AGH University of Science and Technology
in Kraków

Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science

Master of Science Thesis

Optimization of Resource Allocation on the Cloud

Author:
Kamil Figiela

Supervisor:
dr inż. Maciej Malawski

Kraków, August 2013

Aware of criminal liability for making untrue statements I declare that the following thesis was written personally by myself and that I did not use any sources but the ones mentioned in the dissertation itself.

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji
Katedra Informatyki

Praca Magisterska

Optymalizacja alokacji zasobów w chmurze obliczeniowej

Autor:
Kamil Figiela

Promotor:
dr inż. Maciej Malawski

Kraków, Sierpień 2013

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Abstract

Nowdays, hardly any science can be done without a computing infrastructure, and cloud systems are regarded by the scientific community as a potential source of low-cost computing resources that can be provisioned on-demand according to pay-per-use model. Running scientific applications on the cloud imposes the monetary cost of the computation that should be subject to optimization as the funding is usually limited.

We address the problem of resource allocation on multiple cloud platforms formulated as a mixed integer non-linear programming problem (MINLP). We optimize scheduling of bag of tasks applications and workflows under the deadline constraint. The optimization models implemented in AMPL modeling language allow us to apply leading solvers such as Cbc and CPLEX. We assume multiple IaaS clouds with heterogenous VM instances, with limited number of instances per cloud and hourly billing. Our objective, the total cost, includes computation cost as well as data transfer charges which may have significant contribution to the total cost.

The results illustrate typical problems when making decisions on deployment planning on clouds and how they can be addressed using optimization techniques. We indicate how optimization of resource allocation may be used by end-users to minimize their costs or by resellers for a profit.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Maciej Malawski for the continuous support of my M.Sc. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would also like to thank my parents for their endless love and support. Last, but not least, I would also like to thank Jarek Nabrzyski from University of Notre Dame for giving me opportunity to experience real research work two years ago.

This thesis was realized partially in the framework of the project Virtual Physiological Human: Sharing for Healthcare (VPH-Share) - partially funded by the European Commission under the Information Communication Technologies Programme (contract number 269978).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the general idea of this work. Section 1.1 describes the motivation for optimization of resource allocation on the cloud. Section 1.2 discusses the model of scientific applications. Section 1.3 describes cloud business models and services that are used to deploy applications. Section 1.4 states the goals of the thesis. Section 1.4 enumerates challenges to be overcame.

1.1 Motivation

Planning and scheduling activities have been present in people's life ever since first civilization was built. It usually was related to travel planning or construction planning. Ancient Romans needed to take care on where aqueducts should be build depending on the demand and how much material should be used. Every craftsman needed to optimize how much material will he use to create his product. Later on, this became even more important during the Industrial Revolution.

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

This thesis illustrates typical problems when making decisions on resource allocation for scientific applications on IaaS clouds and how they can be addressed using optimization

techniques. In contrast to already well established computing and storage resources (clusters, grids) for the research community, clouds in the form IaaS platforms (pioneered by Amazon EC2) provide on-demand resource provisioning with a pay-per-use model. These capabilities together with the benefits introduced by virtualization, make clouds attractive to the scientific community [5]. As a result, multiple deployment scenarios differing in costs and performance, coupled together with new provisioning models offered by clouds make the problem of resource allocation and capacity planning for scientific applications a challenge.

1.2 Scientific applications

Scientific computing covers a wide range of fields including biology, chemistry, physics, economics, engineering, finance, geophysics, linguistics, mathematics, and mechanics. Applications used in that sciences are often distributed, so that application may run a numbers of magnitude faster than sequential one. This is crucial in some fields such as weather forecasting or financial modeling to get results as fast as possible.

Scientific applications usually may be put to one of the following groups:

- workflows,
- bag of tasks,
- map-reduce applications,
- sequential batches,
- High Performance Computing applications.

In this thesis we will focus on resource allocation for workflows and bag of tasks applications.

1.2.1 Workflows

Scientific workflow is concerned with the automation of scientific processes in which tasks are structured based on their control and data dependencies [6]. Workflow application is composed by connecting multiple scientific tasks to their dependencies. Workflow structure indicates the temporal relationship between the tasks. In general, a workflow can be represented as a Directed Acyclic Graph (DAG) or a non-DAG. Figure 1.1 shows example *Montage* workflow that is used to create mosaics of the sky.

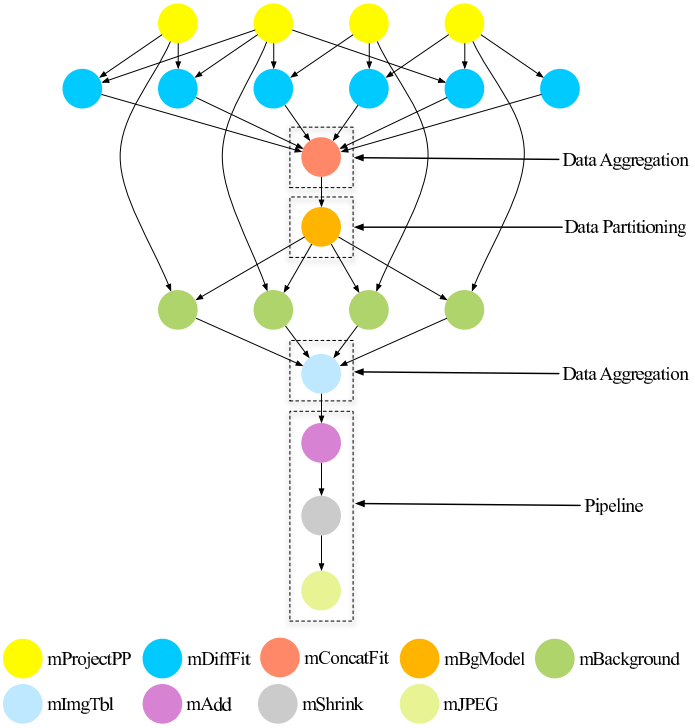


FIGURE 1.1: Montage – example workflow [7]

Workflow is built of four base control structures: *sequence*, *parallelism*, *choice* and *loop* (only for non-DAGs). Sequence represents an ordered series of task with one starting after the previous task has completed. Parallelism represents tasks that are performed concurrently, rather than serially. The choice structure allows to run selected portion of the workflow if certain conditions are met. The iteration structure lets certain block of tasks to be repeated. Consecutive stages of workflow often represent work of scientist – data gathering, preprocessing, processing and results aggregation.

In terms of scientific computing we will be rather interested in data flow in workflow. It is then composed of the following structures (Figure 1.2): *process*, *pipeline*, *data distribution*, *data aggregation* and *data redistribution*.

Scientific workflows are usually run on shared infrastructure as clusters, grids or clouds. It requires to carefully plan and schedule computation to efficiently use given infrastructure. As the workflows are getting bigger and more complex it is nearly impossible to do it manually. Specifically, scheduling workflow applications in a distributed system is an NP-complete problem [8]. Example workflow scheduling algorithms are presented in Chapter 2.

There exists multiple workflow systems that assist scientists to create and deploy their workflow. Popular ones are Pegasus [9] and Taverna [10]. They provide tools to model workflow either in code or via GUI application. Then one is able to generate workflow

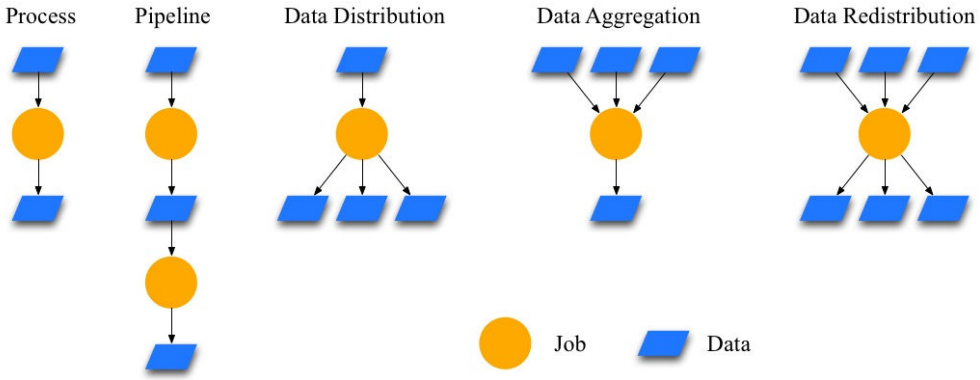


FIGURE 1.2: Data flow structures in Workflows [7]

execution plan on certain (e.g. grid) architecture. Workflow systems differ in supported workflow types (DAG or non-DAG), workflow scheduling policies and algorithms, fault tolerance, and supported infrastructure. The good overview on workflow systems taxonomy is given in [11].

1.2.2 Bag of tasks

Bag of tasks applications represent group of independent tasks that may be processed sequentially or in parallel. There are no dependencies between tasks. A large parameter sweep is good example of such problem. The *map* stage of map-reduce (see Figure 1.3) application can be also considered as a bag of tasks. Additionally, many workflows include stages of a high number parallel tasks. Such examples can be found e.g. in typical scientific workflows executed using Pegasus Workflow Management system, where e.g. CyberShake or LIGO workflows have a parallel stage of nearly homogeneous tasks [7]. Other examples are Wien2K and ASTRO workflows that consist of iteratively executed parallel stages comprising homogeneous tasks [12]. Due to the high number of parallel branches, these stages accumulate the most significant computing time of the whole application, so optimization of the execution of this stage is crucial.

1.3 Introduction to Cloud Computing

Cloud computing became de facto standard of delivering computing resources adopted both by commercial and scientific communities. Nowadays, hardly any science can be performed without computational science and cloud systems are regarded by the scientific community as a potentially attractive source of low-cost computing resources as they can be provisioned on-demand according to pay-per-use model.

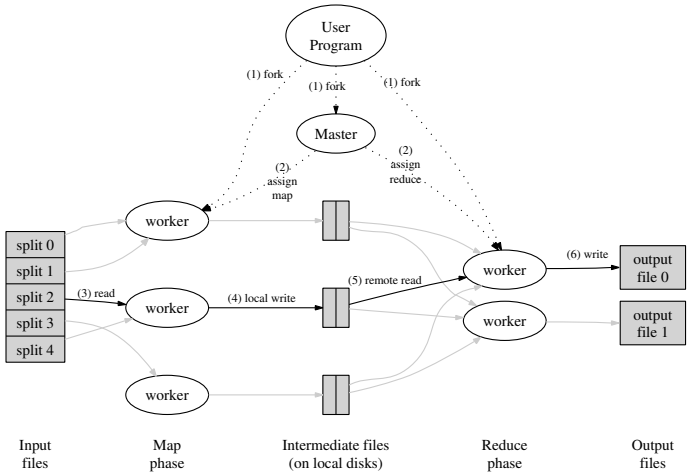


FIGURE 1.3: Map-reduce execution overview[13]

There are multiple definitions of the cloud computing. The term is frequently used for marketing of hosted services or applications running in client-server model. As defined by US National Institute of Standards and Technology (NIST) [14], cloud computing is "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

1.3.1 Service models

We may categorize cloud services in terms of service model that give different level of control and responsibilities to user and service provider:

Software as a Service (SaaS). The capability provided to the user is to use applications deployed by provider running on a cloud infrastructure. The applications are usually available by web-browser based interface or program interface. The underlying cloud infrastructure including network, servers, operating system and application are managed by service provider. Popular SaaS applications include Gmail, Evernote and Salesforce.

Platform as a Service (PaaS). The capability provided to the user is to deploy his own applications created using programming languages, libraries, services and tools provided by the provider. The consumer does not manage underlying cloud infrastructure including network, servers, operating systems, runtime environment, but has control over the deployed application and configuration settings for cloud environment. Example platforms include Heroku, Google App Engine and Nodejitsu.

Infrastructure as a Service (IaaS). The capability provided to the user is to provision processing, storage, networks and other fundamental computing resources where the consumer is able to deploy and run arbitrary software. The consumer does not manage the underlying hardware infrastructure, but has control over operating system, storage, deployed applications and has possibly limited control over networking (i.e. hosts firewall). Example platforms include Amazon EC2 and Rackspace.

1.3.2 Deployment models

Depending on who manages cloud infrastructure we may distinguish the following models:

Private Cloud. The cloud resources are provisioned for exclusive use by a single organization.

Community Cloud. The cloud resources are provisioned for exclusive use by specific community of consumers from organization that have shared concerns. The concept is similar to scientific grid systems.

Public Cloud. The cloud resources are provisioned for open use by the general public.

Hybrid Cloud. The cloud infrastructure is composed of two or more distinct cloud providers that remain unique entities, but are bound together by technology that enables data and application portability. This technique is used for cloud bursting and offloading peak load to public cloud while using private resources when off-peak times.

1.3.3 IaaS compute cloud

Usually IaaS clouds provide three types of resources that are provisioned with a pay-per-use model:

Computing. Provided as virtual machine (VM) instances. Multiple instance types are available that differ with CPU power, RAM memory and additional hardware (i.e. GPU units). VMs are usually billed for instance running time (wall clock time, not CPU time) usually rounded up to full hours.

Storage. Provided as virtualized disk drives for virtual machines (i.e. Amazon's EBS) or as object store available as external service (i.e. Amazon's S3). User is usually

billed for persisting data per GiB[1] per month. Additional charges may also apply (i.e. per IO transaction).

Networking. Provides connectivity between VMs, storages and the Internet. User is billed for the data transferred of the cloud, while incoming data and transfer inside specific cloud usually remains free. Networking is provided in bundle with computing and storage, not as a separate service.

1.4 Problem statement

In this thesis we will focus on optimization of resource allocation of bag of tasks and workflow applications on the hybrid cloud platforms. Planning scientific experiments requires optimization decisions that take into account both execution time and cost, as well as other constraints. Specifically, we address the cost optimization problem of large-scale applications running on multiple heterogeneous clouds, under deadline constraint and for various limits on numbers of instances imposed by cloud providers. The approach to solve this problems is using mathematical modeling with AMPL and mixed integer programming.

1.5 Goals of the thesis

The major goal of this thesis is the theoretical and practical investigation of optimization of resource allocation on the cloud by using integer linear programming tools and methods. The goal will be accomplished by:

- study of existing works on workflow scheduling and resource allocation on the cloud,
- defining application and infrastructure model,
- formulating integer linear problem for bag of tasks applications and workflows,
- implementing optimization model in AMPL,
- evaluating optimization results, performance and stability,
- answering if ILP is suitable method to solve the problem.

[1] Gibibyte = 2^{30} bytes

The goals are addressed in the following chapters. In Chapter 2 we explore existing work on scientific application scheduling. Chapter 3 gives a quick tutorial on mathematical programming. In Chapters 4 and 5 we go through defining infrastructure and application models that will be then implemented and evaluated. In Chapter 6 we answer if ILP is suitable approach for resource allocation on the cloud.

1.6 Summary

In this chapter we discussed the motivation an goals of this work. We given a brief overview on scientific applications focusing on bag of tasks applications and workflows. We also introduced cloud computing. We will dive deeper into the IaaS cloud characteristics in Section 4.2.

Chapter 2

State of the art review

Cloud computing is widely adopted standard for deploying web applications. Cloud users are charged on *pay as you go* basis that enables for cost and service quality optimization by dynamic application scaling. Traditionally, in parallel and distributed systems like clusters and grids, workflow scheduling has been aimed to optimize the makespan or the time of completing all tasks [15]. However, in context of cloud computing, the user needs to take care not only about makespan, but also about the financial cost of deploying application. Therefore, resource allocation on the cloud becomes a multi-objective optimization problem where no single optimal solution exists.

The problem of resource provisioning in IaaS clouds has been recently addressed in [16, 17] and [18]. They typically consider unpredictable dynamic workloads and optimize the objectives such as cost, runtime or utility function by autoscaling the resource pool at runtime. In [16] they address semi-online resource provisioning for processing tasks with dynamic cloud pricing where users bid for the resources (e.g. Amazon EC2 Spot Instances). On the other hand in [17] they consider workload offloading to the cloud from grid resources under deadline or budget constraint. In [18] they evaluate utility-based policy and reactive policies for dynamic resource provisioning. These approaches, however, do not address the problem of data transfer time and cost, which are an important factor when deploying scientific applications.

Automatic cloud scaling and provisioning is often delivered as a cloud service i.e. Amazon Auto Scaling[1]. Policy or rule based services are primarily designed to scale web applications [18] or bag of tasks applications (e.g. [17, 19]). They take input from monitoring systems and perform scaling-up or scaling-down depending on data from monitoring system. This approach is reasonable for applications with dynamic, unpredictable load as

[1] http://aws.amazon.com/autoscaling/

it allows to keep number of instances low, but high enough to provide certain service quality e.g. ensure acceptable request processing time.

The work presented in this thesis is related to heuristic algorithms for workflow scheduling on IaaS clouds, such as the ones described in [20–23]. In [20] the model assumes that infrastructure is provided by only one provider. In contrast, this work presents optimization on hybrid, multi-provider cloud. Infrastructure model considered differs in that we assume multiple heterogeneous clouds with object storage attached to them, instead of individual machines with peer-to-peer data transfers between them. Instead of scheduling each task individually, this approach proposes a global optimization of placement of workflow tasks and data.

Integer programming approach has been applied to the optimization of service selection for activities of QoS aware grid workflows [24]. On the other hand, in model presented in this thesis assumes the IaaS cloud infrastructure, while the objective function takes into account costs and delays of data transfers associated with the tasks.

The cost minimization problem on clouds addressed in [25] uses a different model from ours. We impose a deadline constraint and assume that the number of instances available from providers may be limited. To satisfy these constraints, the planner has to choose resources from multiple providers. Our model also assumes that VM instances are billed per hour of usage.

The model presented in [26] also uses AMPL/CPLEX as solving platform and they define model for deadline-constrained workflow cost optimization. However, their approach does not address the problem of data transfer time and cost. Furthermore, they do not consider that number of instances on the cloud is usually limited.

Pipelined workflows consisting of stages are addressed in [27], where the processing model is a data flow and multiple instances of the same workflow are executed on the same set of cloud resources. The goal of this work is cost optimization instead of meeting the QoS constraints.

The deadline-constrained cost optimization of scientific workloads on heterogeneous IaaS described in [28] addresses multiple providers and data transfers between them, where the application is a bag of tasks.

2.1 Summary

None of the solutions presented in this chapter solves the problem we stated in Chapter 1. They solve other problems such as scheduling on only one cloud platform, or they are

missing important parts of cost estimation (e.g. data transfer cost). So that, the problem of resource allocation on hybrid cloud platforms is still open.

Chapter 3

Mathematical programming using AMPL

In Section 3.1 we present an overview of mathematical programming and we give problem classification in Section 3.2. In Section 3.3 AMPL and other tools for mathematical programming are introduced. Then in Section 3.4 we formulate example models: linear *Whiskas Cat Food Problem* and integer *shift work scheduling*. These problems are not exactly from scientific computing domain, but they present typical modelling challenges.

3.1 Mathematical Programming

Nowdays, the term programming [29] means writing software, but in 1940s this word was used to describe planning and scheduling activities. It appeared then that restrictions in the planning or scheduling problem may be represented mathematically using equalities and inequalities. The solution satysfying all these constraints is considered as acceptable plan or schedule. Mathematical programming enables us to formally define optimization problem: it's variables, objective and constraints.

Defining the problem is not an easy task. If there are too few constraints, the space of possible solutions is too big. On the other hand, too many constraints can rule desirable solutions out. In the worst case there are no solutions at all. The success of programming relies on key insight into the optimized domain and modelling techniques to find a way round the possible difficulty.

In addition to the constraints, one can define the objective — function of the variables that makes it possible to compare solutions and select the best one. It doesn't matter

how many solutions satisfy the constraints — we are interested in the one that minimizes or maximizes the objective.

In development of optimization model it is very important to classify the problem, so we can select the most suitable way of solving it. If constraints and objectives are linear combinations of the variables then the model is called *linear program* and the process of modelling and solving is called *linear programming*. This class of optimization problems is particurarly important becasue a lot of real world optimization problems may be represented in such way. Additionally, there exists a lot of theory and algorithms to solve such problems in fast, deterministic way even if they have thousands of variables. The ideas of linear programming are also important for analyzing and solving problems that are non-linear.

All useful methods of mathematical programming involve using computers. The first computional method of solving optimization problems, the simplex algorithm [30], was introduced by George Dantzig and was subject to several improvements over the decades.

3.2 Problem classification

In spite of the broad applications of linear programming, the linearity assumption is too unrealistic to be applied to many of real problems. If instead smooth non-linear functions of the variables are used in constraints and objectives we call the program as *non-linear program*. Solving such problems is much harder, but not impossible.

There is also another class of problems called *integer programming* that assumes that variables are integer and in general it is much harder than previous. Fortuanetly, computational power of computers is still increasing and there are efficient algorithms to deal with them.

The optimization problems may be categorized in the following groups:

Linear programming (LP) Objective and constraints in this class are linear functions. Problems in this groups are usually solved by using *simplex*, *interior* or *barrier* method.

Quadratic programming (QP) Convex or concave objective and linear constraints. Solved by simplex-type or interior-type method.

Non-linear programming (NLP) Continuous, but not all-linear objective and constraints. May be solved by several methods including gradient, quasi-newton, augmented lagrangian and interior-point. Unless special conditions are met, solution

found is possibly optimal over only some local neighbourhood. If objective is convex (if minimized) or concave (if maximized) and constraints define a convex region it is guaranteed that optimum found is optimal over the entire feasible region.

Mixed-integer programming (MIP) Linear objective and constraints, some or all of variables are integer-valued. Solved by branch-and-bound approach that uses a linear solver to solve subproblems.

Mixed-integer non-linear programming (MINLP) Non-linear objective and constraints, some or all of variables are integer-valued. Solved by branch-and-bound approach that uses a non-linear solver to solve subproblems.

Constraint programming (CP) In that case, no assumptions can be made on the constraints or the objective. This class is the hardest to solve, as no heuristics can be used. Example problem of this class is *boolean satisfiability problem (SAT)*. Often the problem is simplified to find the feasible solution without any optimization.

3.3 AMPL: A Mathematical Programming Language

To successfully solve optimization problem one needs to do a sequence of multiple tasks as follows:

1. Formulate an abstract model: define variables, constraints and objective.
2. Collect the data for a specific problem instance.
3. Generate instance-specific variables, constraints and objective.
4. Solve the problem by running a program called solver that implements algorithm that finds optimal solution.
5. Analyze the results.
6. Refine the model and the data as necessary, and repeat.

Unfortunately, usually people use different form of representing the data than algorithms do. This makes formulation and generation phases complex as modeler would like to express constraints in human-readable language e.g. mathematical notation, and solvers require to provide multiple matrices as input. We need to transform *modeler's form* to *algorithm's form*. Doing it manually is time consuming and erron-prone task.

To automate this task matrix generators were created for specific models. Although they successfully automate matrix generation they are hard to code, debug and maintain.

Modeler needs to be both mathematician and programmer. The other way to solve that problem is to use mathematical modeling language. Several languages (i.e. AMPL [31], Gams [32], PuLP [33], OscaR [34]) were created over the decades.

By using modeling language, modeler may express in comfortable way also designed to serve as input for the computer. Then matrix generation may be fully automated without intermediate state of computer programming, thus mathematical programming becomes cheaper and more reliable. Benefits of formulating in modeling languages become particularly advantageous for models being developed and subject to change. Additionally, modeling language translator may introduce optimizations to generated problem instance.

AMPL is an algebraic mathematical modeling language that resembles traditional mathematical notation to describe variables, objectives and constraints. Code in AMPL will be familiar for anybody that studied basic algebra or calculus, so that he or she doesn't need to be programmer (in present meaning). Algebraic modeling languages allow to express a wide range of optimization problems: linear, nonlinear and integer.

3.3.1 Available solvers

As soon as model is formulated and matrices generated, we may proceed with solving the specific instance of our problem. To do that we will need solver – a program that implements one of solving algorithms. There is wide range of existing solvers available, both open-source (i.e. Cbc [35]) and commercial (i.e. CPLEX [36]) ones that differ with the problem classes they target. Full list of available solvers is published at AMPL website [37].

Usually solvers provide multiple options that let us tune them for the specific application. One may enable or disable certain features of the solver, i.e. for *Bonmin* [38] solver we may choose branching algorithm or configure it to use heuristics.

3.4 Example – Whiskas Cat Food Problem

To get some practice with modeling, we will describe a typical linear programming problem on the example of *Whiskas Cat Food problem*. This is typical planning problem that may be found in many textbooks [30]. The company producing the food wants to produce it as cheaply as possible while ensuring they meet the stated nutritional analysis requirements stated on the cans.

Ingredient	**Price per gram**
Chicken	$ 0.013
Beef	$ 0.008
Mutton	$ 0.010
Rice	$ 0.002
Wheat	$ 0.005
Gel	$ 0.001

TABLE 3.1: Cat food ingredient's pricing.

	Protein	**Fat**	**Fibre**	**Salt**
Chicken	0.100	0.080	0.001	0.002
Beef	0.200	0.100	0.005	0.005
Mutton	0.150	0.110	0.003	0.007
Rice	0.000	0.010	0.100	0.002
Wheat bran	0.040	0.010	0.150	0.008
Gel	–	–	–	–

TABLE 3.2: Ingredient contribution to the final product in grams per gram of ingredient.

Minimum % Crude Protein	8.0
Minimum % Crude Fat	6.0
Maximum % Crude Fibre	2.0
Maximum % Salt	0.4

TABLE 3.3: Cat food nutritional analysis.

Main ingredients of the cat food used are chicken, beef, mutton, rice wheat and gel. The prices for the ingredients are presented in Table 3.1, while ingredient contribution to the total weight of protein, fat, fibre and salt in the final product are give in Table 3.2 and nutritional requirements are presented in Table 3.3. Given that data we may proceed with model formulation.

3.4.1 Problem formulation

In this particular problem data defines the following data sets:

- $I = \{\text{chicken}, \text{beef}, \text{mutton}, \text{rice}, \text{wheat}, \text{gel}\}$ – defines possible ingredients,
- $C = \{\text{protein}, \text{fat}, \text{fibre}, \text{salt}\}$ – defines components of nutrition.

We have also some numbers that describe members of sets;

- p_i – price of given ingredient i in $ per gram

- c_i, c – contribution of ingredient i to component of nutrition c in grams per gram of ingredient.

Identify the decision variables First of all we need to identify decision variables. For the Whiskas Cat Food Problem the decisions are the amounts of each ingredient we put in the can. Formally we could write this as:

$$x_i = \text{ amount (g) of ingredient } i \text{ in a can of cat food} \tag{3.1}$$

Formulate the Objective Function The objective for this problem is to minimize the total cost of ingredients per fan of cat food. We know the cost per gram of each ingredient and the amount is to be found.

$$\min \sum_{i \in I} p_i x_i \tag{3.2}$$

Formulate the constraints The constraints for the Whiskas Cat Food are:

1. The sum of the amounts must make up the whole can (i.e. 100 g).
2. The stated nutritional analysis requirements are met.

First of the constraints can is:

$$\sum_{i \in I} x_i = 100 \tag{3.3}$$

The latter can be written as follows:

Component of nutrition	**Lower bound**	**Upper bound**
Protein	8.0	–
Fat	6.0	–
Fibre	2.0	–
Salt	–	0.4

TABLE 3.4: Bounds for contribution of component of nutrition in percent.

$$\sum_{i \in I} c_{i,protein} x_i \geq 8.0 \tag{3.4}$$

$$\sum_{i \in I} c_{i,fat} x_i \geq 6.0 \tag{3.5}$$

$$\sum_{i \in I} c_{i,fibre} x_i \geq 2.0 \tag{3.6}$$

$$\sum_{i \in I} c_{i,salt} x_i \leq 0.4 \tag{3.7}$$

$$\tag{3.8}$$

or in more general way, we may define lower and upper bounds for each component of nutrition as L_c and U_c, the values are presented in Table 3.4. Then the constraint will be written as

$$\underset{c \in C}{\forall} L_c \leq \sum_{i \in I} c_{i,c} x_i \leq U_c \tag{3.9}$$

$$\tag{3.10}$$

We have formulated general problem using mathematical notation. Now we will proceed with model formulation using AMPL.

3.4.2 Problem formulation using AMPL

AMPL enables us to separate model definition and instance specific data. Usually we create three files: model, data and calling script. In the model file we define abstract optimization model: sets and parameters, objective and constraints. Then in data file we populate the sets and parameters with the numbers for the particular instance of the problem. Both model and data files are loaded from calling script that may do some pre or post processing.

Model formulation First of all, we should define the sets for the ingredients and components of nutrition. We create file called `whiskas.mod` that will contain abstract model of optimization: sets, parameters, constraints and objective.

```
set INGREDIENTS;
set COMPONENTS;
```

Using sets we can define the decision variables

```
var Amount {INGREDIENTS} >= 0;
```

and parameters for the costs, components of nutrition contribution, restrictions and can size.

```
param Cost {INGREDIENTS} >= 0;
param Contribution {INGREDIENTS, COMPONENTS} >= 0;

param Lower {COMPONENTS} default -Infinity;
param Upper {c in COMPONENTS} >= Lower[c], default Infinity;

param CanSize >= 0;
```

and the objective function

```
minimize TotalCost : sum {i in INGREDIENTS} Cost[i] * Amount[i];
```

Note that, for the bounds that are not set, we assume $\pm$*Infinity*. Additionally, we define that *Cost* and *Contribution* parameters should be non negative – AMPL provides also validation of parameter data.

Finally, we may define constraints

```
subject to MeetRequirements {c in COMPONENTS}:
  Lower[c] <= sum {i in INGREDIENTS} Contribution[i, c] * Amount[i] <= Upper[c];

subject to FullCan:
  sum {i in INGREDIENTS} Amount[i] = CanSize;
```

Providing data Now we can provide the model with the data. To do this we create the file `whiskas.dat`.

First of all we need to provide sets we defined in previous paragraph.

```
set INGREDIENTS := CHICKEN BEEF MUTTON RICE WHEAT GEL;
set COMPONENTS  := PROTEIN FAT FIBRE SALT;
```

Then we populate parameters with the data.

```
param     Cost :=
  CHICKEN 0.013
  BEEF    0.008
  MUTTON  0.010
  RICE    0.002
  WHEAT   0.005
  GEL     0.001
;

param     Lower :=
  PROTEIN 8.0
  FAT     6.0
;

param     Upper :=
  FIBRE   2.0
  SALT    0.4
;

param CanSize := 100;

param Contribution :
          PROTEIN   FAT FIBRE  SALT :=
  CHICKEN   0.100 0.080 0.001 0.002
  BEEF      0.200 0.100 0.005 0.005
  MUTTON    0.150 0.110 0.003 0.007
  RICE      0.000 0.010 0.100 0.002
  WHEAT     0.040 0.010 0.150 0.008
  GEL       0.0   0.0   0.0   0.0
;
```

Running AMPL Now we are ready to solve the model we formulated with AMPL. To do that, we create file called `whiskas.run`.

First of all we should reset AMPL environment in case that specific AMPL instance was solving other model before. We can do it with command `reset`. Then we load model using command `model` that takes model file name as an argument. Next, we load data file using command `data`. We also need to tell AMPL which solver we would like to be used. In our case we will use CPLEX (`option solver cplex`). Finally, we call solver and present results. The full file will look like as follows.

```
reset;

model whiskas.mod;

data whiskas.dat;

option solver cplex;
solve;
```

```
display Amount;
```

Now we may run run AMPL and see if the model is working and if so, what is the optimal solution.

```
% ampl whiskas.run
CPLEX 12.4.0.1: optimal solution; objective 0.52
2 dual simplex iterations (0 in phase I)
Amount [*] :=
   BEEF  60
CHICKEN   0
    GEL  40
 MUTTON   0
   RICE   0
  WHEAT   0
;
```

In that case, it appears that it is cheapest to use beef and fill the rest of the can with gel.

3.5 Example – Shift Scheduling

In this section, we will formulate AMPL model for shift scheduling problem [31]. The factory is working in three shift work schedule, operating six days a week from Monday to Saturday. On Saturday there's no third shift as it would overlap with Sunday. There's a certain number of workers required for each shift: more workers are required during first shift and less during the others. Due to work time regulations not all shift combinations of 5 day working week are possible – in fact that there are only 126 shift schedules possible. The crew wages depend on the chosen schedule. We also want to limit number of different schedules used, so workers may work in teams during planned week. The problem is to minimize the cost of running factory while meeting all regulatory constraints. This is the example of *integer linear programming (ILP)* problem, as only the integer solutions are valid.

In this solution, we won't schedule individual persons, but groups of them. We will use similar approach in models presented in Chapter 4 and Chapter 5.

3.5.1 Problem formulation

This particular problem defines the following sets:

- S – set of shifts,

- W – set of acceptable schedules.

We have also some numbers that describe work planning;

- p_w – pay rate at schedule w,
- r_s – number of staff required at shift s,
- n_{min} – minimal number of workers on any schedule.

Identify the decision variables We need to decide if given schedule will be used and if so, how many workers should follow it. Formally we could write this as:

$$u_w = \text{binary, 1 iif schedule } w \text{ should be used, otherwise 0;} \tag{3.11}$$

$$n_w = \text{integer, how many workers should use schedule } w. \tag{3.12}$$

Formulate the Objective Function The objective for this problem is to minimize the total cost of running factory.

$$\min \sum_{w \in W} p_w n_w \tag{3.13}$$

Formulate the constraints The constraints are:

1. The stated minimal number of workers on each shift is met.
2. Each schedule, if used at all, is used by minimal number of workers.

First of the constraints is:

$$\underset{s \in S}{\forall} \sum_{w \in W : s \in W_w} n_w \geq r_s \tag{3.14}$$

The second can be written as follows:

$$\underset{w \in W}{\forall} n_{min} u_w \leq n_w \tag{3.15}$$

$$\underset{w \in W}{\forall} n_w \leq \left(\max_{s \in W_w} r_s \right) u_w \tag{3.16}$$

We have formulated general problem using mathematical notation. Now we will proceed with model problem formulation using AMPL.

3.5.2 Problem formulation using AMPL

After we formulated the abstract model we implement it in AMPL by following the same procedure as in previous section.

Model formulation First of all, we should define the sets and parameters in model file.

```
set SHIFTS;
param Nsched;
set SCHEDS = 1..Nsched;
set SHIFT_LIST {SCHEDS} within SHIFTS;

param rate {SCHEDS} >= 0;
param required {SHIFTS} >= 0;
param least_assign >= 0;
```

Using sets we can define the decision variables

```
var Work {SCHEDS} >= 0 integer;
var Use {SCHEDS} >= 0 binary;
```

and the objective function

```
minimize Total_Cost: sum {j in SCHEDS} rate[j] * Work[j];
```

Finally, we may define constraints

```
subject to Shift_Needs {i in SHIFTS}:
  sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];
subject to Least_Use1 {j in SCHEDS}:
  least_assign * Use[j] <= Work[j];
subject to Least_Use2 {j in SCHEDS}:
  Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

Providing data Now we can provide the model with the data. We provide the data we defined in previous paragraph.

```
set SHIFTS := Mon1 Tue1 Wed1 Thu1 Fri1 Sat1
              Mon2 Tue2 Wed2 Thu2 Fri2 Sat2
              Mon3 Tue3 Wed3 Thu3 Fri3 ;

param rate  default 1 ;

param required :=  Mon1 100  Mon2 78  Mon3 52
                   Tue1 100  Tue2 78  Tue3 52
                   Wed1 100  Wed2 78  Wed3 52
                   Thu1 100  Thu2 78  Thu3 52
                   Fri1 100  Fri2 78  Fri3 52
                   Sat1 100  Sat2 78 ;

param least_assign  := 5;
param Nsched        := 126 ;

set SHIFT_LIST[  1] := Mon1 Tue1 Wed1 Thu1 Fri1 ;
set SHIFT_LIST[  2] := Mon1 Tue1 Wed1 Thu1 Fri2 ;
........
set SHIFT_LIST[126] := Tue2 Wed2 Thu2 Fri2 Sat2 ;
```

Running AMPL

```
reset;
option omit_zero_rows 1;
model whiskas.mod;

data whiskas.dat;

option solver cplex;
solve;

display Work;
display Total_Cost;
```

Now we are ready to run the model and get the results.

```
% ampl sched.run
CPLEX 12.4.0.1: optimal integer solution; objective 266
136 MIP simplex iterations
23 branch-and-bound nodes
Work [*] :=
  3   7
  6  28
 16   8
 18   8
 20  12
 29   7
 37  21
 61   5
 66   5
 78  24
```

```
 82  28
 91  12
100   8
102   6
112  28
118  24
122  30
126   5
;

Total_Cost = 266
```

Given that data we may assign schedules to the specific workers.

3.6 Summary

This chapter presented basics of mathematical optimization. We discussed briefly history of optimization, classification of optimization problems and introduced tools for mathematical optimization using computers. We also shown the example optimization problems and how they can be solved using AMPL.

Chapter 4

Bag of tasks optimization

In this chapter we discuss optimization of scientific applications that may be represented as bag of uniform tasks. The chapter is organized as follows: after defining application model in Section 4.1 and infrastructure model in Section 4.2, we formulate the problem using AMPL by specifying the variables, parameters, constraints and optimization goals in 4.3. Section 4.4 presents the results obtained by applying the model to the scenarios involving multiple public and private clouds, overlapping computation and data transfers, and identifying special cases. Section 4.5 provides a sensitivity analysis of the model and show how such analysis can be useful for potential users or computing service resellers. Section 4.6 shows estimation on how the model behaves if the task sizes are not uniform and change dynamically.

4.1 Application model

The goal of this optimization model is to minimize the cost of processing a given number of tasks on a hybrid cloud platform, as discussed in Section. 4.2. We assume that tasks are independent from each other, but they have identical computational cost and require a constant amount of data transfer.

We assume that for each task a certain amount of input data needs to be downloaded, and after it finishes, the output results need to be stored. In the case of data-intensive tasks, the transfers may contribute a significant amount of total task run time.

The assumption of homogeneous tasks can be justified by the reason that there are many examples of scientific applications (e.g. scientific workflows or large parameter sweeps) that include a stage of a high number of parallel nearly identical tasks. Such examples can be found e.g. in typical scientific workflows executed using Pegasus Workflow

Management system, where e.g. CyberShake or LIGO workflows have a parallel stage of nearly homogeneous tasks [7]. Other examples are Wien2K and ASTRO workflows that consist of iteratively executed parallel stages comprising homogeneous tasks [12]. Due to the high number of parallel branches, these stages accumulate the most significant computing time of the whole application, so optimization of the execution of this stage is crucial. Moreover, if the tasks are not ideally homogeneous, it is possible to approximate them using a uniform set of tasks with the mean computational cost and data sizes of the application tasks. Of course, in real execution the actual task performance may vary, so the solution obtained using our optimization method becomes only approximate of the best allocation, and the actual cost may be higher and deadline may be exceeded. In order to estimate the quality of this approximation, we evaluate the impact of dynamic task runtime and and non-uniform tasks in section 4.6.

4.2 Infrastructure model

Three types of cloud services are required to run scientific application on the cloud: virtual machines, storage and networking. Amazon S3 and Rackspace Cloud Files are good examples of storage providers, while Amazon EC2, Rackspace, GoGrid and ElasticHosts represent computational services. In addition, the optimization model proposed in this thesis includes a private cloud running on own hardware. Each cloud provider offers multiple types of virtual machine instances with different performance and price.

For each provider the number of running virtual machines may be limited. This is mainly the case for private clouds that have a limited capacity, but also the public clouds often impose limits on the number of virtual machines. E.g. Amazon EC2 allows maximum of 20 instances and requires to request a special permission to increase that limit [39].

Most of cloud providers charge their users for each running virtual machine on an hourly basis. Some providers charge in 5-minute (i.e. CloudSigma) or 1-minute cycles (i.e. Google Compute Engine), but it is not widespread practice yet. Additionally, users are charged for remote data transfer while local transfer inside provider's cloud is usually free. These two aspects of pricing policies may have a significant impact on the cost of completing a scientific task.

Cloud services are characterized by their pricing and performance. Instance types are described by price per hour, relative performance and data transfer cost as presented in Table 4.1. To assess the relative performance of clouds it is possible to run application-specific benchmarks on all of them, or to use publicly available cloud benchmarking

Instance type	Price per hour	Instance performance in CCU
Amazon Web Services (AWS) [US East]		
m2.4xlarge	$2.400	27.25
m2.2xlarge	$1.200	14.89
linux.c1.xlarge	$0.680	8.78
m2.xlarge	$0.500	7.05
m1.xlarge	$0.680	5.15
m1.large	$0.340	4.08
c1.medium	$0.170	3.43
m1.small	$0.085	0.92
Rackspace Cloud [Dallas]		
rs-16gb	$0.960	4.95
rs-2gb	$0.120	4.94
rs-1gb	$0.060	4.93
rs-4gb	$0.240	4.90
GoGrid [CA, US]		
gg-8gb	$1.520	23.2
gg-4gb	$0.760	9.28
gg-2gb	$0.380	4.87
gg-1gb	$0.190	4.42
ElasticHosts [UK]		
eh-8gb-20gh	$0.654	9.98
eh-4gb-8gh	$0.326	5.54
eh-2gb-4gh	$0.164	4.75
eh-1gb-2gh	$0.082	4.30
Hypothetical instance of private cloud		
private	$0.000	1.00

TABLE 4.1: Example pricing and performance of cloud instances.

services, such as CloudHarmony [40]. CloudHarmony defines performance of cloud instances in the units named CloudHarmony Compute Units (CCU) as similar to Amazon EC2 Compute Unit (ECU), which are approximately equivalent to CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Storage platforms include fees for data transfer.

4.3 Problem formulation

To perform optimization of the total cost, Mixed Integer Non-Linear Problem (MINLP) is formulated and implemented in A Mathematical Programming Language (AMPL) [31]. As we shown in Chapter 3, AMPL requires to specify input data sets and variables to define the search space, as well as constraints and objective function to be optimized.

4.3.1 Input data

The formulation requires the following input sets, which represent the cloud infrastructure model:

- $S = \{s3, cloudfiles\}$ – defines available cloud storage sites,
- $P = \{amazon, rackspace, \dots\}$ – defines possible computing cloud providers,
- $I = \{m1.small, \dots, gg.1gb, \dots\}$ – defines instance types,
- $PI_p \subset I$ – instances that belong to provider P_p,
- $LS_s \subset P$ – compute cloud providers that are local to storage platform S_s.

Each instance type I_i is described by the following parameters:

- p_i^I – fee in \$ for running instance I_i for one hour,
- ccu_i – performance of instance in CloudHarmony Compute Units (CCU),
- p_i^{Iout} and p_i^{Iin} – price for non-local data transfer to and from the instance, in \$ per MiB.

Storage sites are characterized by:

- p_s^{Sout} and p_s^{Sin} characterize price in \$ per MiB for non local data transfer.

Additionally, we need to provide data transfer rates in MiB per second between storage and instances by defining function $r_{i,s} > 0$.

We assume that the computation time of a task is known and constant, this also applies to input and output data size. We also assume that tasks are atomic (non divisible). Computation is characterized by the following parameters:

- A^{tot} – number of tasks,
- t^x – execution time in hours of one task on 1 CCU machine,
- d^{in} and d^{out} – data size for input and output of one task in MiB,
- p^R – price per request for queuing service,
- t^D – total time for completing all tasks in hours (deadline).

4.3.2 Auxiliary parameters

The formulation of the problem requires a set of precomputed parameters which are derived from the main input parameters of the model. The relevant parameters include:

$$t_{i,s}^{net} = \frac{d^{in} + d^{out}}{r_{i,s} \cdot 3600} \tag{4.1}$$

$$t_{i,s}^{u} = \frac{t^{x}}{ccu_i} + t_{i,s}^{net} \tag{4.2}$$

$$c_{i,s}^{t} = (d^{out} \cdot (p_i^{Iout} + p_s^{Sin}) + d^{in} \cdot (p_s^{Sout} + p_i^{Iin})) \tag{4.3}$$

$$a_{i,s}^{d} = \lfloor \frac{t^{D}}{t_{i,s}^{u}} \rfloor \tag{4.4}$$

$$t_{i,s}^{q} = \lceil t_{i,s}^{u} \rceil \tag{4.5}$$

$$a_{i,s}^{q} = \lfloor \frac{t_{i,s}^{q}}{t_{i,s}^{u}} \rfloor \tag{4.6}$$

$$t_{i,s}^{d} = \lceil \lfloor \frac{t^{D}}{t_{i,s}^{u}} \rfloor \cdot t_{i,s}^{u} \rceil \tag{4.7}$$

- $t_{i,s}^{net}$ – *transfer time*: time for data transfer between I_i and S_s,
- $t_{i,s}^{u}$ – *unit time*: time for processing a task on instance I_i using storage S_s that includes computing and data transfer time (in hours),
- $c_{i,s}^{t}$ – cost of data transfer between instance I_i and storage S_s,
- $a_{i,s}^{d}$ – number of tasks that can be done on one instance I_i when using storage S_s running for t^D hours,
- $t_{i,s}^{q}$ – *time quantum*: minimal increase of instance running time that is sufficient to increase the number of processed tasks, rounded up to full hour,
- $a_{i,s}^{q}$ – number of tasks that can be done in $t_{i,s}^{q}$ hours,
- $t_{i,s}^{d}$ – *instance deadline*: number of hours that can be effectively used for computation.

4.3.3 Variables

Variables that will be optimized and define the solution space are listed below:

- N_i – number of instances of type I_i to be deployed,
- A_i – number of tasks to be processed on instances of type I_i,

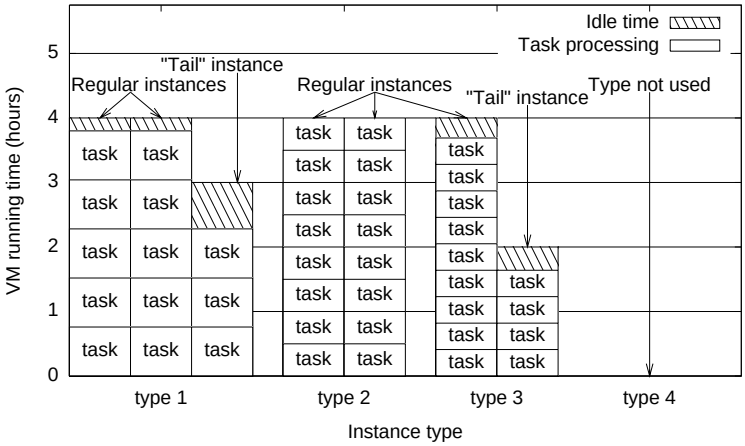


FIGURE 4.1: Task scheduling policy

- D_s – 1 iif S_s is used, otherwise 0; only one storage may be used,
- R_i – number of remainder (tail) hours for instance I_i,
- H_i – 1 iif $R_i \neq 0$, otherwise 0.

Fig. 4.1 illustrates how the tasks are assigned to multiple running instances. The tasks are atomic and there is no checkpointing or preemption. Even though all the tasks have the same computational cost, their total processing time depends on performance of the VM type and data transfer rate. Moreover, the users are charged for the runtime of each VM rounded up to full hours. In our model the tasks are assigned to the instances in the following way. First, in the process of optimization A_i tasks are assigned to the instance I_i. This gives N_i instances of type I_i running for $t^d_{i,s}$ hours. Remaining tasks are assigned to an additional instance running for R_i hours. We will refer to them as tail hours.

In order to enforce provider's instance limit, H_i is introduced which indicates if I_i has any tail hours. E.g. in Fig. 4.1 instances of type 1 have 3 tail hours, and instances of type 2 have no tail hours.

4.3.4 Formulation of constraints and objectives

Cost of running a single task which includes the cost of VM instance time required for data transfer and task computing time, together with data transfer costs and request cost, can be described as:

$$
\begin{aligned}
\left(t^{net} + t^{u}\right) \cdot p^{I} + \\
d^{in} \cdot \left(p^{Sout} + p^{Iin}\right) + \\
d^{out} \cdot \left(p^{Iout} + p^{Sin}\right) + \\
p^{R}
\end{aligned}
$$

The objective function represents the total cost of running multiple tasks of the application on the cloud infrastructure and it is defined as:

$$\underset{\text{total cost}}{\text{minimize}} \sum_{i \in I}((\sum_{s \in S} D_s \cdot t^d_{i,s} \cdot N_i + R_i) \cdot p^I_i + A_i \cdot (p^R + \sum_{s \in S} D_s \cdot c^t_{i,s})) \tag{4.8}$$

subject to the constraints:

$$\underset{i \in I}{\forall} A_i \in \mathbb{Z} \wedge 0 \leq A_i \leq A^{tot} \tag{4.9}$$

$$\underset{s \in S}{\forall} D_s \in \{0, 1\} \tag{4.10}$$

$$\underset{i \in I}{\forall} N_i \in \mathbb{Z} \wedge 0 \leq N_i \leq n^{Imax}_i \tag{4.11}$$

$$\underset{i \in I}{\forall} R_i \in \mathbb{Z} \wedge 0 \leq R_i \leq t^D - 1 \tag{4.12}$$

$$\underset{i \in I}{\forall} H_i \in \{0, 1\} \tag{4.13}$$

$$\underset{i \in I}{\forall} A_i \geq \sum_{s \in S} (N_i \cdot a^d_{i,s}) \cdot D_s \tag{4.14}$$

$$\underset{i \in I}{\forall} A_i \leq \sum_{s \in S} (N_i \cdot a^d_{i,s} + \max(a^d_{i,s} - 1, 0)) \cdot D_s \tag{4.15}$$

$$\underset{i \in I}{\forall} R_i \geq \sum_{s \in S} (A_i - N_i \cdot a^d_{i,s}) \cdot t^u_{i,s} \cdot D_s \tag{4.16}$$

$$\underset{i \in I}{\forall} R_i \leq \sum_{s \in S} (A_i - N_i \cdot a^d_{i,s} + a^q_{i,s}) \cdot t^u_{i,s} \cdot D_s \tag{4.17}$$

$$\sum_{i \in I} A_i = A^{tot} \tag{4.18}$$

$$\sum_{s \in S} D_s = 1 \tag{4.19}$$

$$\underset{i \in I}{\forall} H_i \leq R_i \leq \max(t^D - 1, 0) \cdot H_i \tag{4.20}$$

$$\underset{p \in P}{\forall} \sum_{i \in PI_p} (H_i + N_i) \leq n^{Pmax}_p \tag{4.21}$$

Interpretation of the constraints is the following:

- (4.9) to (4.13) define weak constraints for a solution, i.e. they are to ensure that the required variables have the appropriate integer or binary values,
- (4.14) and (4.15) ensure that number of instances is adequate to number of assigned tasks; for chosen storage D_s

  $$N_i \cdot a^d_{i,s} \leq A_i \leq N_i \cdot a^d_{i,s} + \max\left(a^d_{i,s} - 1, 0\right)$$

  where the lower bound is given by full allocation of N_i machines and the upper bound includes a fully allocated tail machine,
- (4.16) and (4.17) ensure that number of tail hours is adequate to number of remaining tasks, implement $R_i = \left\lceil \left(A_i - N_i \cdot a^d_{i,s}\right) \cdot t^u_{i_s} \right\rceil$,
- (4.18) ensures that all tasks are processed,
- (4.19) ensures that only one storage site is selected,
- (4.20) ensures that R_i has proper value, implements $H_i = \begin{cases} 1, & R_i > 0 \\ 0, & R_i = 0 \end{cases}$,
- (4.21) enforces providers' instance limits.

4.3.5 Solver choice

Defining the problem in AMPL enables to choose among a wide range of solvers that can be used as backend. The problem itself is MINLP, but can be reduced to Integer Linear Programming (ILP) problem. The nonlinear part of problem comes from storage choice, so by fixing storage provider and running optimization procedure for each storage separately the optimal solution is found.

Initially Bonmin [41] solver was used, but after the model was fully implemented and subject to more tests, it appeared that CBC [35] solver performs better with default options. This results from the fact that Bonmin is a solver designed to solve MINLP problems and uses various heuristics, while CBC uses a branch and bound algorithm tuned for ILP. As the problem is linear and convex, CBC finds global optimum. The model was optimized so that it should give acceptable results in ∼0.10 seconds [1].

[1] As measured on quad-core Intel i5 machine.

4.4 Experiments and results

To evaluate the model, we first run the optimization process for two scenarios with a private cloud and (1) infinite public clouds (Section 4.4.1) and (2) finite public clouds (Section 4.4.2). We also evaluated the effect of possible overlapping of computations and data transfers in Section 4.4.3. When testing the model under various input parameters, we identified interesting special cases described in Section 4.4.4. Finally, we discuss a sensitivity analysis, its implications and potential usage in Section 4.5.

Data intensive tasks that require relatively large amount of data for short computing time (we assumed 512 MiB of input and 512 MiB of output) and compute intensive tasks that require relatively small amount of data (we assumed 0.25 MiB of input and 0.25 MiB of output). Each case consists of 20,000 tasks, each of them requires 0.1 hour of computing time on a VM with performance of 1 CCU. Two scenarios with infinite and finite public clouds were considered, where as the costs and performance of public clouds we used the data from CloudHarmony benchmarks. This dataset gives the data of 4 compute cloud providers (Amazon EC2, Rackspace, ElasticHosts and GoGrid) and 2 storage providers (Amazon S3 and Rackspace Cloud Files), giving the instance prices between $0.06 and $2.40 per hour and performance between 0.92 and 27.4 CCU (see Table 4.1). The pricing model assumes that the private cloud instances have the performance of 1 CCU and $0 cost. For each scenario the deadline parameter ranges between 5 and 100 hours. Since only two storage providers (S3 and Cloud Files) were considered, the solver is run separately for these two parameters.

4.4.1 Private + infinite public clouds

This scenario assumes that the private cloud can run a maximum of 10 instances, while the public clouds have unlimited capacity. The results for tasks that require small amount of data are shown in Fig. 4.2. As the deadline is extended, the total cost linearly drops as expected. As many tasks as possible are run on the private cloud, and for all the remaining tasks the instance type with best price to performance ratio is selected.

This situation changes as data size grows (Fig. 4.3) – for data intensive tasks the total cost is nearly constant as all the work is done on a public cloud. This results from the fact that data transfer cost to and from the private cloud is higher than the cost of running instances on public cloud. In our case it turns out that the lowest cost can be achieved when using Cloud Files storage from Rackspace, since in our dataset the best instance type in terms of price to performance was available at Rackspace.

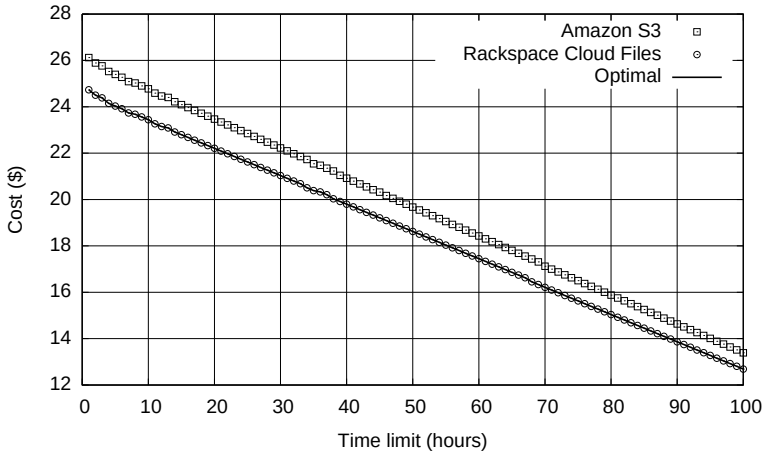


FIGURE 4.2: Small data processed on infinite public cloud

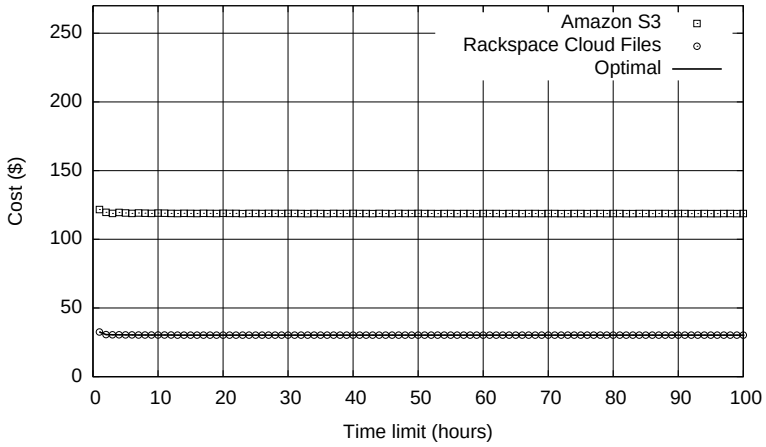


FIGURE 4.3: Large data processed on infinite public cloud

4.4.2 Private + finite public clouds

If assumption that public clouds are limited is made, situation is not so straightforward (See Fig. 4.4 and 4.5). For relatively long deadlines, a single cloud platform for both VMs and storage can be used, which means that the data transfer is free. As the deadline shortens we first observe a linear cost increase. At the point when the selected cloud platform reaches its VM limit, additional clouds need to be used, so we need to begin paying for the data transfer. Therefore the cost begins to increase more rapidly.

This effect is very significant in the case of data intensive tasks (Fig. 4.5) as the cost growth may become very steep. For example, in our tests the task processing cost in 28 hours was \$157.39, in 30 hours it was \$131.14 and in 34 hours it was only \$30.26. For longer deadlines there was no further decrease. We can also observe that for longer deadlines the Cloud Files storage provider is less expensive for the same reason as it

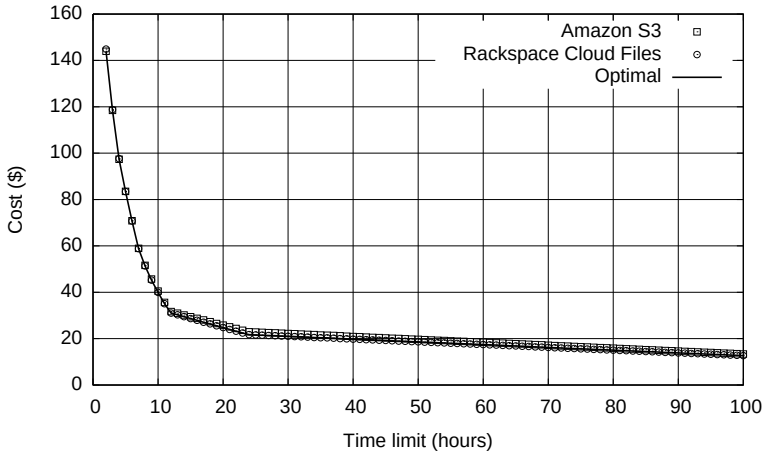


FIGURE 4.4: Small data processed on finite public cloud

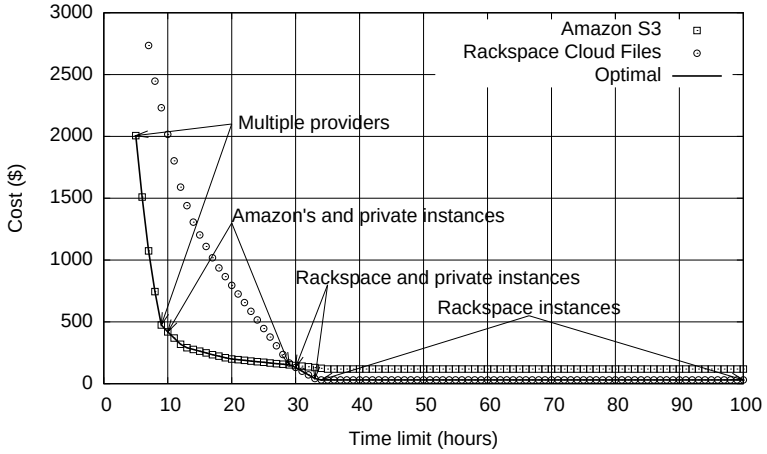


FIGURE 4.5: Large data processed on finite public cloud

was in Fig. 4.3. Shorter deadlines, however, require to run more powerful instances from other clouds (Amazon EC2), thus it becomes more economical to use its local S3 storage.

4.4.3 Overlapping computation and data transfers

In the model we assumed that computation and data transfers are not overlapping. To achieve parallelism of these two processes, the model needs to be modified in the following way. The total task computation time is maximum of task execution and data transfer time. Additionally, input for the first task and output of the last task must be transferred sequentially. Equations 4.2, 4.4 and 4.7 are updated for this case as follows:

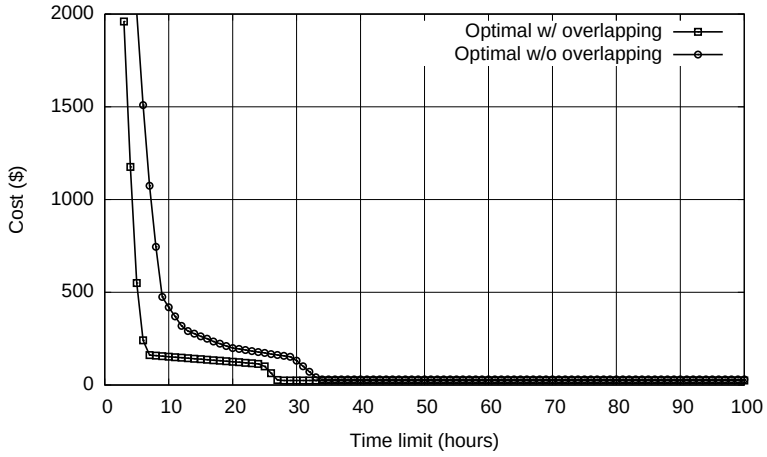


FIGURE 4.6: Large data processed on finite public cloud with overlapping computation and data transfer

$$t^u_{i,s} = \max(\frac{t^x}{ccu_i}, t^{net}_{i,s}) \tag{4.22}$$

$$a^d_{i,s} = \lfloor \frac{(t^D - t^{net}_{i,s})}{t^u_{i,s}} \rfloor \tag{4.23}$$

$$t^d_{i,s} = \lceil \lfloor \frac{(t^D - t^{net}_{i,s})}{t^u_{i,s}} \rfloor \cdot t^u_{i,s} \rceil \tag{4.24}$$

Fig. 4.6 shows results for the same experiment as in Section 4.4.2 and Fig. 4.5, but with computing and transfers overlapping. Overlapping reduces total cost as time required for task processing is significantly lower. This is especially true for shorter deadlines when multiple clouds are used, as transfer rates are lower between different clouds comparing to one provider infrastructure.

4.4.4 Identifying special cases

Running a test case with very large tasks which cannot be completed within one hour on largest available instance revealed an interesting model behavior. Results are shown on Fig. 4.7. Local minima may occur for certain deadlines thus cost does not increase monotonically with decrease of deadline. This is a consequence of our task scheduling policy, as explained in Section 4.3.3. In the case of large tasks, the schedule for deadline of 9 hours as seen in Fig. 4.8a costs less than for deadline of 10 hours as in Fig. 4.8b. In the first case the total number of VM-hours of the *gg-8mb* instance is equal to 56, but in the case of deadline of 10 hours the total number is 58, which results in higher cost. This is the result of the policy, which tries to keep VMs running time as close to the deadline

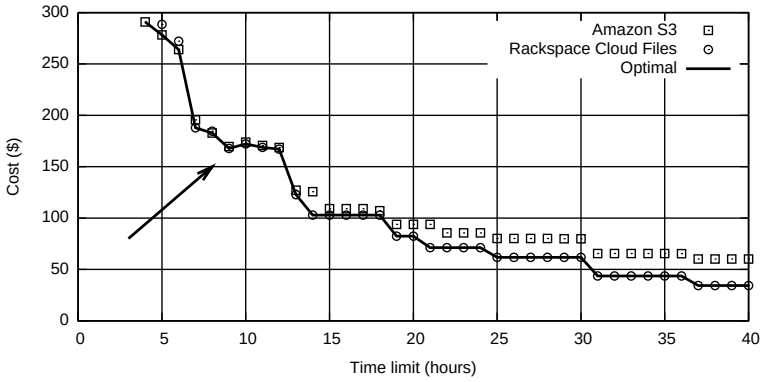


FIGURE 4.7: Special case with local minimum for deadline 9

as possible. Such policy is based on general observation, that for a given budget it is usually more economical to start less VMs for longer time than to start more VMs for shorter time. However, there are rare cases when such policy may lead to non-optimal solutions. It should be noted, that the solution of the model returned by the solver in this case is optimal, but it is the model itself that does not allow to find the minimum cost.

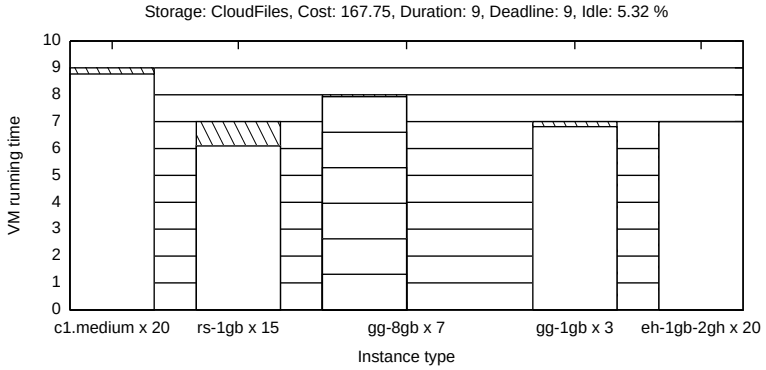
Moreover, for longer deadlines the cost is a step function – e.g. the cost for deadline of 18 hours is the same as for 14 hours. These two observations suggest that the model could be modified in such a way that the deadline is also a variable with upper bound constraint. Similar result can be achieved in a simple way by solving the current model for multiple deadlines in the neighborhood of the desired deadline and by selecting the best solution.

4.5 Sensitivity analysis

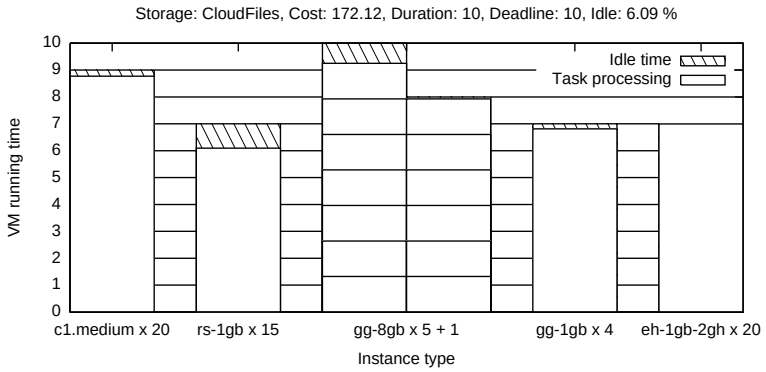
In order to better assess how the model behaves in response to the changing constraints and for varying input parameters, I performed a sensitivity analysis by sampling a large parameter space. Fig. 4.9 shows the solutions obtained for deadlines ranging from 2 to 40 hours and for input and output data sizes ranging from 0 to 256 MiB. As it can be seen in the plot, for all data sizes the cost increases monotonically with the decrease of the deadline, which confirms that no anomalies are observed. The same data can be also observed as animated plot available as on-line supplement[2].

Fig. 4.10 presents these results from a perspective where each curve shows how the cost depends on data size for varying deadlines.

[2]See also `http://youtu.be/FWjgMwLdZW4`



(A) Deadline = 9 hours



(B) Deadline = 10 hours

FIGURE 4.8: Task schedule for the identified special case

One of the questions that our model can help answer is how the changes in cost are sensitive to the changes of deadline. Fig. 4.11 shows the cost function as well as corresponding elasticity, which is computed as: $Ef(x) = \frac{x}{f(x)} f'(x)$, where in this case f is cost and x is deadline. Elasticity gives the information on what is the percentage change of cost in response to the percentage change of deadline. The function has negative values, since the cost decreases with the increase of deadline. It is interesting to see in which ranges the elasticity has larger absolute value, which corresponds to more steep cost function. Here we can see that the elasticity grows for short deadlines and close to the deadline of 25 hours, which is the point where the solution cannot use only the VM instances from most cost-effective clouds and requires more expensive ones to meet the deadline.

Identifying such ranges with high absolute value of elasticity is important for potential users of the cloud system, including researchers (end users) or resellers (brokers). The end user can for example observe that changing the deadline from 40 to 30 hours or even from 20 to 10 hours will not incur much additional cost. However, changing the deadline from 30 to 20 hours is very costly, so it should be avoided. On the other hand, the situation looks different from the perspective of a reseller, who buys cloud resources from providers and offers them to end-users in order to make profit. The reseller can for

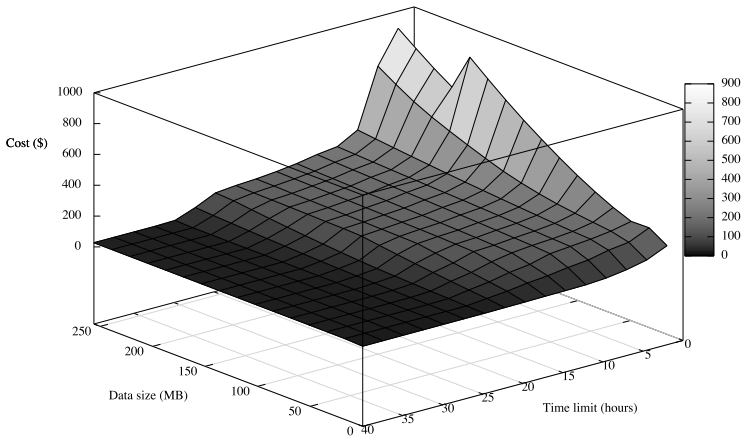


FIGURE 4.9: Optimal cost for a wide range of deadline constraints and data sizes.

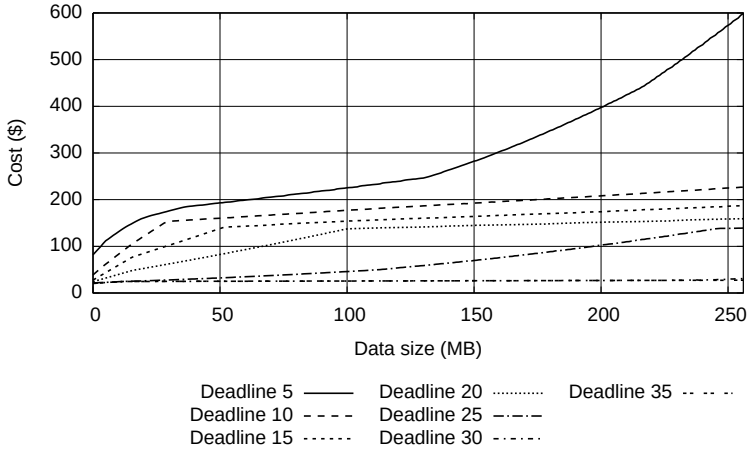
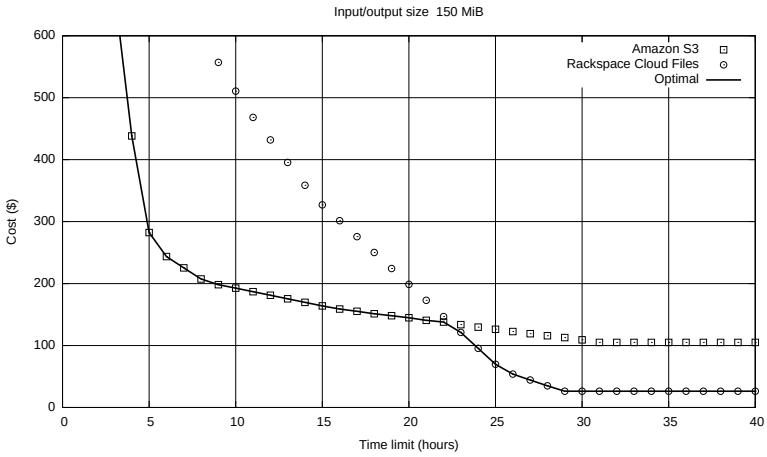


FIGURE 4.10: Cost as a function of data size for different deadlines.

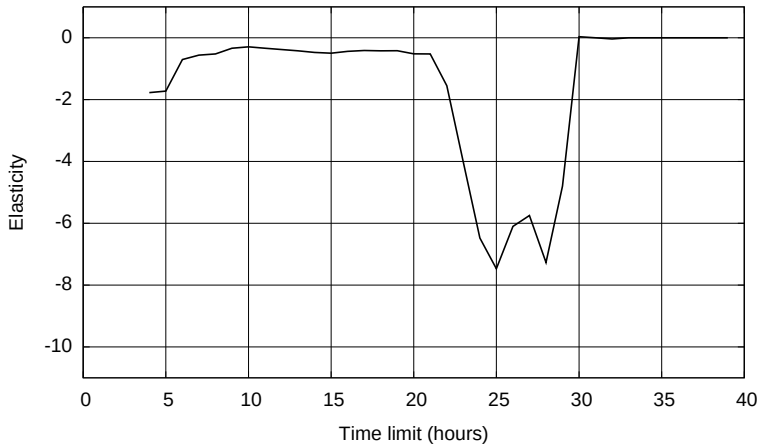
example offer to compute the given set of tasks in 20 hours for $150 with minimal profit, but it can also offer to complete the same set of tasks in 30 hours for $50. Such price can seem attractive to the end user who pays 1/3 of the price for increasing the deadline by 50%, but it is in fact very beneficial for the reseller, whose profit reaches 100%. Such cost analysis is an important aspect of planning large computing experiments on cloud infrastructures.

4.6 Impact of dynamic environment

As stated in section 4.1, we assume that execution time, transfer rate and data access time for all the tasks are constant. However, in real environments the actual runtime of tasks will vary. The goal of the following simulation experiment was to estimate the



(A) Cost as a function of deadline



(B) Elasticity of cost versus deadline

FIGURE 4.11: Sensitivity of costs in response to changes in deadline

impact of this runtime variation on the quality of results obtained using our optimization model.

For all the task assignment solutions presented in Fig. 4.9 we add runtime variations to the task runtimes in the following way. For each task we generate a random error in the range from $-v$ to v using uniform distribution, where v is the runtime variation range. We tested values of $v = 10\%, 20\%, 30\%, 40\%, 50\%$. Such modified task runtimes are then used to calculate the actual runtime and cost of VM. Due to variation of task runtimes, it is possible that some computations may not finish before the deadline (time overrun) and that the cost of additional VM-hours may need to be paid (cost overrun).

We assume that task size variations include also variations of data transfer time. We don't take into account variations of data transfer costs, since the transfer costs for each task depend linearly on data size, so the aggregated impact of positive and negative variations cancels to nearly 0.

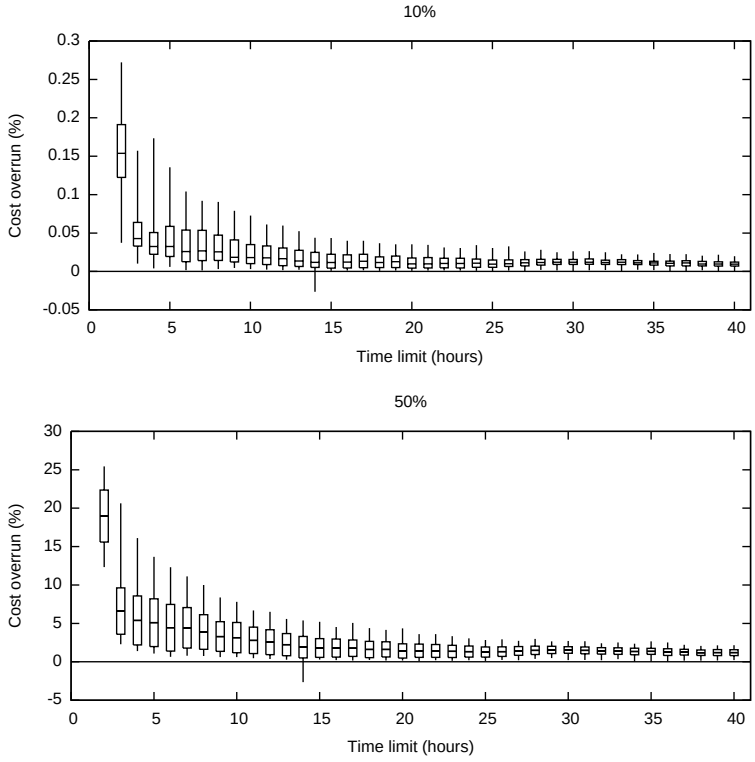


FIGURE 4.12: Impact of runtime variation on cost overrun as a function of deadline. Results obtained for random variation of task runtime in the range from −10% to +10% (top) and from −50% to +50% (bottom).

Fig. 4.12 shows the cost overrun with 10% and 50% of runtime variation range. The box-plots for each deadline represent averaged results for the data sizes from 0 to 256 MiB as in Fig. 4.9, with box boundaries at quartiles and whiskers at maximum and minimum values. We observe that the highest overruns are for the shortest deadlines, which results from the fact that when each VM instance runs only for a few hours, then the additional hour will incur relatively high additional cost. On the other hand, for longer deadlines the cost of additional VM-hour becomes less significant. We also observe that the aggregated impact of positive and negative variations in task execution time may cancel to nearly 0 and in some cases the cost overrun may be negative.

In a similar way Fig. 4.13 shows the deadline overrun in the presence of runtime variations. We can observe that the deadline overrun is much smaller than cost overrun. This means that the actual finish time of all tasks is not significantly affected by the runtime variation, due to the cancellation effect. We observe that even for high variations in the range from −50% to 50% the actual runtime rarely exceeds the deadline by more than 10%. This overrun can be thus easily compensated by solving the optimization problem with a respectively shorter deadline giving a safety margin in the case of expected high variations of actual task runtimes. Similar estimation is also possible in the case when

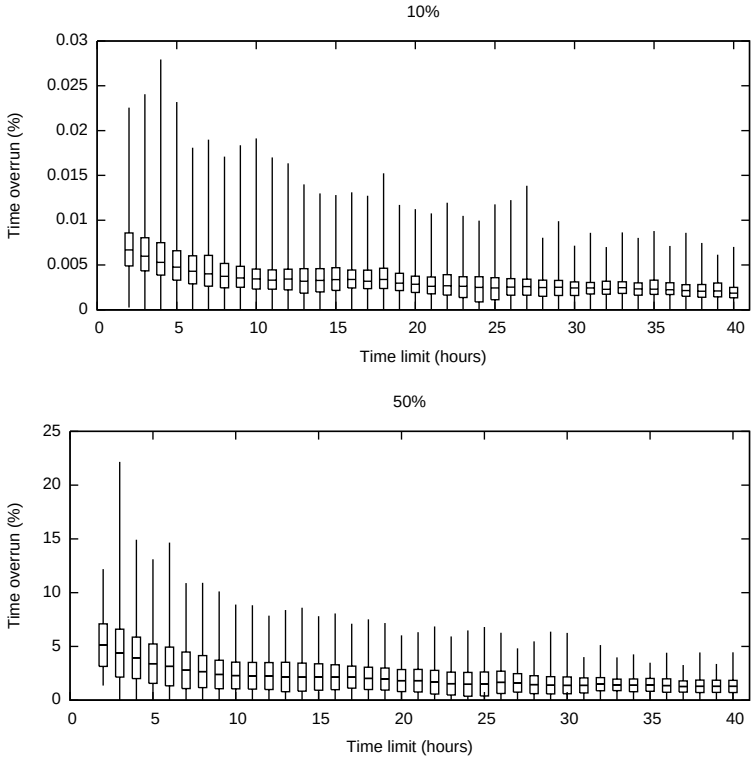


FIGURE 4.13: Impact of runtime variation on deadline overrun as a function of deadline. Results obtained for random variation of task runtime in the range from −10% to 10% (top) and from −50% to +50% (bottom).

the application consists of many tasks of similar size which distribution in known in advance.

4.7 Summary

In this chapter we discussed bag of tasks applications model and cloud infrastructure characteristics. This led us to formulating AMPL model for optimizing cost of running bag of tasks applications. The model was later evaluated in terms of optimization runtime and sensitivity. The results show that the total cost grows slowly for long deadlines, since it is possible to use free resources from a private cloud. However, for short deadlines it is necessary to use the instances from public clouds, starting from the ones with best price/performance ratio. The shorter the deadlines, the more costly instance types have to be added, thus the cost grows more rapidly. Moreover, our results can be also useful for multi-objective optimization. In such a case, it would be possible to run the optimization algorithm in a certain neighborhood of the desired deadline and select the best solution using a specified cost/time trade-off. Alternatively, multiple solutions as in Fig. 4.5, 4.9 or 4.10 may be corollary acceptable solution.

Chapter 5

Workflow optimization

In this chapter we discuss optimization of scientific applications that may be represented as workflows. In Section 5.1 we discuss the details of scientific workflows. Then in Section 5.2 we formulate optimization problem using AMPL by defining the data, variables, constraints and the objective. Finally, in we evaluate in Section 5.3 by scheduling example workflows from Workflow Generator Gallery.

5.1 Application model

As described in Section 1.2.1, a scientific workflow describes the dependencies between tasks and in most cases may be represented as directed acyclic graph (DAG), where nodes represent tasks and edges represent task dependencies.

There are different approaches to optimize workflow scheduling (see Chapter 2). Mathematical programming allows to describe the model mathematically and use a set of available optimization solvers. On the other hand, an attempt to apply this method to the general problem of scheduling large-scale workflows on heterogeneous cloud resources would be impractical due to the problem complexity, therefore simplified models need to be analyzed. Optimization models can be simplified in several ways. We may simplify application model, infrastructure model or make certain assumptions on the schedule.

In this model we assume that workflow is represented as a DAG. We also assume that a workflow is divided into several levels (layers) that can be executed sequentially and tasks within one level do not depend on each other (see Fig. 5.1). Each layer represents a bag of tasks that can be partitioned in several groups (e.g. application A, application B, etc.) that share computational cost and input/output size. We assume that only one task group is executed on a specific cloud instance (VM). This forbids instance sharing

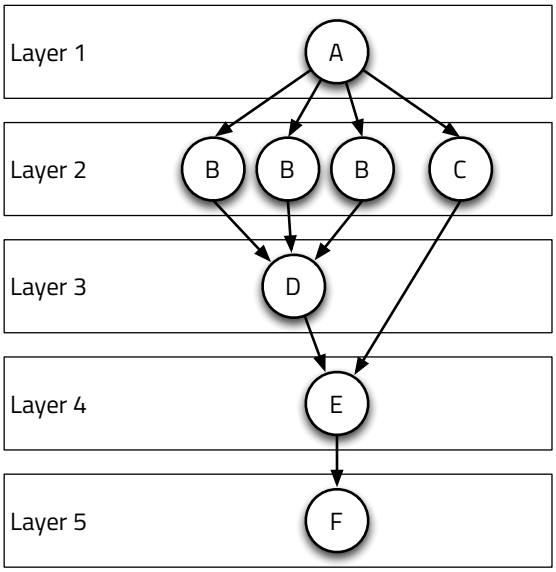


FIGURE 5.1: Example application structure

between multiple layers, which means that each application needs its own specific VM template with a preconfigured environment.

In this model we will use the same infrastructure model as presented in Section 4.2.

5.2 Problem formulation using AMPL

Similarly to the model presented in Chapter 4 we will formulate and implement the model in AMPL.

5.2.1 Input data

The formulation requires the following input sets, which represent the infrastructure model, in a similar way as we approached the problem in Chapter 4:

- $S = \{s3, cloudfiles\}$ – defines available cloud storage sites,
- $P = \{amazon, rackspace, \dots\}$ – defines possible computing cloud providers,
- $I = \{m1.small, \dots, gg.1gb, \dots\}$ – defines instance types,
- $PI_p \subset I$ – instances that belong to provider P_p,
- $LS_s \subset P$ – compute cloud providers that are local to storage platform S_s.

Each instance type I_i is described by the following parameters:

- p_i^I – fee in \$ for running instance I_i for one hour,
- ccu_i – performance of instance in CloudHarmony Compute Units (CCU),
- p_i^{Iout} and p_i^{Iin} – price for non-local data transfer to and from the instance, in \$ per MiB (1 MiB = $1024 * 1024$ Bytes)

Storage sites are characterized by:

- p_s^{Sout} and p_s^{Sin} characterize price in \$ per MiB for non local data transfer.

Additionally we need to provide data transfer rates in MiB per second between storage and instances by defining function $r_{i,s} > 0$.

Our application model is different from the one in Chapter 4 because it groups tasks into layers:

- L – set of layers,
- G – set of tasks groups,
- G_l – set of tasks groups belonging to layer l,
- A_t^{tot} – number of tasks in group t,
- t_t^x – execution time in hours of a single task of group t on 1 CCU machine,
- d_t^{in} and d_t^{out} – data size for input and output of one task t in MiB,
- p^R – price per request for queuing service, such as Amazon SQS, required to execute a single task,
- t^D – total time for completing workflow (deadline).

5.2.2 Auxiliary parameters

A set of precomputed parameters which are derived from the main input parameters of the model includes:

$$t_{i,s}^{net} = \frac{d^{in} + d^{out}}{r_{i,s} \cdot 3600} \tag{5.1}$$

$$t_{i,s}^{u} = \frac{t^x}{ccu_i} + t_{i,s}^{net} \tag{5.2}$$

$$\begin{aligned} c_{i,s}^{T} = &(d^{out} \cdot (p_i^{Iout} + p_s^{Sin}) \\ &+ d^{in} \cdot (p_s^{Sout} + p_i^{Iin})) \end{aligned} \tag{5.3}$$

$$\tag{5.4}$$

- $t_{i,s}^{net}$ – *transfer time*: time for data transfer between I_i and S_s,
- $t_{i,s}^{u}$ – *unit time*: time for processing a task on instance I_i using storage S_s that includes computing and data transfer time (in hours),
- $c_{i,s}^{T}$ – cost of data transfer between instance I_i and storage S_s,
- I_i^{idx} – set of possible instance I_i indexes (from 0 to $n_i^{Imax} - 1$).

5.2.3 Variables

Variables that will be optimized and define the solution space are:

- $A_{t,i,x}$ – binary, 1 iif (if and only if) instance I_i with index x is launched to process task group G_t, otherwise 0;
- $H_{t,i,x}$ – integer, for how many hours is instance launched;
- $T_{t,i,x}$ – integer, how many tasks of G_t are processed on that instance,
- D_l^t – actual computation time for L_l,
- D_l – integer, maximal number of hours that instances are allowed to run in L_l.

5.2.4 Formulation of objectives

Cost of running one task including instance and transfer cost is:

$$\left(t^{net} + t^{u}\right) \cdot p^{I} + d^{in} \cdot \left(p^{Sout} + p^{Iin}\right) + d^{out} \cdot \left(p^{Iout} + p^{Sin}\right) + p^{R} \tag{5.5}$$

while the objective function represents the total cost of running multiple tasks of the application on the cloud infrastructure is defined as:

$$\underset{\text{total cost}}{\text{minimize}} \sum_{t \in G, i \in I, x \in I_i^{idx}} \left((p_i^I * H_{t,i,x} + p^R + c_{i,s}^T) * T_{t,i,x} \right) \tag{5.6}$$

subject to the constraints:

$$\sum_{l \in L} D_l \leq t^D \tag{5.7}$$

$$\underset{l \in L}{\forall} D_l^t \leq D_l \leq D_l^t + 1 \tag{5.8}$$

$$\underset{t \in G, i \in I, x \in I_i^{idx}}{\forall} A_{t,i,x} \leq H_{t,i,x} \leq A_{t,i,x} \cdot t^D \tag{5.9}$$

$$\underset{t \in G, i \in I, x \in I_i^{idx}}{\forall} A_{t,i,x} \leq T_{t,i,x} A_{t,i,x} \cdot A_t^{tot} \tag{5.10}$$

$$\underset{t \in G, i \in I, x \in I_i^{idx}}{\forall} H_{t,i,x} \leq D_l \tag{5.11}$$

$$\underset{l \in L, t \in G_l, i \in I, x \in I_i^{idx}}{\forall} T_{t,i,x} \cdot t_{t,i,s}^u \leq D_l^t \tag{5.12}$$

$$\underset{t \in G, i \in I, x \in I_i^{idx}}{\forall} T_{t,i,x} \cdot t_{t,i,s}^u \leq H_{t,i,x} T_{t,i,x} \cdot t_{t,i,s}^u + 1 \tag{5.13}$$

$$\tag{5.14}$$

$$\underset{t \in G}{\forall} \sum_{i \in I, x \in I_i^{idx}} T_{t,i,x} = A_t^{tot} \tag{5.15}$$

$$\underset{t \in G, i \in I, x \in \{1..(n_i^{Imax} - 1)\}}{\forall} H_{t,i,x} \leq H_{t,i,x-1} \tag{5.16}$$

$$\underset{t \in G, i \in I, x \in \{1..(n_i^{Imax} - 1)\}}{\forall} A_{t,i,x} \leq A_{t,i,x-1} \tag{5.17}$$

$$\underset{t \in G, i \in I, x \in \{1..(n_i^{Imax} - 1)\}}{\forall} T_{t,i,x} \leq T_{t,i,x-1} \tag{5.18}$$

$$\underset{l \in L, p \in P}{\forall} \sum_{i \in PI_p, t \in G_l, x \in I_i^{idx}} A_{t,i,x} \leq n_p^{Pmax} \tag{5.19}$$

Interpretation of the constraints is the following:

- (5.7) ensures that workflow finishes in given deadline,
- (5.8) fix that $D = \lceil D^t \rceil$,
- (5.9) ensure that H may be allocated only iif A is 1,
- (5.10) ensure that T may be allocated only iif A is 1,
- (5.11) enforces layer deadline on instances runtime,
- (5.12) enforces layer finishes work in D^t,
- (5.13) adjust H respectively to T in order to make sure that all the instances run for enough time to process all tasks allocated to them we require,
- (5.15) ensures that all tasks are processed,

- (5.16 to 5.18) reject symmetric solutions,
- (5.19) enforces instance limits per cloud.

To keep this model in MIP class we had to take different approach than in previous model, and schedule each virtual machine instance separately. The drawback of this approach is that we need to increase the number of decision variables. The search space is also divided by storage provider. Additionally, the deadline becomes a variable with upper bound as it may happen that shorter deadline may actually give a cheaper solution (see Fig. 5.3 and its discussion).

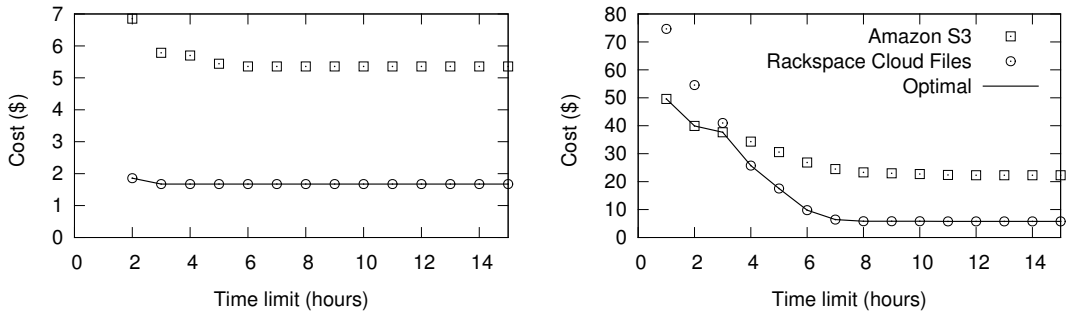
5.3 Evaluation

To evaluate our model on realistic data, we use CloudHarmony [40] benchmarks to parameterize the infrastructure model, and we use the Workflow Generator Gallery workflows [7] as test applications.

In the infrastructure model we assumed that we have 4 public cloud providers (Amazon EC2, RackSpace, GoGrid and ElasticHosts) and a private cloud with 0 cost. The infrastructure has two object storage services, S3 that is local to EC2 and CloudFiles that is local to RackSpace, so data transfers between local compute and storage are free.

We tested our model with all application types from the gallery: Montage, CyberShake, Epigenomics, LIGO and SIPHT for all available workflow sizes (from 50 to 1000 tasks per workflows, up to 5000 tasks in the case of SIPHT workflow). We varied the deadline from 1 to 30 hours with 1-hour increment. We solve the problem for two cases, depending on whether the data is stored on Amazon S3 or on RackSpace CloudFiles.

Fig. 5.2 shows the example results obtained for the Epigenomics application and workflows of two sizes (400 and 500 tasks). For longer deadlines the private cloud instances



(A) 400 tasks, 1 GiB data size (B) 500 tasks, 4 GiB data size

FIGURE 5.2: Result of the optimization procedure for the Epigenomics application.

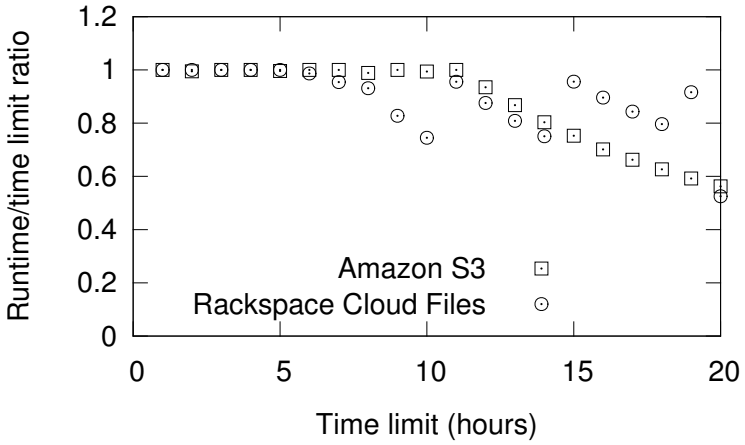


FIGURE 5.3: Ratio of actual completion time to deadline for Epigenomics workflow with 500 tasks.

and the cheapest RackSpace instances are used so the cost is low when using CloudFiles. For shorter deadlines the cost grows rapidly, since we reach the limit of 15 instances per cloud and additional instances must be spawned on a different provider, making the transfer costs higher. This effect is amplified in Fig. 5.2b, which differs from Fig. 5.2a not only by the number of tasks but also by the data size of one layer. This means that the transfer costs are growing more rapidly, so it becomes more economical to store the data on Amazon EC2 that provides more powerful instances required for short deadlines.

One interesting feature of our model is that for longer deadlines it can find the cost-optimal solutions that have shorter workflow completion time than the requested deadline. This effect can be observed in Fig. 5.3 and is caused by the fact that for long deadlines the simple solution is to run the application on a set of the least expensive machines.

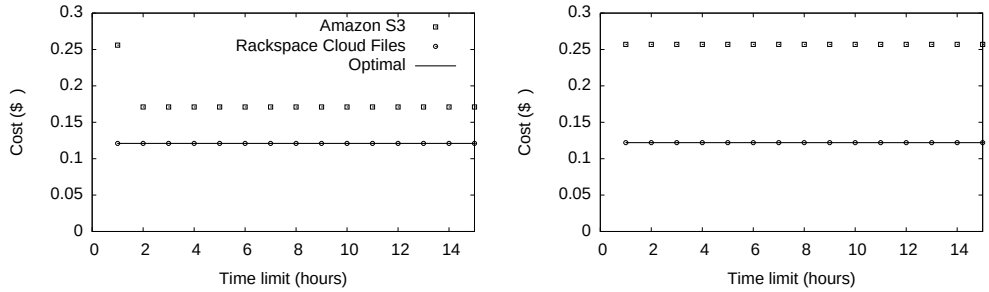
Figures 5.4 to 5.7 show results obtained for other workflows. These workflows are relatively small and even for short deadlines our model is able to schedule tasks to be executed in a very short time on cheapest instances on a single cloud, this results in flat characteristics.

To investigate how the model behaves for workflows with the same structure, but with much longer run times of tasks, we run the optimization for Montage workflow with tasks 1000× longer. This corresponds to the scenario where tasks are in the order of hours instead of seconds. The sample results in Fig. 5.8 show how the cost increases much steeply with shorter deadlines, illustrating the trade-off between time and cost. The difference between Figs. 5.6f and 5.8 illustrates that the model is more useful for workflows when tasks are of granularity that is similar to the granularity of the (hourly) billing cycle of cloud providers.

The run time of the optimization algorithm for workflows with up to 1000 tasks ranges from few seconds up to 4 minutes using the CPLEX [36] solver running on a server with 4 16-core 2.3 GHz AMD Opteron processors (model 6276), with a limit set to 32 cores. Fig. 5.9a shows that the time becomes much higher for shorter deadlines and increases for very long deadlines. This is correlated with size of search space: the longer the deadline, the search space is larger, while for shorter deadlines the problem has a very small set of acceptable solutions. The problem becomes more severe for bigger and more complex workflows like SIPHT as optimization time becomes very high (Fig. 5.9b).

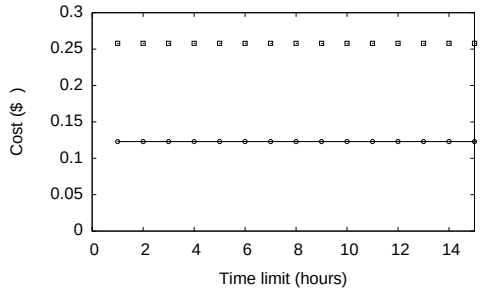
5.4 Summary

In this chapter, we presented a cost optimization model for scientific workflows executing on multiple heterogeneous clouds. The model, formulated in AMPL, allows us to find the optimal assignment of workflow tasks, grouped into layers, to cloud instances. The model was tested on a set of benchmark workflows and we observed that it gives useful solutions in a reasonable amount of computing time. By solving the model for multiple deadlines, we can produce trade-off plots, showing how the cost depends on the deadline. Such plots are a step towards a scientific cloud workflow calculator, supporting resource management decisions for both end-users and workflow-as-a-service providers.

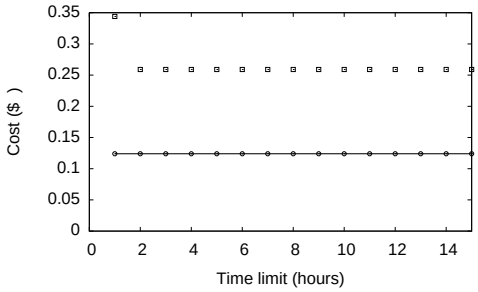


(A) 100 tasks

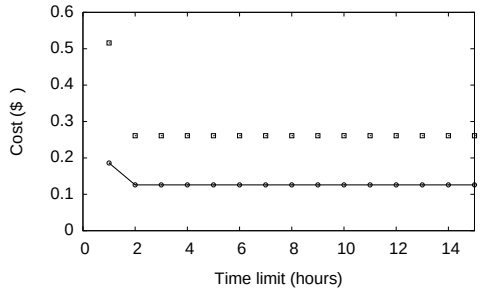
(B) 200 tasks



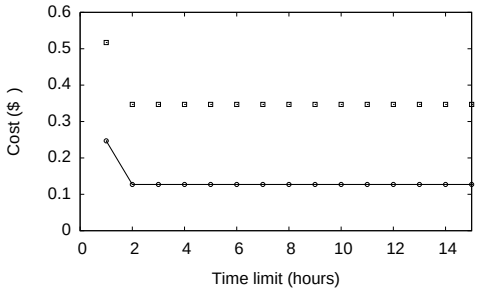
(C) 300 tasks



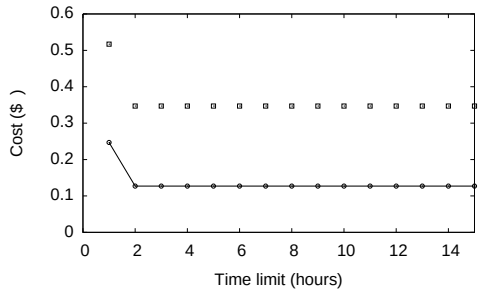
(D) 400 tasks



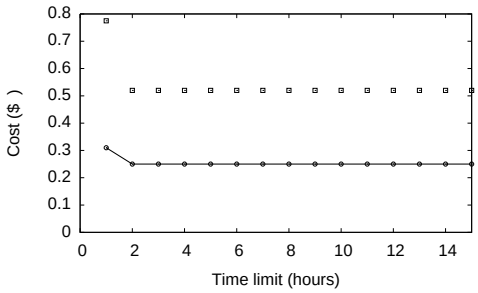
(E) 500 tasks



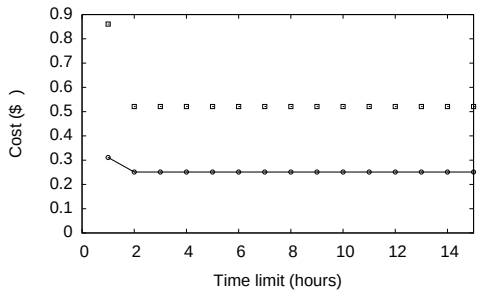
(F) 700 tasks



(G) 600 tasks

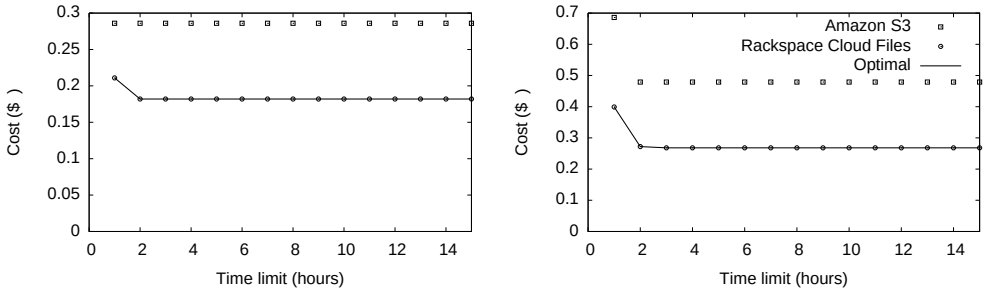


(H) 900 tasks



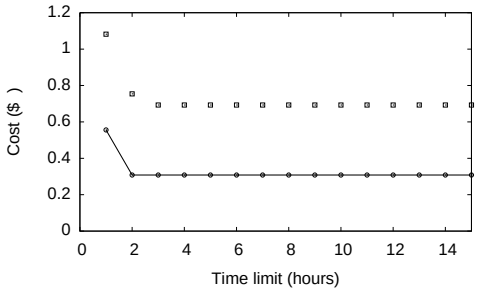
(I) 1000 tasks

FIGURE 5.4: Optimal cost found by the model for CyberShake workflow.

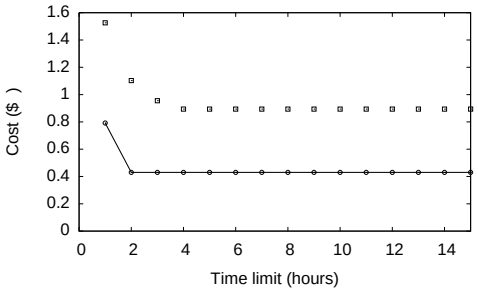


(A) 100 tasks

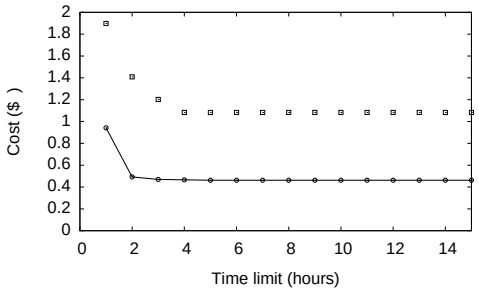
(B) 200 tasks



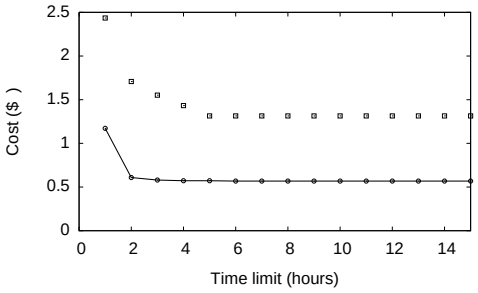
(C) 300 tasks



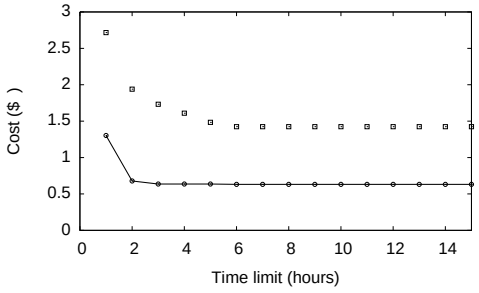
(D) 400 tasks



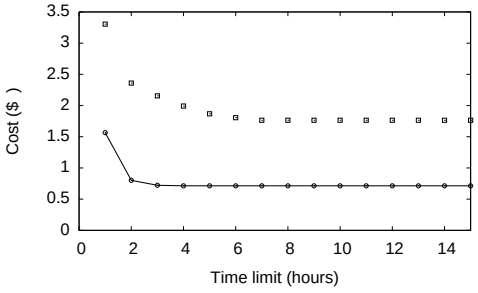
(E) 500 tasks



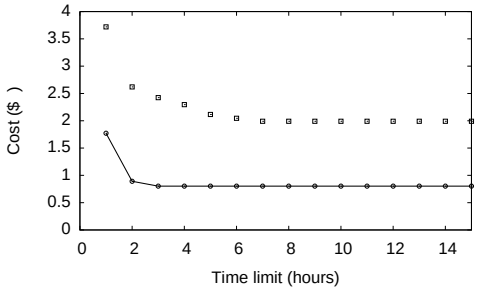
(F) 600 tasks



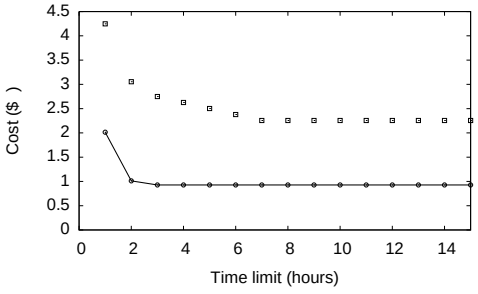
(G) 700 tasks



(H) 800 tasks

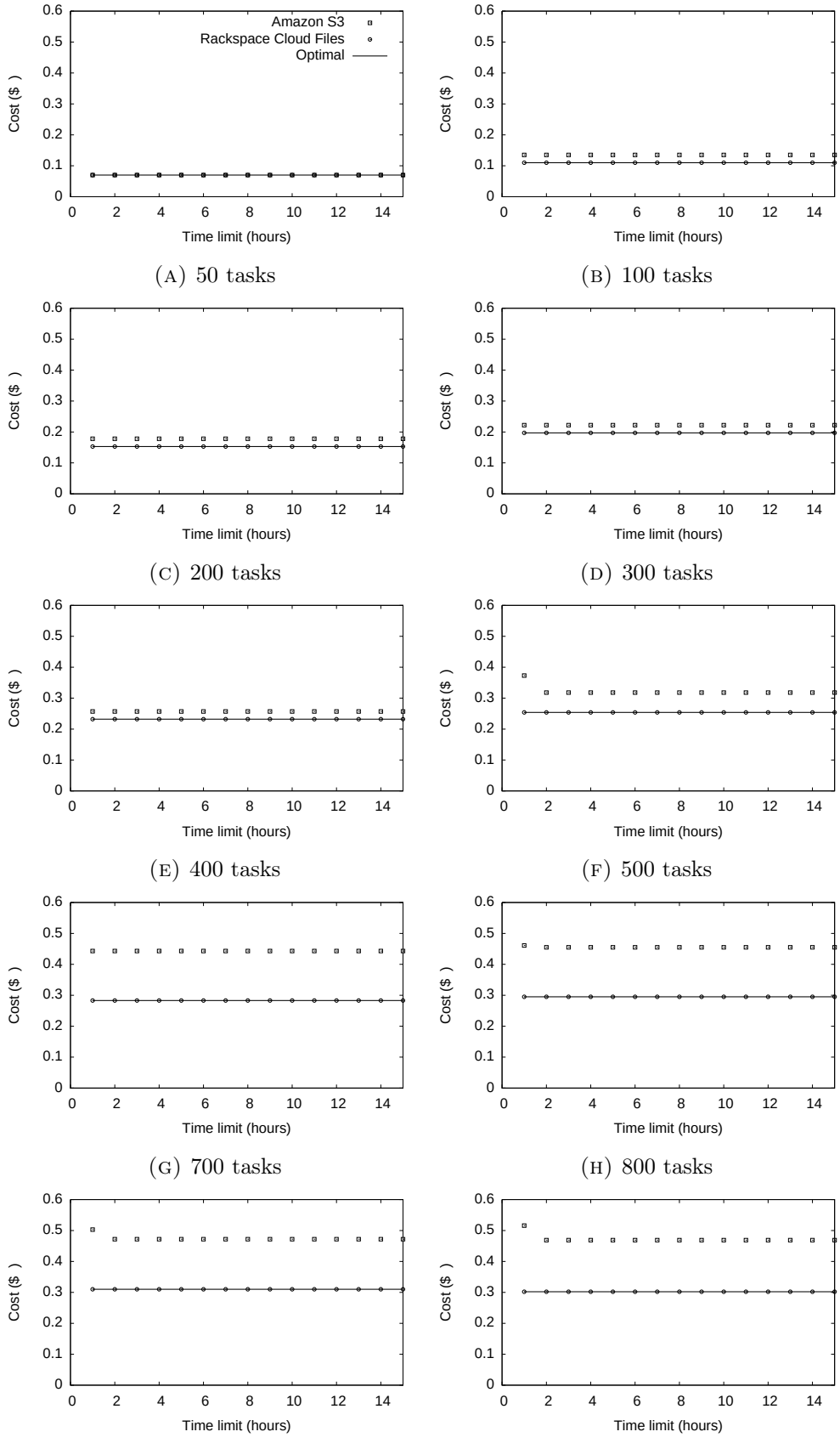


(I) 900 tasks



(J) 1000 tasks

FIGURE 5.5: Optimal cost found by the model for LIGO workflow.



(A) 50 tasks (B) 100 tasks

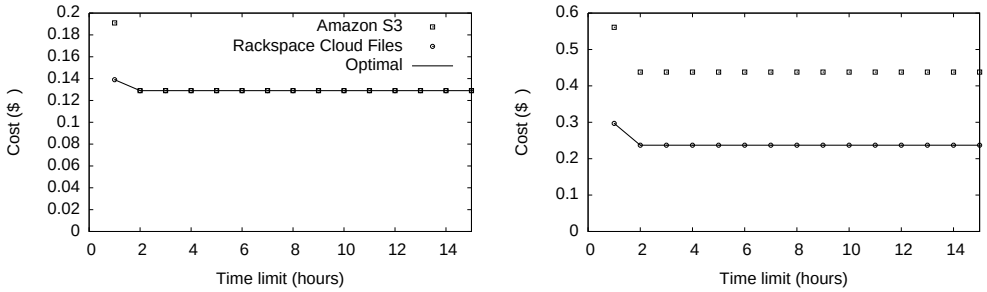
(C) 200 tasks (D) 300 tasks

(E) 400 tasks (F) 500 tasks

(G) 700 tasks (H) 800 tasks

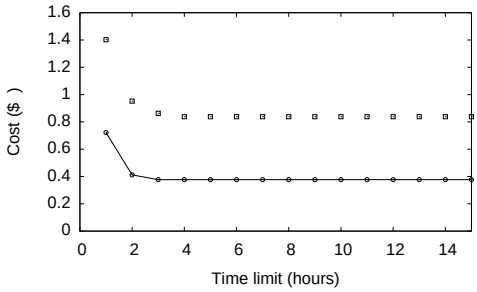
(I) 900 tasks (J) 1000 tasks

FIGURE 5.6: Optimal cost found by the model for Montage workflow.

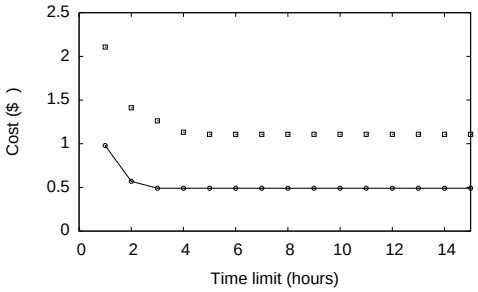


(A) 50 tasks

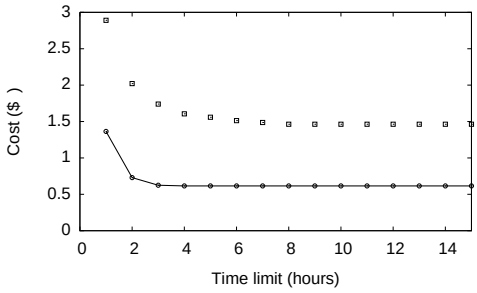
(B) 200 tasks



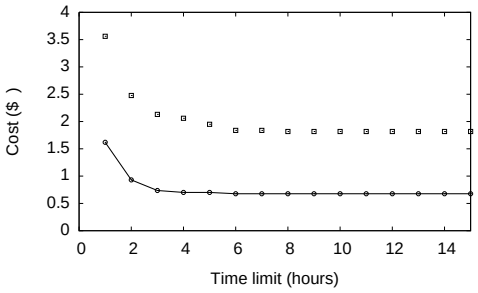
(C) 400 tasks



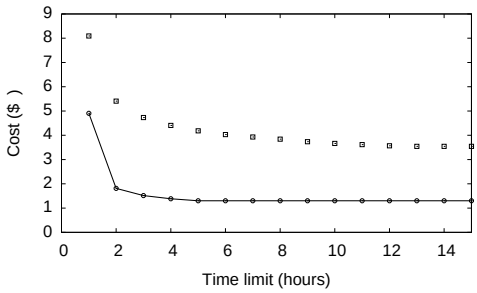
(D) 600 tasks



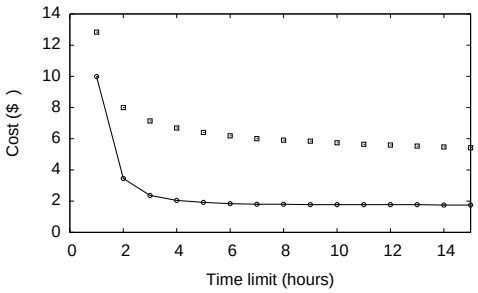
(E) 800 tasks



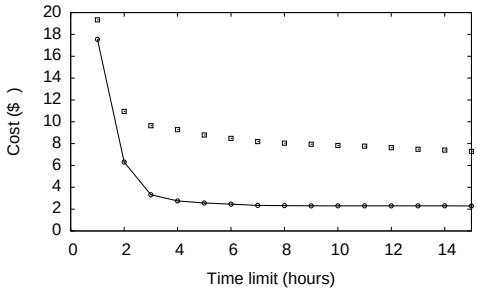
(F) 1000 tasks



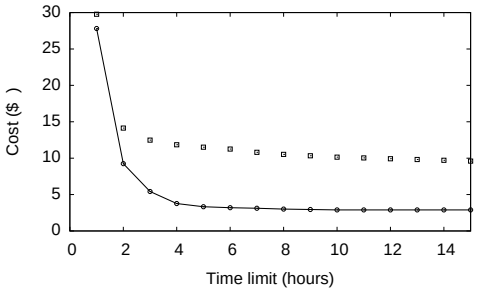
(G) 2000 tasks



(H) 3000 tasks



(I) 4000 tasks



(J) 5000 tasks

FIGURE 5.7: Optimal cost found by the model for SIPHT workflow.

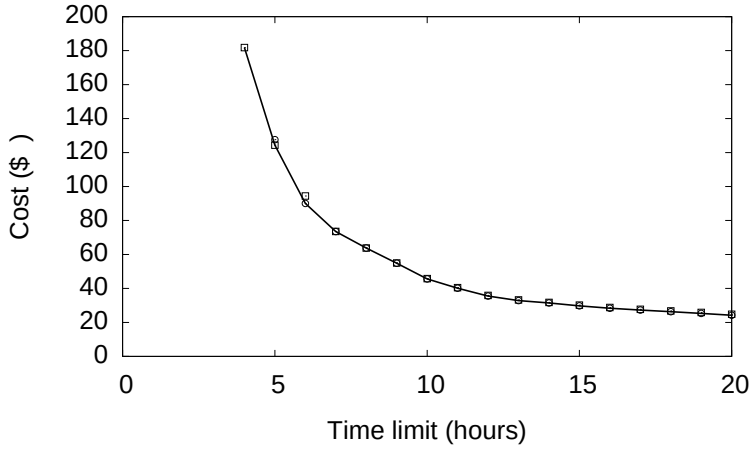
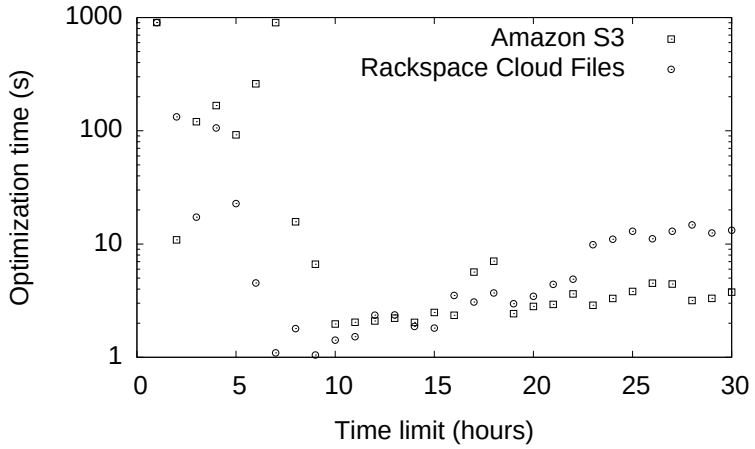
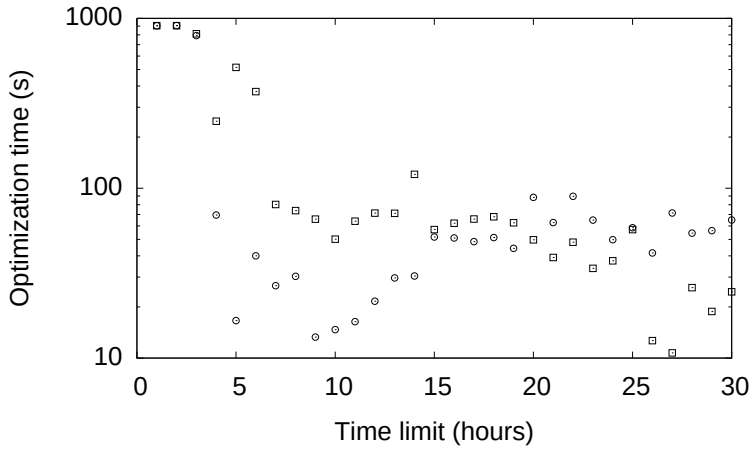


FIGURE 5.8: Optimal cost obtained by the model for Montage 500 workflow with tasks runtimes artificially multiplied by 1000.



(A) Epigenomics, 600 tasks



(B) SIPHT, 5000 tasks

FIGURE 5.9: Solver execution wall time.

Chapter 6

Conclusions and future work

6.1 Conclusions

The results presented in this thesis illustrate typical problems when making decisions on deployment planning on clouds and how they can be addressed using optimization techniques. The major goal of this thesis was the theoretical and practical investigation of optimization of resource allocation on the cloud by using integer linear programming tools and methods. To realize this goal, it required an analysis of cloud computing model, existing workflow scheduling and resource allocation algorithms, as well as mathematical programming – that were presented in first three chapters.

To practically evaluate integer linear programming approach to the problem, we defined application model of bag of tasks applications and workflows. We also defined the infrastructure model of multiple heterogenous clouds, including private and public ones. The optimization model takes into account the cost of compute instances and data transfer that proved to have significant contribution to the total. The mixed integer nonlinear optimization models were then implemented in AMPL modeling language and optimized with Cbc and CPLEX solvers. The models were evaluated in terms of results, performance and solution stability by performing a parameter sweep and analyzing the results.

We conclude that the integer linear programming proved to be a useful approach for resource allocation for scientific computing. The AMPL appeared to be friendly tool to perform such optimization.

6.2 Future work

As a future work we intend to perform experiments on real cloud infrastructure by using real applications and data sets. That would allow us to better understand cloud infrastructure deployment issues and see how dynamic environment affects offline resource allocation.

Furthermore, the infrastructure model should be extended to better reflect cloud infrastructure caveats and new cloud features such as the shorter billing cycle introduced by Cloud-Sigma or Google Compute Engine. Additional cloud services should be also considered, as they may be also used by scientific applications

Additionally, the mathematical programming approach could be applied as a subproblem solving method for other heuristic algorithms, e.g. for on-line scheduling. For that, we should investigate multi-stage optimization algorithms.

Last, but not least, the performance of workflow optimization model may be subject to improvements by simplifying application or infrastructure model or by applying other modeling techniques.

Appendix A

Source Code

Source code of the models and corresponding scripts used to generate graphs are available at public GitHub repository kfigiela/msc-thesis.

Requirements

- AMPL (version 20120804)
- CPLEX (version 12.4.0.1)
- CBC (version 2.7)
- Bonmin (version 1.5)
- Ruby (version 1.9.3)
- Bash (version 3.2)

Repository layout

- *thesis* – sources of the thesis,
- *bag-of-tasks* – sources of the bag of tasks model with corresponding helper scripts,
- *workflows* – sources of the workflows model with corresponding helper scripts.

Appendix B

Publications

Below are the papers which describe results obtained when preparing this thesis. These papers already have been published or have been accepted and will be published soon. The author of this thesis is also a co–author of each of the presented papers.

List of publications

- Maciej Malawski, Kamil Figiela, Jarek Nabrzyski, *Cost minimization for computational applications on hybrid cloud infrastructures*, Future Generation Computer Systems, Volume 29, Issue 7, September 2013, Pages 1786-1794, ISSN 0167-739X, `http://dx.doi.org/10.1016/j.future.2013.01.004`.
- Maciej Malawski, Kamil Figiela, Marian Bubak, Ewa Deelman, Jarek Nabrzyski, *Cost Optimization of Execution of Multi-level Deadline-constrained Scientific Workflows on Clouds*, Proceedings of 10th International Conference on Parallel Processing and Applied Mathematics, Warsaw, Poland, 2013 (accepted).

Bibliography

[1] Ewa Deelman, Gideon Juve, Maciej Malawski, and Jarek Nabrzyski. Hosted science: Managing computational workflows in the cloud. *Parallel Processing Letters*, 2013. ISSN 0129-6264.

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

[3] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12. IEEE Computer Society Press, 2012. ISBN 978-1-4673-0804-5. URL `http://portal.acm.org/citation.cfm?id=2389026`.

[4] Marian Bubak, Marek Kasztelnik, Maciej Malawski, Jan Meizner, Piotr Nowakowski, and Susheel Varma. Evaluation of cloud providers for VPH applications. In *CCGrid2013 - 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2013.

[5] Ewa Deelman. Grids and clouds: Making workflow applications work in heterogeneous distributed environments. *International Journal of High Performance Computing Applications*, 24(3):284–298, August 2010. doi: 10.1177/1094342009356432.

[6] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 1846285194.

[7] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, Mei-Hui Su, and K. Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10. IEEE, November 2008. ISBN 978-1-4244-2827-4. doi: 10.1109/WORKS.2008.4723958. URL `http://dx.doi.org/10.1109/WORKS.2008.4723958`.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

[9] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, July 2005. ISSN 1058-9244. URL `http://dl.acm.org/citation.cfm?id=1239649.1239653`.

[10] Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Aleksandra Nenadic, Ian Dunlop, Alan Williams, Thomas Oinn, and Carole Goble. Taverna, reloaded. In M. Gertz, T. Hey, and B. Ludaescher, editors, *SSDBM 2010*, Heidelberg, Germany, June 2010. URL `http://www.taverna.org.uk/pages/wp-content/uploads/2010/04/T2Architecture.pdf`.

[11] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, September 2005. ISSN 0163-5808. doi: 10.1145/1084805.1084814. URL `http://doi.acm.org/10.1145/1084805.1084814`.

[12] Rubing Duan, R. Prodan, and Xiaorong Li. A sequential cooperative game theoretic approach to Storage-Aware scheduling of multiple Large-Scale workflow applications in grids. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 31–39. IEEE, 2012. ISBN 978-1-4673-2901-9. doi: 10.1109/Grid.2012.14. URL `http://dx.doi.org/10.1109/Grid.2012.14`.

[13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL `http://doi.acm.org/10.1145/1327452.1327492`.

[14] Peter Mell and Timothy Grance. The nist definition of cloud computing. *NIST Special Publication 800-145*. URL `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.

[15] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002. ISSN 1045-9219. doi: 10.1109/71.993206.

[16] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 229–238, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0552-5.

[17] Hyunjoo Kim, Yaakoub el-Khamra, Ivan Rodero, Shantenu Jha, and Manish Parashar. Autonomic management of application workflows on hybrid computing infrastructure. *Sci. Program.*, 19:75–89, April 2011. ISSN 1058-9244.

[18] M. Mazzucco, M. Vasar, and M. Dumas. Squeezing out the cloud via profit-maximizing resource allocation policies. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 19–28, 2012. doi: 10.1109/MASCOTS.2012.13.

[19] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 43–52, 2010. doi: 10.1109/CCGRID.2010.80.

[20] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158 – 169, 2013. ISSN 0167-739X. doi: 10.1016/j.future.2012.05.004. URL `http://www.sciencedirect.com/science/article/pii/S0167739X12001008`.

[21] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0771-0. doi: 10.1145/2063384.2063449. URL `http://dx.doi.org/10.1145/2063384.2063449`.

[22] Juan Jose Durillo Barrionuevo, Hamid Mohammadi Fard, and Radu Prodan. Moheft: A multi-objective list-based method for workflow scheduling. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, December 3-6, 2012*, pages 185–192, 2012.

[23] Luiz Fernando Bittencourt and Edmundo Roberto Mauro Madeira. Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. *J. Internet Services and Applications*, 2(3):207–227, 2011.

[24] Ivona Brandic, Sabri Pllana, and Siegfried Benkner. Specification, planning, and execution of qos-aware grid workflows within the amadeus environment. *Concurrency and Computation: Practice and Experience*, 20(4):331–345, 2008. ISSN 1532-0634. doi: 10.1002/cpe.1215.

[25] Suraj Pandey, Adam Barker, Kapil Kumar Gupta, and Rajkumar Buyya. Minimizing Execution Costs when Using Globally Distributed Cloud Services. In *24th*

IEEE International Conference on Advanced Information Networking and Applications, pages 222–229. IEEE Computer Society, 2010.

[26] T.A.L. Genez, L.F. Bittencourt, and E. R M Madeira. Workflow scheduling for saas / paas cloud providers considering two sla levels. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 906–912, 2012. doi: 10.1109/NOMS.2012.6212007.

[27] Enforcing qos in scientific workflow systems enacted over cloud infrastructures. *Journal of Computer and System Sciences*, 78(5):1300 – 1315, 2012. ISSN 0022-0000.

[28] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. *Future Generation Computer Systems*, 29(4):973 – 985, 2013. ISSN 0167-739X. doi: 10.1016/j.future.2012.12.012. URL `http://www.sciencedirect.com/science/article/pii/S0167739X12002324`.

[29] "programming". Oxford Dictionaries. Oxford University Press. URL `http://oxforddictionaries.com/definition/english/programming`.

[30] George Dantzig. *Linear Programming and Extensions*. Princeton University Press, August 1998. ISBN 0691059136. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0691059136`.

[31] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

[32] General Algebraic Modeling System (GAMS). URL `http://www.gams.com`.

[33] PuLP: an LP modeler written in Python. URL `https://projects.coin-or.org/PuLP`.

[34] OscaR: Operational Research in Scala. URL `https://bitbucket.org/oscarlib/oscar/wiki/Home`.

[35] John Forrest. Cbc (coin-or branch and cut) open-source mixed integer programming solver, 2012. URL `https://projects.coin-or.org/Cbc`.

[36] IBM. Cplex solver, 2013. URL `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[37] Solvers that Work with AMPL. URL `http://www.ampl.com/solvers.html`.

[38] Bonmin (Basic Open-source Mixed INteger programming). URL `http://projects.coin-or.org/Bonmin`.

[39] Amazon EC2 FAQ. URL `http://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2`.

[40] CloudHarmony. Cloud benchmarks `http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html`, 2011. URL `http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html`.

[41] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, and Nicolas Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, May 2008. ISSN 15725286. doi: 10.1016/j.disopt.2006.10.011.