$ is binary matrix which provides if task k will be executed on VM v

– T^r_v is vector of real numbers which provides how long each VM v works

– 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.

Search space is constrained 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. This is achieved by the following constraints:

– determine working time for each VM:
$\forall v \in V : \sum_{k}^{K} T^e_{k,v} * A_{k,v} = T^r_v$

– determine time of the longest working VM:
$\forall v \in V : T^r_v \leq w$

– enforce number of tasks assigned to each VM:
$\forall v \in V : \sum_{k}^{K} A_{k,v} = N_v$

– enforce that each task should be executed:
$\forall k \in K : \sum_{v}^{V} A_{k,v} = 1$

3.8. Summary

This chapter describes adaptive multi-stage algorithm for minimizing cost of workflow execution on IaaS clouds under deadline constraint. At the beginning there is a description of algorithm idea - split workflow to levels and execute two-phase planning for each level. Flow is described in Fig. 3.1 and then each step is explained. Illustrative example presented in Fig. 3.2, Fig. 3.3, Fig. 3.4 and Fig. 3.5 allows to easily understand how algorithm works. At the end there is a description of optimization models implemented using mixed-integer programming language.

4. Implementation

This chapter describes implementation of presented algorithm and optimization models. At the beginning there is a high level description (section 4.1) and a description of experiments input (section 4.2). Two flow diagrams (section 4.3) show how experiment runner, workflow planner and executor work. Then there is showed generated output (section 4.4) which includes planning results. At the end there are used languages and tools listed (section 4.5), how to run tutorial (section 4.6) followed by link to public repository with a quick layout description (section 4.7). There is also presented a project organisation and a high level flow of application presented on diagrams. At the end there is a sample output generated by application.

4.1. High level description

Application consists of modules specified below:

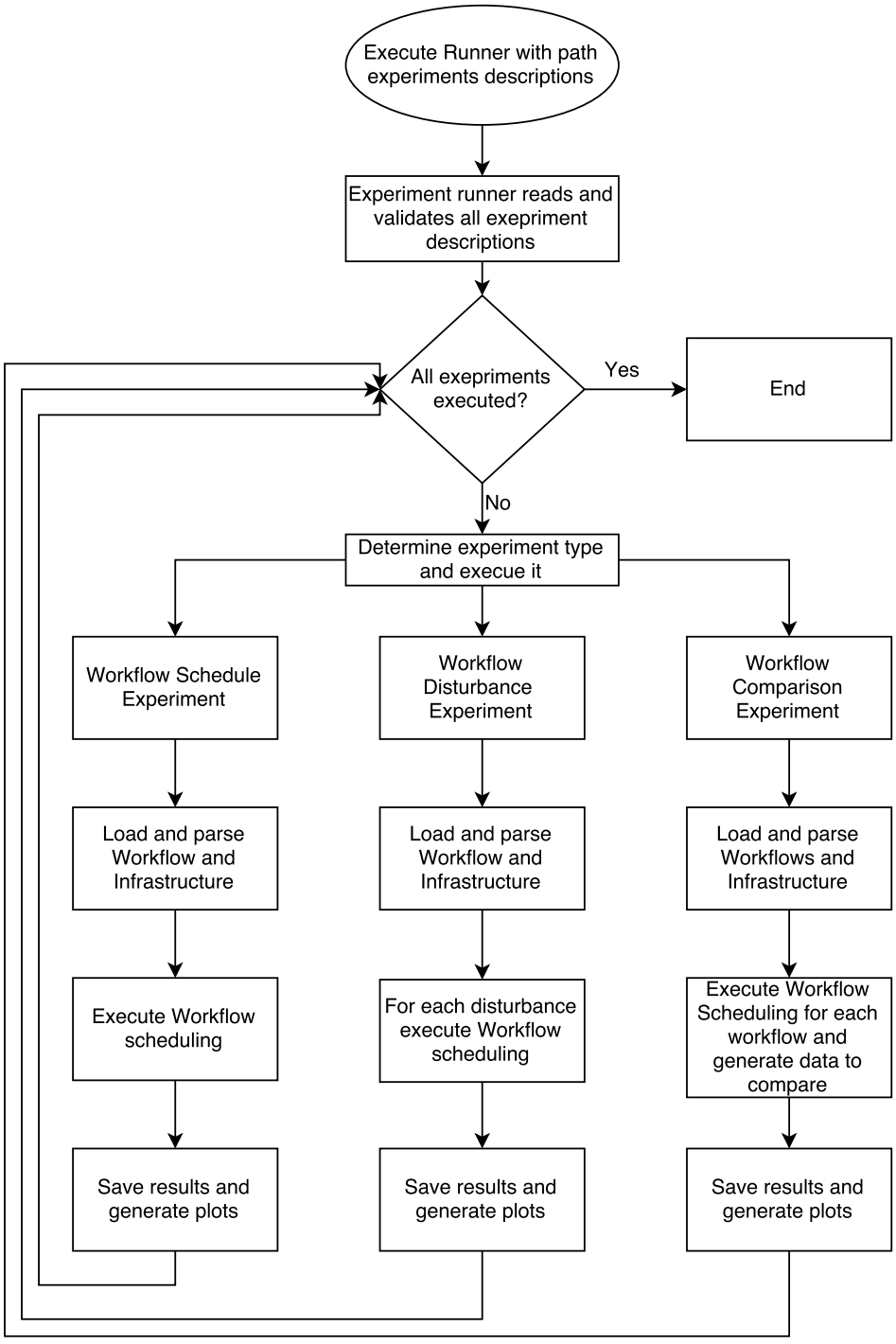
1. Experiment runner
2. Input data loader
3. *Global planning phase* main planner
4. *Global planning phase* alternative planner
5. *Local planning phase* planner
6. Task executor at simulated cloud
7. Output data presenter

4.2. Experiment Input

There are three types of experiments implemented. Except standard workflow planning and executing experiment, application provides two more experiments: workflow comparison and disturbance. Experiments are described in the next chapter 5. Experiments and Results. Experiments inputs are described by .json files which include a lists of experiments. They define .dag files with workflows and .json files with infrastructure descriptions. Sample input file with description is attached in appendix A.1

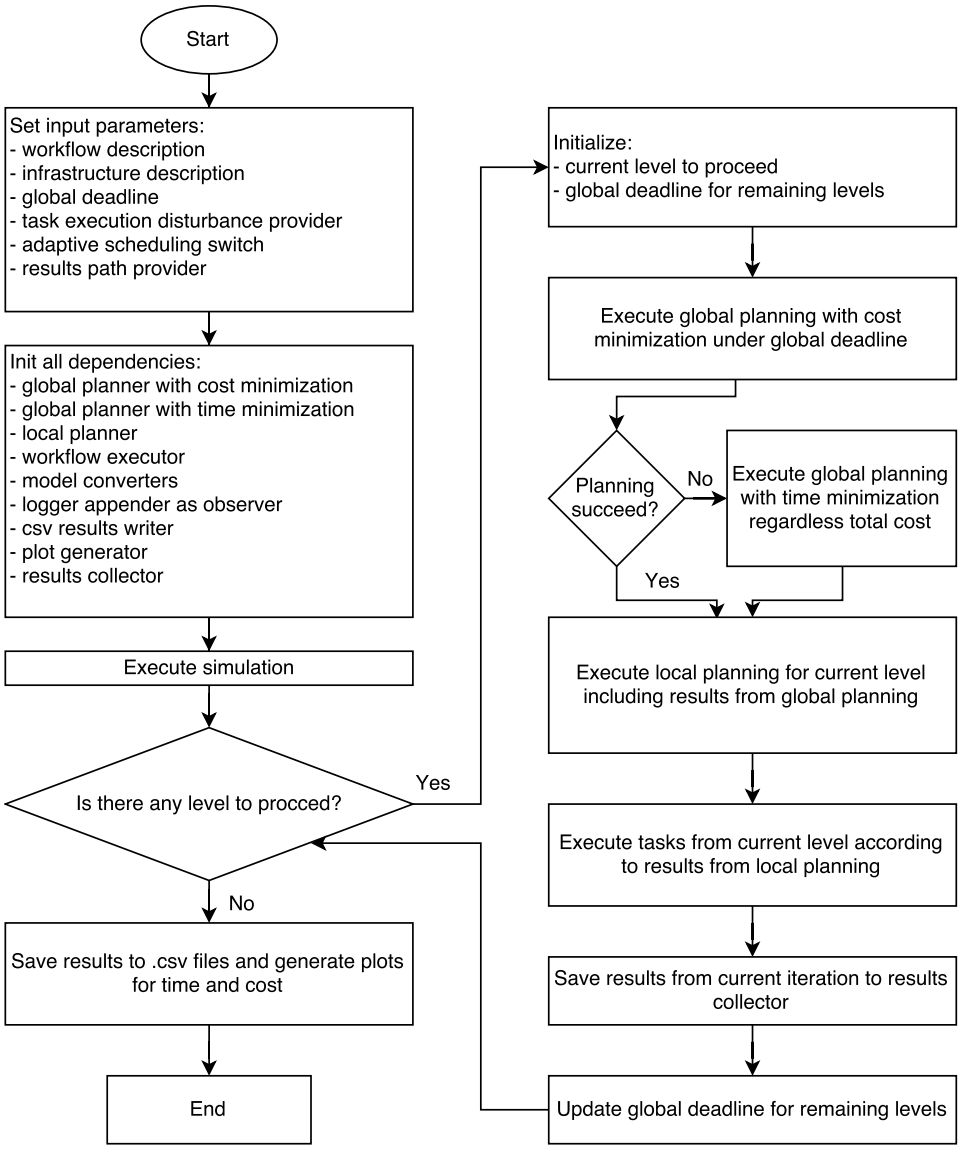
4.3. Flow Diagrams

Whole flow is split into two levels: first one is experiment phase (Fig. 4.1), second one is workflow schedule and execution phase (Fig. 4.2).



(a) Experiment Runner Flow.

Figure 4.1: Experiment Runner Flow - experiments are read and dispatched to proper runner.



(a) Workflow Scheduling and Execution Flow.

Figure 4.2: Workflow Scheduling and Execution Flow. This is implementation of adaptive algorithm described in the previous chapter 3. Adaptive Algorithm. After each step all observers are notified. In current implementation only one observer is registered, which saves all notification and results in the file. Thanks to this approach it is easy to implement other ones, e.g. gui which could visualize whole experiment in the real time.

4.4. Generated Output

During execution of the whole experiment output is saved in files. Such files (group in directories) are generated:

```
results
  {experimentName}/
    {timestamp}/
      {workflowDescription}/
```

```
{date}/
  cmpl/ {raw solution from cmpl}
    iteration-0-global-main
    iteration-0-local
    iteration-1-global-alternative
    iteration-1-local
    iteration-2-global-main
    iteration-2-local
  csvResults/ {data for plot generation}
    schedule_cost_illustrative.csv
    schedule_time_illustrative.csv
  plots/ {generated plots}
    schedule_cost_illustrative.pdf
    schedule_time_illustrative.pdf
  execution.log {experiment execution data}
```

execution.log file contains:

1. Information about input data.
2. Information when each action was performed and finished.
3. Results of each planner execution (results from *global planning phase* and *local planning phase*).
4. Iteration summary.
5. Timestamps before each log entry.

Generated output is presented in appendix A.2.

Below you can find the output (without timestamps) from illustrative workflow scheduling experiment, which contains summary of input data, information which step is executed, results from planner and task executor after each iteration:

4.5. Languages and Tools

Languages and tools used in implementation:

– optimization models are implemented in CMPL modeling language [27]

– solutions are computed by CBC solver [28]

– input data is loaded from files: workflow from DAG, infrastructure from JSON

– plots are generated in R language

– main application and task executor are implemented in Java 8

– all dependencies are handled by gradle (e.g. gson, log4j, junit; details are available in build.gradle file)

4.6. How To Run

Requirements:

– java 8

– gradle 2.3

– R-3.1.2 (with libraries: ggplot2, reshape2, grid)

– cmpl (1.10.0) with cbc solver (included in cmpl package)

Both R and cmpl *bin* directories must be included in the *PATH* system environment. Both programs are executed during experiments.

To run experiment execute pl.edu.agh.ki.mgr.workflowplanner.app.ExperimentRunner class with experiment description file as program argument, e.g.
pl.edu.agh.ki.mgr.workflowplanner.app.ExperimentRunner experiments/disturbance/disturbance-montage5000.json

4.7. Source Code

Source code (including R scripts and sample test data) is available in the repository [29]. Repository layout (all files are stored under main directory WorkflowPlanner):

– *experiments* - contains test data for experiments (workflows, infrasturucture, experiments definitions)

– *lib* - contains jCMPL.jar dependency

– *models* - contains optimization models written in cmpl language

– *RScripts* - contains scripts which generate plots

– *src* - contains source code including unit tests

4.8. Summary

This chapters presents implementation of the algorithm and the optimization models. There are high level flows, requirements to run experiment, sample input and generated output. Then used languages and tools are listed. At the and there is a link to public repository with source code.

5. Experiments and Results

This chapter describes performed experiments which evaluated the algorithm. Section 5.1 shows performed experiments and then section 5.2 describes testing environment including used VMs and hardware. Then each workflow is characterized (section 5.3) and next sections present results of the experiments.

5.1. Experiments Description

Algorithm implementation has been evaluated by performing a series of experiments. Platform which allows to execute experiments contains implementation of presented algorithm and simple cloud simulator which executes tasks on VMs. Experiments have been performed on different workflows and on different VMs.

Algorithm has been evaluated in three experiments. First one is planning workflow execution on available infrastructure with given deadline. Tasks runtimes are modified in a different way: random, with task under and over estimations. Second one is workflow disturbance experiments which show how time and cost change depending on runtime disturbance. The last one is workflow comparison which compares relative cost and time execution of different workflows. Detailed description is in the sections below.

5.2. Experiments Environment

Experiments have been executed on ASUS K55VM laptop with:

– Intel Core i7-3610QM CPU @ 2.30GHZ

– 8GB RAM

– SSD hard drive

– Windows 7 64bit

Average experiment execution time took from a few seconds up to a few minutes.

Available Amazon VMs have been used as an infrastructure in the experiments. VM *m3.large* is used as a reference VM type. For performance estimation of other instance type is used the ECU value as provided by Amazon.

Tasks are executed in a simple simulator. It executes one level of tasks on the assigned VMs and introduces runtime variations of task execution times to simulate the behaviour of the real infrastructure. It is possible to configure runtime variations so that they allow to easily test different scenarios.

Used VMs:

VM	CCU	Price
t2.micro	1	2
t2.small	1	3
t2.medium	1	6
m3.medium	3	8
m3.large	6	17
m3.xlarge	13	33
m3.2xlarge	26	67
c4.large	8	15
c4.xlarge	16	30
c4.2xlarge	31	59
c4.4xlarge	62	118
c4.8xlarge	132	237
c3.large	7	13
c3.xlarge	14	26
c3.2xlarge	28	52
c3.4xlarge	55	103
c3.8xlarge	108	206

Runtime disturbances have been 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 some experiments in order to simulate shorter/longer execution time, disturbance has been multiplied by 0.75 or 1.25 respectively. In some cases disturbance was equal to zero which means that real execution time equals estimated one.

5.3. Workflows Description

Workflows used in experimets are taken from workflow gallery available in pegasus gallery. Each of them can be split into independent levels. They are described below.

5.3.1. Montage

Montage is a workflow [10] created by NASA/IPAC to create mosaics of the sky from many input images. In experiments version with 5000 tasks (11 levels) has been used. Task runtime estimations have been taken from logs from previous workflow execution on real Amazon infrastructure [4]. Since the real Montage workflow consists of very small tasks (having execution time in the order of seconds), they

have been artificially extended by multiplying execution time by 3600. The deadline has been set to 3500 time units (hours).

5.3.2. Cybershake

Cybershake workflow is used by the Southern California Earthquake Center to characterize earthquake hazards in a region. In experiments version with 1000 tasks (4 levels; respectively 6, 496, 497, 1 task in level) has been used. In fact, only two levels in this workflow could be easily run in parallel. Deadline has been set to 2000 hours (units).

5.3.3. Genome

Genome is another workflow. In experiments version with 400 tasks (9 levels) has been used. Deadline has been set to 3200 hours (units).

5.4. Workflow Scheduling Experiments

This experiment is the base one. It uses the following input:

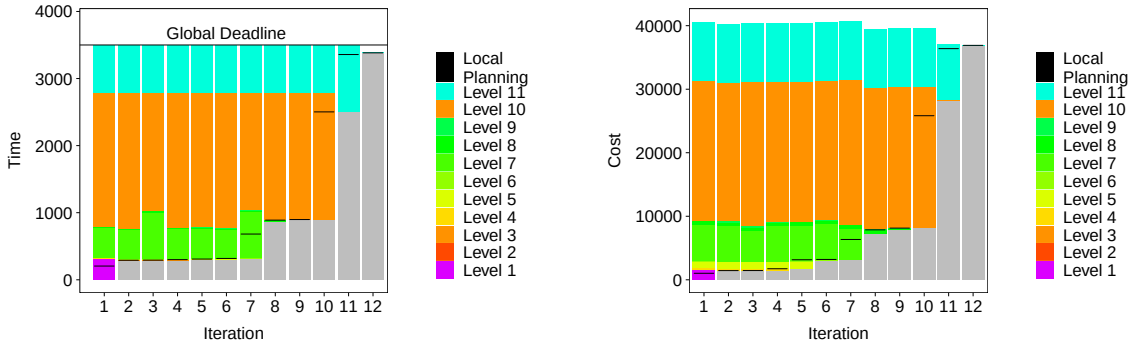
- workflow
- infrastructure
- global deadline
- type of runtime disturbances provider
- flag if real execution time should be taken while updating remaining global deadline (adaptive or static version)

and then executes the whole algorithm. As results, the application generates two plots - the same as in the illustrative example (Fig. 3.5)

Following subsections contain results for workflows described above. Each workflow has been executed with four different parameters set:

- disturbance - random, adaptive - true
- disturbance - random, adaptive - false
- disturbance - real runtime shorter with disturbance (tasks are overestimated), adaptive - true
- disturbance - real runtime longer with disturbance (tasks are underestimated), adaptive - true

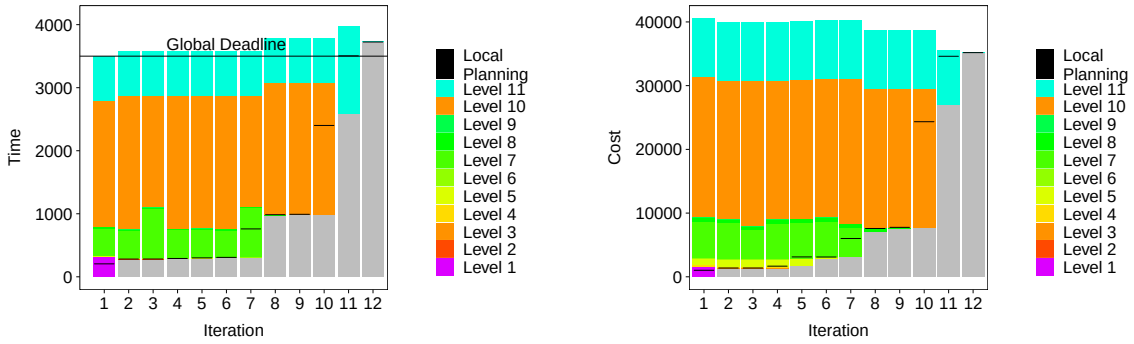
5.4.1. Montage



(a) Workflow execution time.

(b) Workflow execution cost.

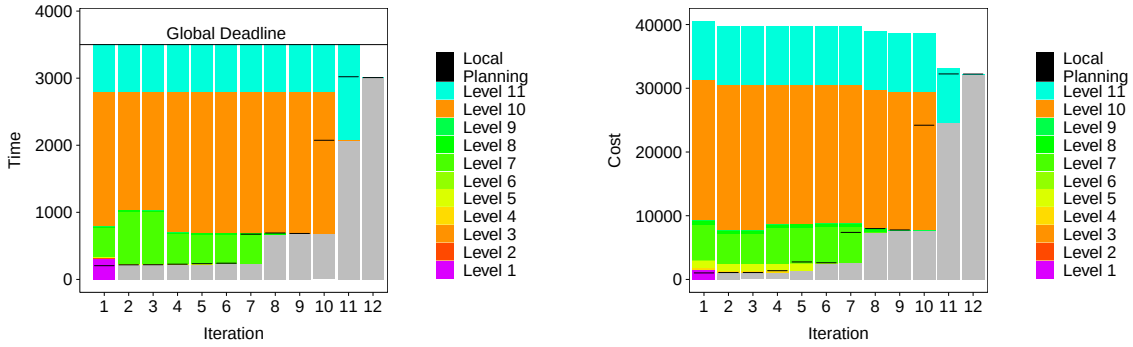
Figure 5.1: [Montage] Workflow planning and execution with random disturbance. As we can see algorithm plans workflow to stick to given deadline regardless of real task execution time. Estimated cost changes after each iteration because different VMs are selected.



(a) Workflow execution time.

(b) Workflow execution cost.

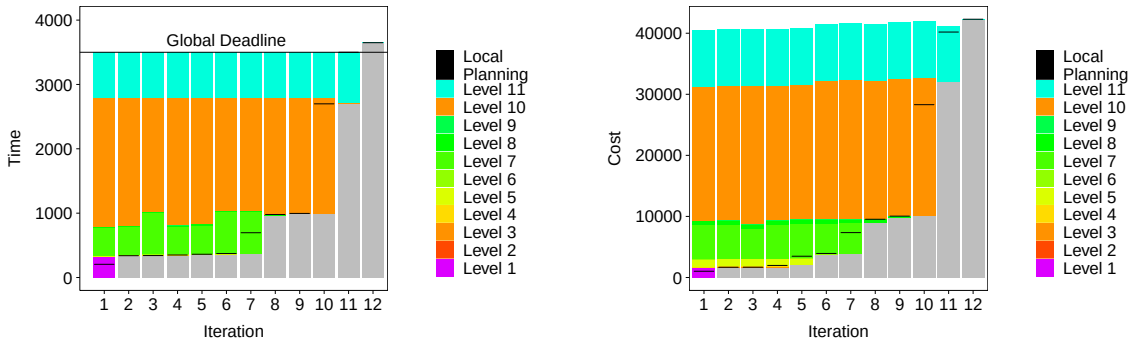
Figure 5.2: [Montage] Workflow planning and execution with random disturbance but in *static mode* - algorithm does not update remaining deadline with real time but plans in advance. As we can see algorithm plans workflow to minimize cost but starting from second iteration it exceeds the given deadline.



(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.3: [Montage] Workflow planning and execution when runtimes are shorter than estimated (case with tasks overestimation). As we can see algorithm minimizes total cost by choosing slower and cheaper VMs. At the same time it uses all available given deadline and sticks to it.



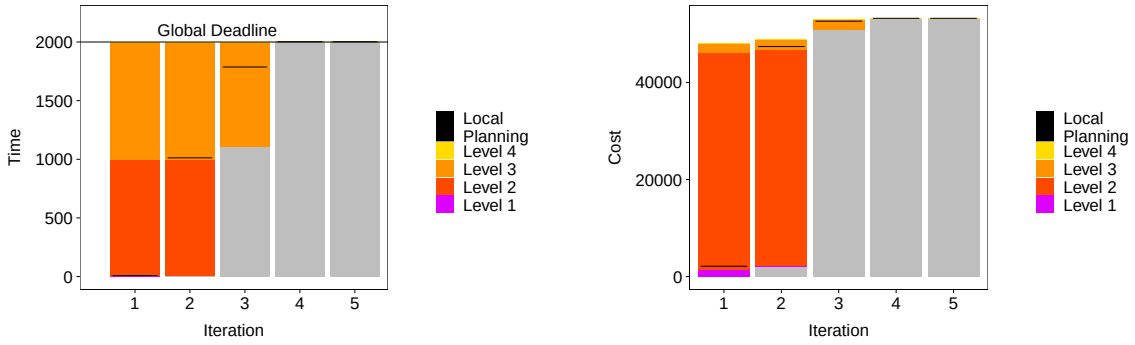
(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.4: [Montage] Workflow planning and execution when runtimes are longer than estimated (case with tasks underestimation). As we can see algorithm tries to keep given deadline by choosing more powerful VMs (and more expensive). In fact deadline has not been met in the last iteration, but it was impossible in a current implementation of algorithm.

5.4.2. Cybershake

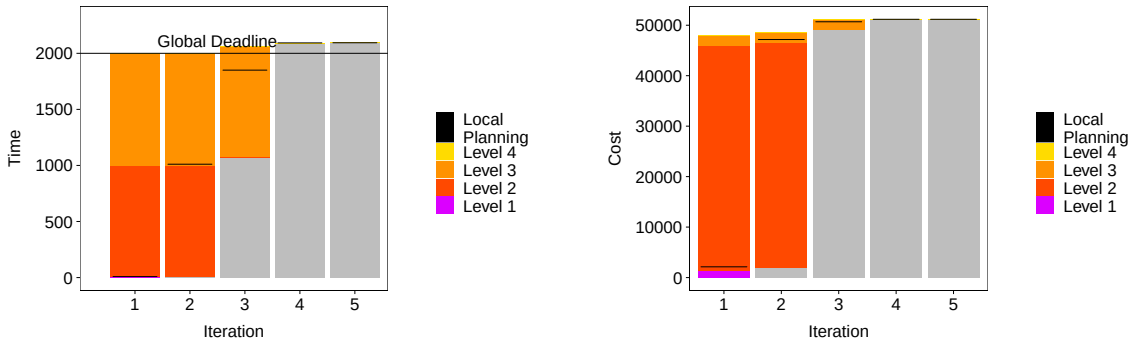
Results for Cybershake



(a) Workflow execution time.

(b) Workflow execution cost.

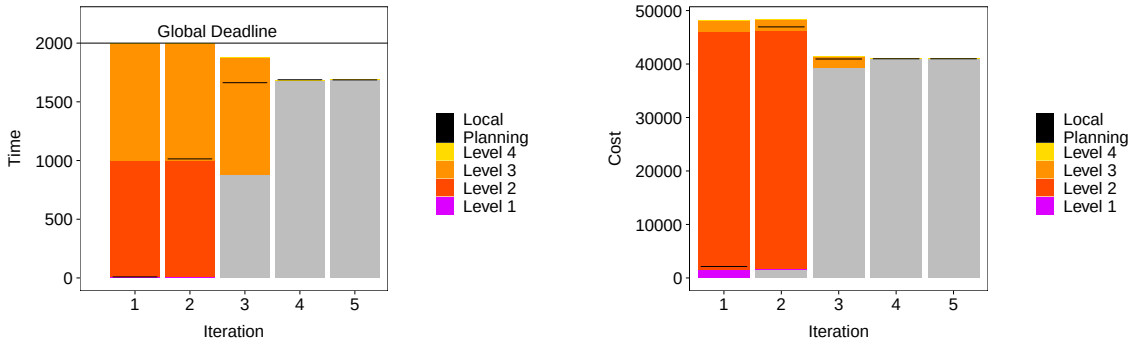
Figure 5.5: [Cybershake] Workflow planning and execution with random disturbance. As we can see algorithm plans workflow to stick to given deadline regardless of real task execution time. Estimated cost increases after each iteration because more expensive VMs are selected.



(a) Workflow execution time.

(b) Workflow execution cost.

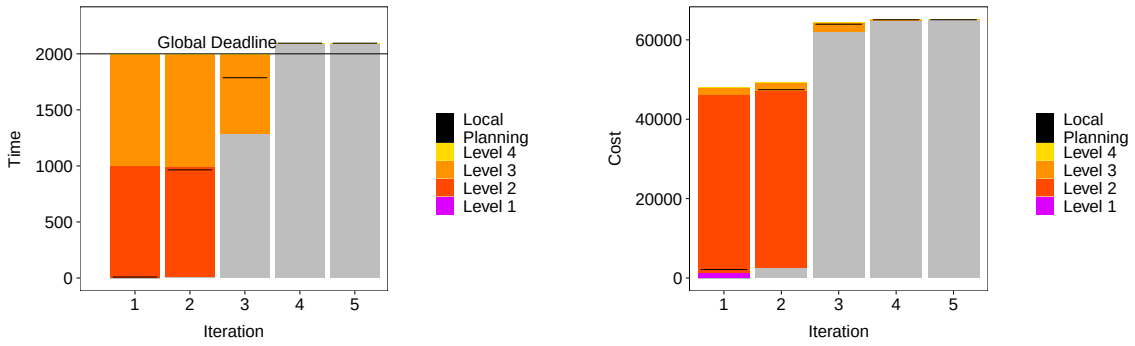
Figure 5.6: [Cybershake] Workflow planning and execution with random disturbance but in *static mode* - algorithm does not update remaining deadline with real time but plans in advance. As we can see algorithm plans workflow to minimize cost but starting from third iteration it exceeds given deadline.



(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.7: [Cybershake] Workflow planning and execution when runtimes are shorter than estimated (case with tasks overestimation). As we can see algorithm minimizes total cost by choosing slower and cheaper VMs. At the same time it uses whole available given deadline and sticks to it in 1st and 2nd iteration. Starting from third iteration total time is lower than then deadline. At the same time total cost is lower comparing to estimation from the first iteration.



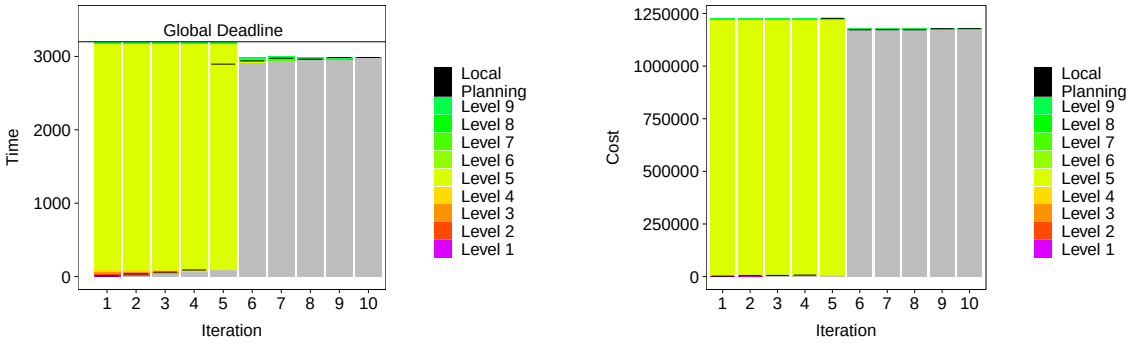
(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.8: [Cybershake] Workflow planning and execution when runtimes are longer than estimated (case with tasks underestimation). As we can see algorithm tries to keep given deadline by choosing more powerful VMs (and more expensive). In fact deadline was not met beacuse it was impossible in a given deadline.

5.4.3. Genome

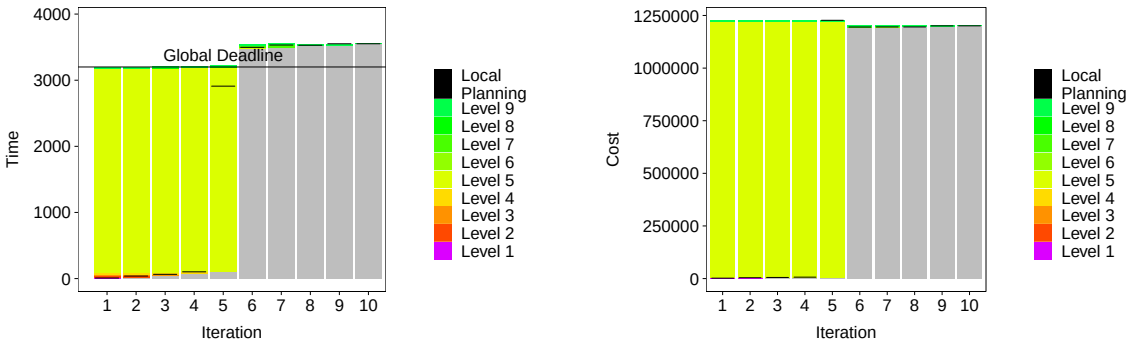
Results for Genome



(a) Workflow execution time.

(b) Workflow execution cost.

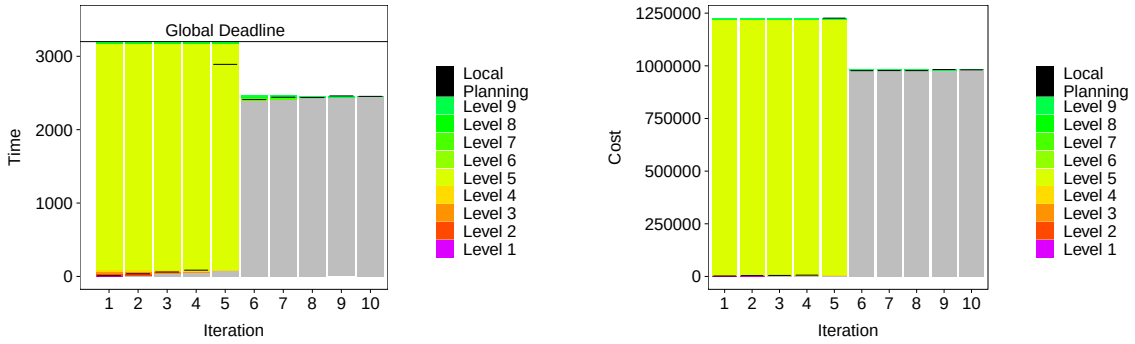
Figure 5.9: [Genome] Workflow planning and execution with random disturbance. As we can see algorithm plans workflow to stick to given deadline in a first part of levels. Then it selects slower VMs.



(a) Workflow execution time.

(b) Workflow execution cost.

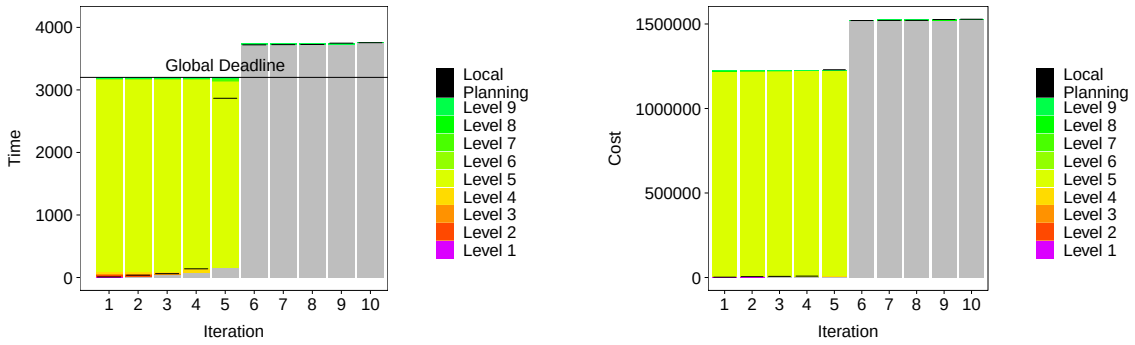
Figure 5.10: [Genome] Workflow planning and execution with random disturbance but in *static mode* - algorithm does not update remaining deadline with real time but plans in advance. As we can see algorithm plans workflow to minimize cost but starting from 6th iteration it exceeds given deadline.



(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.11: [Genome] Workflow planning and execution when runtimes are shorter than estimated (case with tasks overestimation). As we can see algorithm minimize total cost by choosing slower and cheaper VMs. At the same time total cost is lower comparing to estimation from the first iteration.



(a) Workflow execution time.

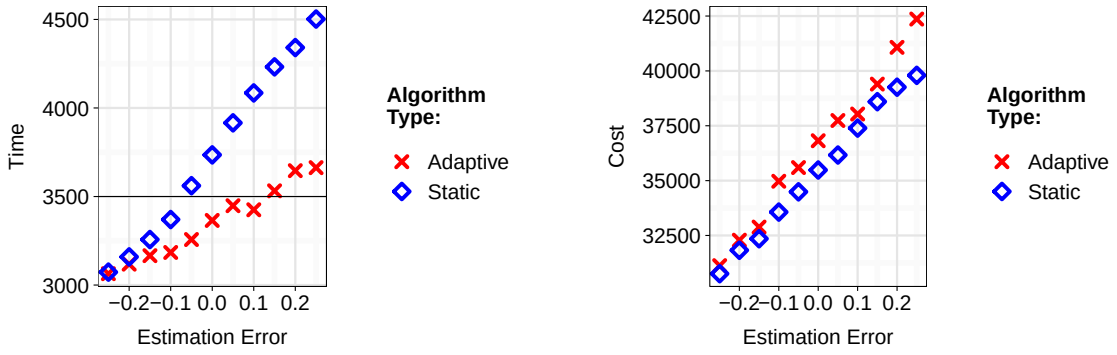
(b) Workflow execution cost.

Figure 5.12: [Genome] Workflow planning and execution when runtimes are longer than estimated (case with tasks underestimation). As we can see algorithm tries to keep given deadline by choosing more powerful VMs (and more expensive). In fact deadline was not met beacuse it was impossible in a given deadline.

5.5. Workflow Disturbance Experiments

This experiment presents how the completion time and total cost depend on the varying estimation error μ. The errors have been 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).

5.5.1. Montage

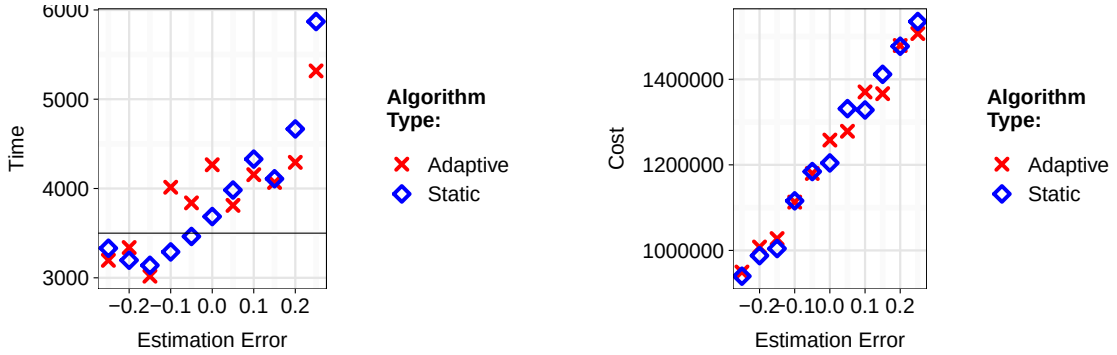


(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.13: [Montage] In plot (a) we observe that 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%. The next 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.

5.5.2. Genome



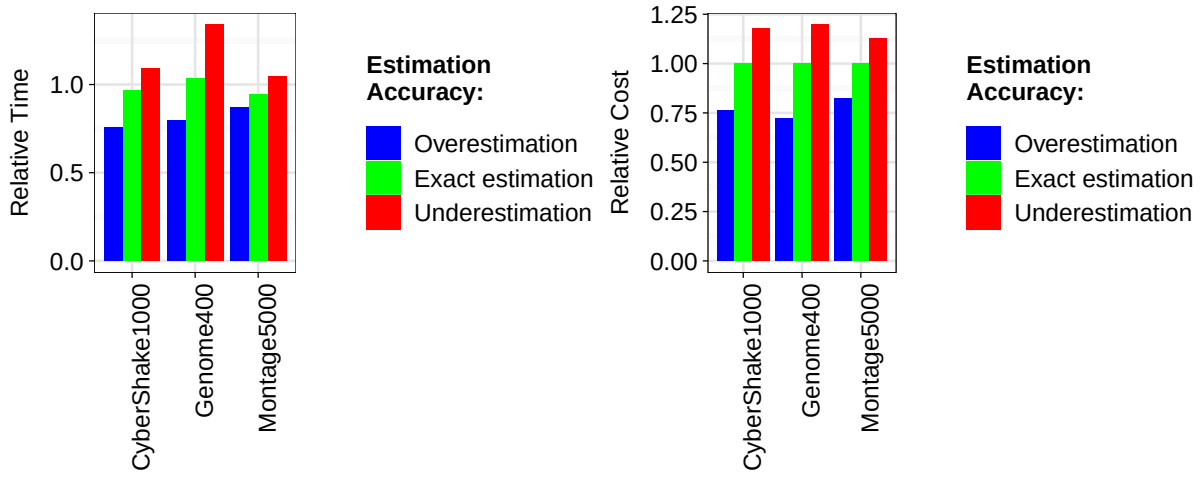
(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.14: [Genome] We can see that this workflow (4 levels, only 2nd and 3rd could be run in parallel) is very susceptible on disturbances. Results are not as clear as in the Montage workflow.

5.6. Workflow Comparison Experiments

These experiments compare behaviour of workflows when tasks are under or over estimated. Generally, observed behaviour is similar in all the cases.



(a) Workflow execution time.

(b) Workflow execution cost.

Figure 5.15: Plots which show that all three workflows behave in a similar way. Overestimation and underestimation represent error distribution shifted by -0.25 and 0.25, respectively. Relative 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.

5.7. Summary

In this chapter presented performed experiments which evaluated planning algorithm are presented. Experiments has been performed on different workflows (each one is described) and with different runtime disturbances. Plots present that algorithm works as expected.

6. Conclusions and Future Work

This work presents the adaptive algorithm for scheduling workflows in clouds with inaccurate estimates of runtimes under global deadline constraint. The results of preliminary evaluation shows that the implemented algorithm works as designed and is able to meet the given deadline while minimizing the cost.

6.1. Accomplished tasks

All tasks defined in 1.5 are accomplished:

1. Problem analysis - chapter 2. State of the Art Overview.
2. Review existing solutions - chapter 2. State of the Art Overview.
3. Design adaptive algorithm - chapter 3. Adaptive Algorithm.
4. Implement adaptive algorithm - chapter 4. Implementation.
5. Verify adaptive algorithm - chapter 5. Experiments and Results.
6. Summarize results and define future work - chapter 6. Conclusions and Future Work.

6.2. Algorithm Summary

The algorithm adapts to the actual situation at runtime:

1. When tasks are executed quicker than estimated – the algorithm selects slower (and cheaper) VMs, and minimizes the total cost.
2. When tasks are executed slower than estimated – the algorithm selects faster (and more expensive) VMs, which increases total cost, but allows not to exceed the deadline for the whole workflow.
3. 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*.
4. When execution of tasks is longer than estimates (which is the worst case scenario) then the total cost increases, but this is a general problem for all scheduling algorithms.

5. When deadline is exceeded (or it is not possible to plan execution under deadline) then the algorithm minimizes the total time regardless of costs.

6.3. Future Work

During implementation and evaluation a few ways that could be enhanced in the future work have been found. They include:

1. Improvement of pricing in optimization models by e.g. reusing already assigned VMs.
2. Extending models with data transfer time and cost (now it is included in task estimates).
3. Splitting levels with many tasks to smaller ones with 'logic' independent levels (see results for Genome and Cybershake workflows).
4. Improvement of task estimation (i.e. take into account multi-core CPUs).
5. Use machine learning to predict estimates errors based on real execution time of previous levels (see workflow comparison experiment and scheduling with tasks underestimation).
6. Use this algorithm as a part of engine to execute workflows in computing clouds, followed by more systematic testing.

6.4. Summary

Scheduling workflows in IaaS clouds with runtimes uncertainties are an interesting challenge. There are different approaches to achieve this. One of them is presented in details in this work which accomplishes given requirements. Presented algorithm is published in Parallel Processing and Applied Mathematics (see appendix B. Publication).

Bibliography

[1] Malawski, Maciej and Figiela, Kamil and Bubak, Marian and Deelman, Ewa and Nabrzyski, Jarek. *Scheduling Multilevel Deadline-Constrained Scientific Workflows on Clouds Based on Cost Optimization*. Scientific Programming, 2015

[2] Malawski, Maciej and Juve, Gideon and Deelman, Ewa and Nabrzyski, Jarek. *Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds*. Future Generation Computer Systems, 2015.

[3] Malawski, Maciej and Figiela, Kamil and Nabrzyski, Jarek. *Cost minimization for computational applications on hybrid cloud infrastructures*. Future Generation Computer Systems, 2013.

[4] Figiela, Kamil and Malawski, Maciej. *Modeling, Optimization and Performance Evaluation of Scientific Workflows in Clouds*. IEEE, 2014.

[5] Bhardwaj, Sushil and Jain, Leena end Jain, Sandeep *Cloud Computing: a study of infrastructure as a service (IaaS)*. International Journal of Engineering and Information Technology (pages 60-63), 2010

[6] Ilyushkin, Alexey and Ghit, Bogdan and Epema, Dick. *Scheduling workloads of workflows with unknown task runtimes*. IEEE, 2015.

[7] Andrei Tchernykh, Uwe Schwiegelsohn, Vassil Alexandrov, El-ghazali Talbi. *Towards Understanding Uncertainty in Cloud Computing Resource Provisioning*. Procedia Computer Science (pages 1772-1781), 2015.

[8] Chirkin, Artem M. and Belloum, A. S. Z. and Kovalchuk, Sergey V. and Makkes, Marc X. *Execution Time Estimation for Workflow Scheduling*. IEEE, 2014.

[9] Deelman, Ewa and others. *Pegasus, a workflow management system for science automation*. Future Generation Computer Systems, 2015.

[10] Juve, Gideon and Chervenak, Ann and Deelman, Ewa and Bharathi, Shishir and Mehta, Gaurang and Vahi, Karan. *Characterizing and profiling scientific workflows*. Future Generation Computer Systems (pages 682–692), 2013.

[11] Genez, Thiago A. L. and Bittencourt, Luiz F. and Madeira, Edmundo R. M. *Using Time Discretization to Schedule Scientific Workflows in Multiple Cloud Providers*. IEEE Sixth International Conference on Cloud Computing (pages 123–130), 2013.

[12] Ruben Van den Bossche and Kurt Vanmechelen and Jan Broeckhove *Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds*. Future Generation Computer Systems, 2013.

[13] Luiz Fernando Bittencourt and Edmundo Roberto Mauro Madeira *HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds*. J. Internet Services and Applications (pages 207–227), 2011.

[14] Abdelzaher, Tarek and Diao, Yixin and Hellerstein, Joseph and Lu, Chenyang and Zhu, Xiaoyun *Introduction to Control Theory And Its Application to Computing Systems*. Performance Modeling and Engineering (pages 185–215), 2008.

[15] Fard, Hamid Mohammadi and Prodan, Radu and Fahringer, Thomas *A Truthful Dynamic Workflow Scheduling Mechanism for Commercial Multicloud Environments*. IEEE Transactions on Parallel and Distributed Systems (pages 1203–1212), 2013.

[16] Pietri, Ilia and Juve, Gideon and Deelman, Ewa and Sakellariou, Rizos *A Performance Model to Estimate Execution Time of Scientific Workflows on the Cloud*. Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science (pages 11–19), 2014.

[17] Mao, Ming and Humphrey, Marty *Auto-scaling to minimize cost and meet application deadlines in cloud workflows*. SC '11, 2011.

[18] CloudHarmony. *What is CCU? CPU benchmarking in Cloud.* `http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html`. CloudHarmony, 2014.

[19] Voorsluys, William and Broberg, James and Buyya, Rajkumar *Introduction to Cloud Computing*. Cloud Computing: Principles and Paradigms (pages 1–41), 2011

[20] Marston, Sean and Li, Zhi and Bandyopadhyay, Subhajyoti and Zhang, Juheng and Ghalsasi, Anand *Cloud computing - The business perspective*. Decision Support Systems (pages 176–189), 2011.

[21] Juve, Gideon and Chervenak, Ann and Deelman, Ewa and Bharathi, Shishir and Mehta, Gaurang and Vahi, Karan *Characterizing and profiling scientific workflows*. Future Generation Computer Systems (pages 682–692), 2013.

[22] Bharathi, Shishir and Chervenak, Ann and Deelman, Ewa and Mehta, Gaurang and Su, Mei Hui and Vahi, Karan *Characterization of scientific workflows*. 2008 3rd Workshop on Workflows in Support of Large-Scale Science, WORKS 2008

[23] Rees, D. G. *Linear Programming, An Introduction.*. The Statistician, 1987

[24] Bisschop, Johannes *Linear Programming Tricks*. AIMMS - Optimization Modeling (pages 63–75), 2006

[25] Amazon *AWS Pricing*. `http://aws.amazon.com/ec2/pricing/`.

[26] Pegasus *Pegasus Workflows Gallery*. `https://pegasus.isi.edu/workflow_gallery/`.

[27] Mike Steglich. *CMPL (Coin Mathematical Programming Language*. `https://projects.coin-or.org/Cmpl` 2015

[28] John Forrest *Cbc (Coin-or branch and cut) open-source mixed integer programming solver*. `https://projects.coin-or.org/Cbc`.

[29] Tomasz Dziok, *Repository with optimization models*. `https://bitbucket.org/tdziok/mgr-cloudplanner`.

A. File Formats and Outputs

A.1. Input files format

This is a sample experiment definition for the scheduling experiment:

```
{
  "workflowScheduleExperiments": [
    {
      "simulatorParameters": {
        "executionTimeModifier": "GaussianShorterExecutionTime",
        "adjustToRealTime": true
      },
      "infrastructureDescription": {
        "filePath": "experiments/infrastructure/awsAllVmTypes.json",
        "name": "All AWS Instances"
      },
      "workflowDescription": {
        "filePath": "experiments/dag/montage5000_real.dag",
        "name": "Montage5000"
      },
      "deadline": 3500,
      "resultsDirectory": "results"
    }
  ]
}
```

It is possible to pass a list of different experiments in one input file. Most of properties are self-descriptive. Worth mentioning are *simulatorParameters*. First one *executionTimeModifier* defines accuracy of task estimates (if they are executed shorter, longer, random, exact, etc.) during execution phase. Second one *adjustToRealTime* says if real execution time should be taken into account when updating remaining deadline at the end of iteration. If set to false then global deadline is decreased by estimated total time from level planning. So in this mode algorithm does not adjust to current task execution time.

Below you can find layout of input files including files with experiments, workflows and infrastructure:

```
experiments/
   comparison/ {comparison experiments}
      comparison-montage.json
   dag/ {workflow description}
      montage5000.json
   disturbance/ {disturbance experiments}
      disturbance-montage.json
   infrastructure/ {infrastructure description}
      awsAllVmTypes.json
   schedule/ {schedule experiments}
      schedule-montage5000.json
```

A.2. Output files format

This is output (without timestamps) from illustrative workflow scheduling experiment, which contains summary of input data, information which step is executed, results from planner and task executor after each iteration:

```
Executing experiment:
SimulatorParameters[
    executionTimeModifier=PaperExecutionTimeModifier,
    adjustToRealTime=true]
InfrastructureDescription[
    filePath=experiments/infrastructure/illustrativeVMs.json,
    name=2 Instances]
WorkflowDescription[
    filePath=experiments/dag/illustrative_2_2_1.dag,
    name=illustrative_2_2_1-toLess]
Deadline 6
ResultsDirectory results
UPDATE: Simulation started
> --------------------------------
> Input data
> --------------------------------
> Infrastructure:
> VM Id | VM CCU | VM price
> 0     | 5      | 10
> 1     | 10     | 25
> --------------------------------
> Workflow:
```

```
> Level Id | Average level CCU | Task count
> 0        | 20                | 2
> 1        | 10                | 2
> 2        | 20                | 1
> --------------------------------
> Global deadline: 6
> --------------------------------
UPDATE: Iteration started. Level: 0
UPDATE: Global planning started. Level: 0
UPDATE: Global planning failed. Level: 0.
    Error: jCmpl error: No solution found so far
UPDATE: Global planning alternative succeed. Level: 0
UPDATE: Local planning started. Level: 0
UPDATE: Local planning succeed. Level: 0
UPDATE: Executing tasks started. Level: 0
UPDATE: Executing tasks finished. Level: 0
UPDATE: Iteration finished. Level: 0
> --------------------------------
> Global planning results
> Used model: Alternative model (minimize time,
    deadline can not be meet)
> Estimate cost: 185
> Estimate time: 8
> Level Id | #Tasks | Avg CCU | Est time | Est cost | VMs[#Tasks]
> 0        | 2      | 20      | 4        | 90       | 0[1], 1[1]
> 1        | 2      | 10      | 2        | 45       | 0[1], 1[1]
> 2        | 1      | 20      | 2        | 50       | 1[1]
> --------------------------------
> --------------------------------
> Local planning results
> Level: 0
> Estimate time: 4
> Estimate cost: 115
> TaskId  | TaskCCU | VmId | VmCCU | Est time | Est cost
> 0       | 22      | 1    | 10    | 3        | 75
> 1       | 18      | 0    | 5     | 4        | 40
> --------------------------------
> --------------------------------
> Level execution
> Level Id: 0
```

```
> Real cost: 70
> Real time: 2
> Task Id | VM Id | Real time | Real cost
> 0       | 1     | 2         | 50
> 1       | 0     | 2         | 20
> ---------------------------------
> *********************************
> Iteration results
> Iteration Id: 0
> Remaining time before scheduling: 6
> ---------------------------------
> Global planning results for all remaining level
> Estimate time: 8
> Estimate cost: 185
> ---------------------------------
> Local planning results for next level
> Estimate time: 4
> Estimate cost: 115
> ---------------------------------
> Execution results for last level
> Execution time: 2
> Execution cost: 70
> ---------------------------------
> (level execution time - estimated time from global planning) = -2
> (level execution time - estimated time from local planning)  = -2
> (level execution cost - estimated cost from global planning) = -20
> (level execution cost - estimated cost from local planning)  = -45
> ---------------------------------
> Total time elapsed: 2
> Total current cost: 70
> *********************************
UPDATE: Iteration started. Level: 1
UPDATE: Global planning started. Level: 1
UPDATE: Global planning succeed. Level: 1
UPDATE: Local planning started. Level: 1
UPDATE: Local planning succeed. Level: 1
UPDATE: Executing tasks started. Level: 1
UPDATE: Executing tasks finished. Level: 1
UPDATE: Iteration finished. Level: 1
> ---------------------------------
```

```
> Global planning results
> Used model: Main model (minimize total cost under given deadline)
> Estimate cost: 95
> Estimate time: 4
> Level Id | #Tasks | Avg CCU | Est time | Est cost | VMs[#Tasks]
> 1        | 2      | 10      | 2        | 45       | 0[1], 1[1]
> 2        | 1      | 20      | 2        | 50       | 1[1]
> --------------------------------
> --------------------------------
> Local planning results
> Level: 1
> Estimate time: 2
> Estimate cost: 45
> TaskId  | TaskCCU | VmId | VmCCU | Est time | Est cost
> 2       | 10      | 1    | 10    | 1        | 25
> 3       | 10      | 0    | 5     | 2        | 20
> --------------------------------
> --------------------------------
> Level execution
> Level Id: 1
> Real cost: 90
> Real time: 4
> Task Id | VM Id | Real time | Real cost
> 2       | 1     | 2         | 50
> 3       | 0     | 4         | 40
> --------------------------------
> ********************************
> Iteration results
> Iteration Id: 1
> Remaining time before scheduling: 4
> --------------------------------
> Global planning results for all remaining level
> Estimate time: 4
> Estimate cost: 95
> --------------------------------
> Local planning results for next level
> Estimate time: 2
> Estimate cost: 45
> --------------------------------
> Execution results for last level
```

```
> Execution time: 4
> Execution cost: 90
> --------------------------------
> (level execution time - estimated time from global planning) = 2
> (level execution time - estimated time from local planning)  = 2
> (level execution cost - estimated cost from global planning) = 45
> (level execution cost - estimated cost from local planning)  = 45
> --------------------------------
> Total time elapsed: 6
> Total current cost: 160
> ********************************
UPDATE: Iteration started. Level: 2
UPDATE: Global planning started. Level: 2
UPDATE: Global planning failed. Level: 2.
    Error: jCmpl error: No solution found so far
UPDATE: Global planning alternative succeed. Level: 2
UPDATE: Local planning started. Level: 2
UPDATE: Local planning succeed. Level: 2
UPDATE: Executing tasks started. Level: 2
UPDATE: Executing tasks finished. Level: 2
UPDATE: Iteration finished. Level: 2
> --------------------------------
> Global planning results
> Used model: Alternative model (minimize time,
    deadline can not be meet)
> Estimate cost: 50
> Estimate time: 2
> Level Id | #Tasks | Avg CCU | Est time | Est cost | VMs[#Tasks]
> 2        | 1      | 20      | 2        | 50       | 1[1]
> --------------------------------
> --------------------------------
> Local planning results
> Level: 2
> Estimate time: 2
> Estimate cost: 50
> TaskId  | TaskCCU | VmId | VmCCU | Est time | Est cost
> 4       | 20      | 1    | 10    | 2        | 50
> --------------------------------
> --------------------------------
> Level execution
```

```
> Level Id: 2
> Real cost: 50
> Real time: 2
> Task Id | VM Id | Real time | Real cost
> 4       | 1     | 2         | 50
> ---------------------------------
> *********************************
> Iteration results
> Iteration Id: 2
> Remaining time before scheduling: 0
> ---------------------------------
> Global planning results for all remaining level
> Estimate time: 2
> Estimate cost: 50
> ---------------------------------
> Local planning results for next level
> Estimate time: 2
> Estimate cost: 50
> ---------------------------------
> Execution results for last level
> Execution time: 2
> Execution cost: 50
> ---------------------------------
> (level execution time - estimated time from global planning) = 0
> (level execution time - estimated time from local planning)  = 0
> (level execution cost - estimated cost from global planning) = 0
> (level execution cost - estimated cost from local planning)  = 0
> ---------------------------------
> Total time elapsed: 8
> Total current cost: 210
> *********************************
UPDATE: Simulation finished
UPDATE: Started plots generation
UPDATE: Executing R script started.
    Command: [Rscript.exe,
    RScripts\workflowScheduleTimePlot.R,
results\...\schedule_time_illustrative_2_2_1-toLess_D6.csv,
results\...\schedule_time_illustrative_2_2_1-toLess_D6,
6, 8]
UPDATE: Executing R script succeed.
```

```
    Script name: RScripts\workflowScheduleTimePlot.R
UPDATE: Executing R script started.
    Command: [Rscript.exe, RScripts\workflowScheduleCostPlot.R,
    results\...\schedule_cost_illustrative_2_2_1-toLess_D6.csv,
    results\...\schedule_cost_illustrative_2_2_1-toLess_D6,
    252]
UPDATE: Executing R script succeed.
    Script name: RScripts\workflowScheduleCostPlot.R
```

B. Publication

Below there is attached article published in Parallel Processing and Applied Mathematics with presented algorithm.

Dziok, Tomasz and Figiela, Kamil and Malawski, Maciej *Adaptive Multi-level Workflow Scheduling with Uncertain Task Estimates*. Parallel Processing and Applied Mathematics (pages 90-100), Springer International Publishing, ISBN 978-3-319-32151-6, 2016

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.

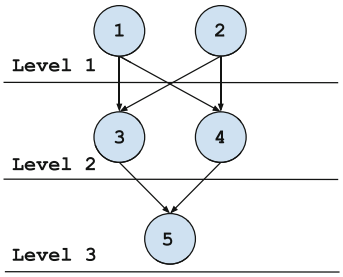


Fig. 1. DAG example

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

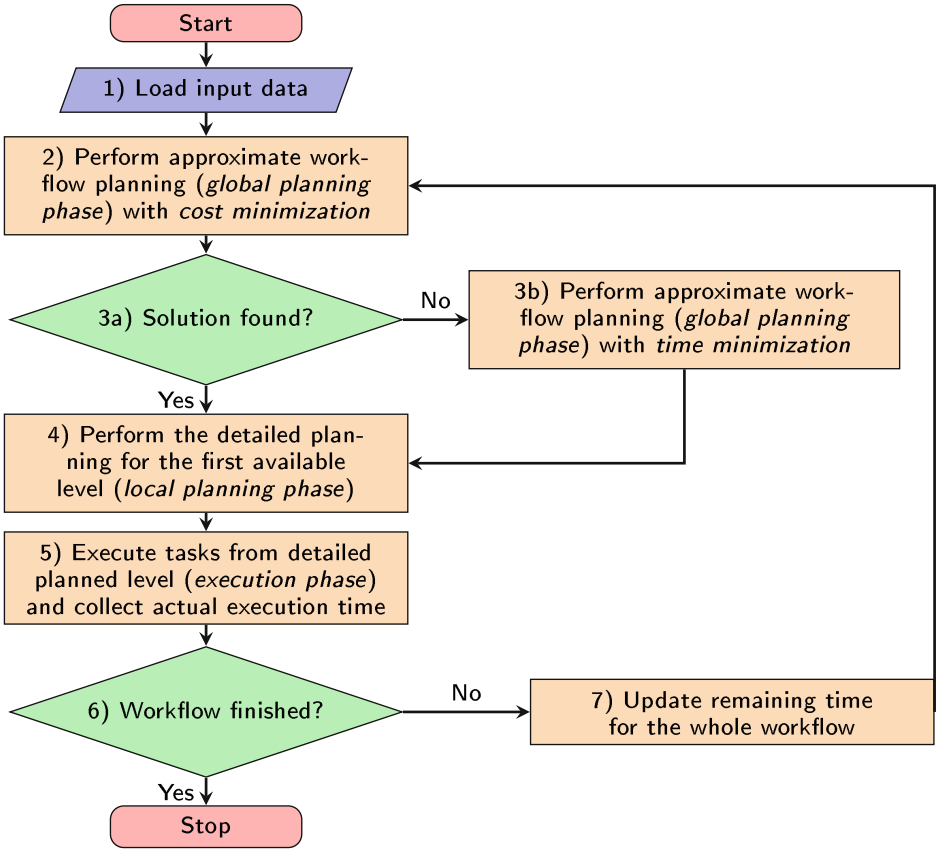


Fig. 2. High level flow of scheduling algorithm.

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

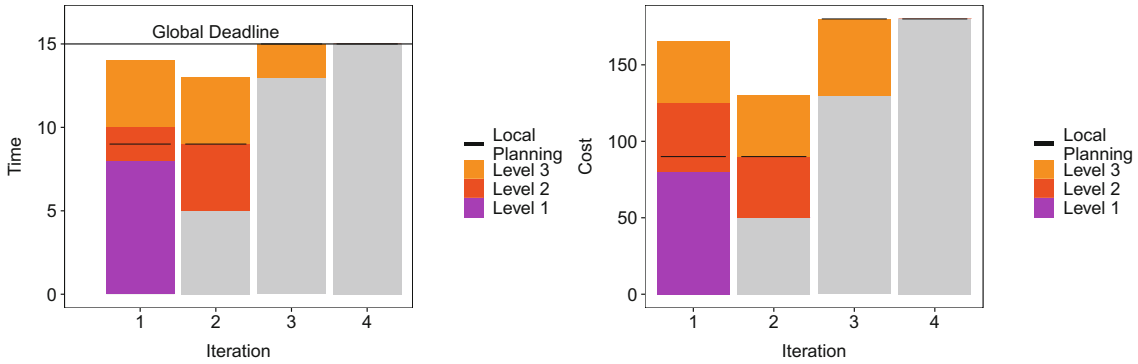


Fig. 3. Execution time and cost of the algorithm, shown level by level. In the first iteration, the global planning phase estimates the completion time of level 1 is 8 (purple bar) and the local planning estimates it to be 9 (solid line). In iteration 2, it turns out that the level L1 finished at time 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.

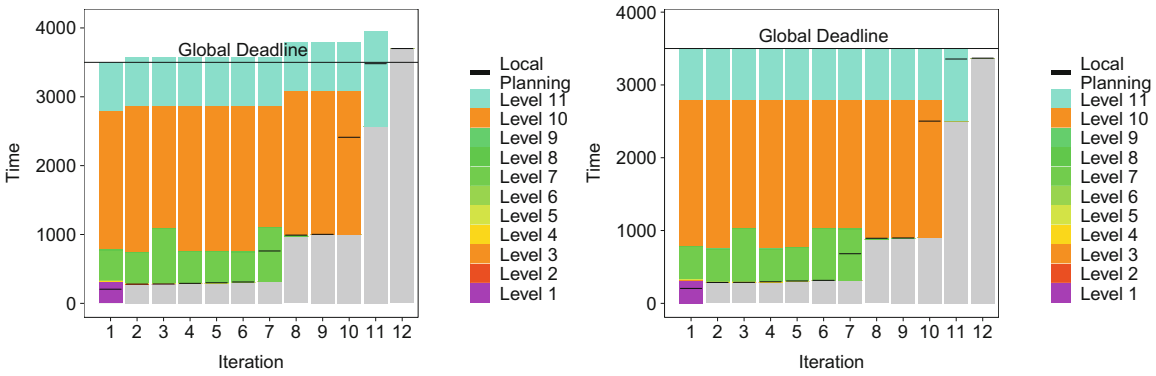
Implementation of Algorithm and Models. Optimization models are implemented in CMPL modeling language [19]. As a solver we use CBC [11]. Input data is loaded from DAG files (workflows) and JSON files (infrastructure). The simulator which executes the tasks and introduces the runtime variations is implemented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

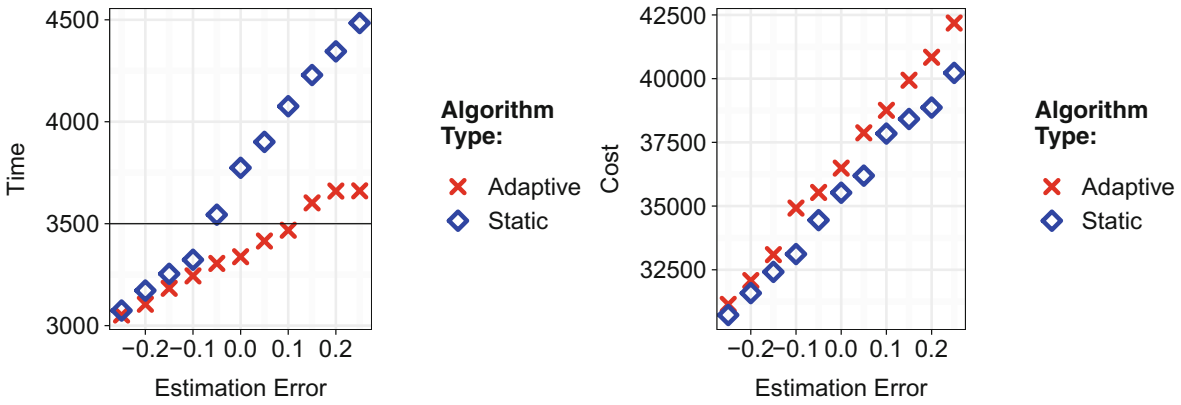
Fig. 4. Execution time plot for Montage 5000 workflow with random errors of estimates.

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.

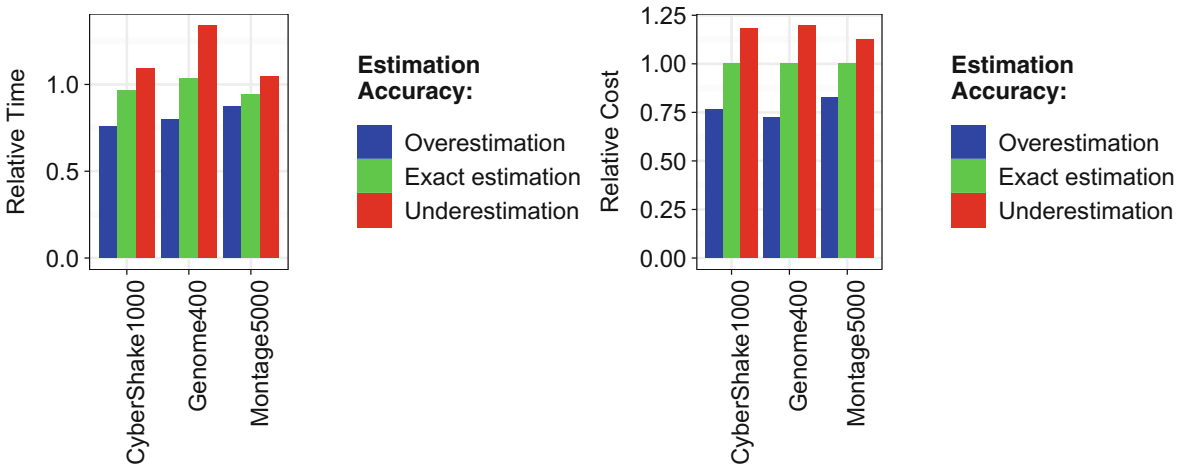
Figure 5 presents how the completion time and total cost depend on the varying estimation error μ. The errors were generated using the normal distribution with the standard deviation of 0.25 and the mean of μ, with μ from -0.25 (overestimation) to 0.25 (underestimation). In plot (a) we observe that our 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