Efficient allocation for distributed and connected Cloud

Thibaud Ecarot

THESE DE DOCTORAT CONJOINTE TELECOM SUDPARIS ET
UNIVERSITÉ PIERRE ET MARIE CURIE

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Thibaud Ecarot

Pour obtenir le grade de

DOCTEUR DE TELECOM SUDPARIS

Allocation efficace de ressource Cloud dans l'intérêt du fournisseur et des consommateurs

---

Soutenue le : 29 septembre 2016

devant le jury composé de :

Prof. François Spies	Rapporteur	Université de Franche-Comté, France
MC. Lila Boukhatem	Rapporteur	University Paris-Sud, France
Prof. Pierre Sens	Examinateur	UPMC, France
Prof. Yacine Ghamri-Doudane	Examinateur	Université de La Rochelle, France
Cédric Brandily	Encadrant	Thales Services, France
Prof. Djamal Zeghlache	Directeur de thèse	Télécom Sud Paris, France

Thèse N°: 2016TELE0017

Résumé

Institut Mines-Telecom
Telecom Sud Paris

Allocation efficace de ressource Cloud dans l'intérêt du fournisseur et des consommateurs

par le doctorant Thibaud ECAROT

Dans ce travail, nous nous intéressons à la modélisation des ressources Cloud, indépendamment des couches existantes, afin d'apporter un cadre (framework) de représentation unique et ouvert à l'arrivée anticipée du XaaS (Anything as a Service). Nous fournissons, à l'aide de ce framework, un outil de placement des ressources pour une plate-forme donnée. Les travaux de thèse se portent aussi sur la prise en compte des intérêts des utilisateurs ou consommateurs et des fournisseurs. Les solutions existantes ne se focalisent que sur l'intérêt des fournisseurs et ce au détriment des consommateurs contraints par le modèle d'affaire des fournisseurs. La thèse propose des algorithmes évolutionnaires en mesure de respecter les objectifs et intérêts des deux acteurs.

JOINT THESIS BETWEEN TELECOM SUDPARIS AND PIERRE AND MARIE CURIE UNIVERSITY

Doctoral school: Informatique, Télécommunications et Electronique de Paris

Presented by

Thibaud Ecarot

For the degree of

DOCTEUR DE TELECOM SUDPARIS

Efficient allocation for distributed and connected Cloud

Defense date: September, 29 2016

Jury Members:

Prof. François Spies	Evaluator	Franche-Comté University, France
L. Lila Boukhatem	Evaluator	Paris-Sud University, France
Prof. Pierre Sens	Examiner	UPMC, France
Prof. Yacine Ghamri-Doudane	Examiner	La Rochelle University, France
Cédric Brandily	Evaluator	Thales Services, France
Prof. Djamal Zeghlache	Thesis advisor	Télécom Sud Paris, France

Thesis N°: 2016TELE0017

Abstract

Institut Mines-Telecom
Telecom Sud Paris

Efficient allocation for distributed and connected Cloud

by PhD student Thibaud Ecarot

This thesis focuses on optimal and suboptimal allocation of resources in cloud infrastructures taking jointly into account the consumers' and providers' interests. A mathematical model of this joint optimization problem serves as a basis to propose evolutionary algorithms well suited for multiple objectives, criteria and constraints. The proposed algorithms are compared to the state of the art that has so far remained provider-centric. Our algorithms optimize the dynamic allocation of cloud resources while considering both the users and the providers objectives and requirements in order to avoid provider lock-in where end users are tied to the provider's business models.

Acknowledgements

This thesis would not have been possible without the help, support and patience from my principal advisor, Pr. Djamal Zeghlache. The good advice, support and friendship of my second supervisor, research engineer, Cedric Brandily, has also been invaluable to me on both an industrial and personal level and for which I am extremely grateful.

I would like to acknowledge the financial, academic and technical support of Telecom Sud Paris, Thales Services SAS and its staff, particularly the computer facilities of the Thales Services SAS infrastructure, as well as the work environment of Telecom Sud Paris. I am also thankful to the French ministry of higher education and scientific research with national association of research and technology for their administrative and financial support.

I would also like to thank my wife, Laura, for her support and patience during the writing of my thesis. I warmly thanks my family for their valuable assistance. Last, but by no means least, my thanks go to my friends in Britain and America amongst others for their support and encouragement.

Contents

List of Figures

List of Tables

Abbreviations

NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
QoS	**Q**uality **A**s **O**f **S**ervice
IaaS	**I**nfrastructure **A**s **A** **S**ervice
PaaS	**P**latform **A**s **A** **S**ervice
SaaS	**S**oftware **A**s **A** **S**ervice
VM	**V**irtual **M**achine
DRS	**D**istributed **R**esource **S**cheduler
DPM	**D**istributed **P**ower **M**anagement
CPU	**C**entral **P**rocessing **U**nit
RAP	**R**esource **A**llocation **P**roblem
MORAP	**M**ulti **O**bjective **R**esource **A**llocation **P**roblem
OCCI	**O**pen **C**loud **C**omputing **I**nterface

Foreword

Following a joint consultation and brainstorming with both Cedric Brandily and Djamal Zeghlache, we observed that there existed an operational problem within cloud platforms. Cedric Brandily is a research engineer at Thales European Research Centre for eGov and Secured information Systems (ThereSIS) based in Palaiseau, France. He is also my supervisor at the company. Professor Djamal Zeghlache of Telecom Sud Paris is my thesis advisor and a member of the R3S team of the SAMOVAR UMR 5157 research unit. The CNRS (Centre National de la Recherche Scientifique) research unit SAMOVAR (Distributed Services, Architectures, Modeling, Validation and Network Administration) regroups around 50+ full time faculty and 100 research assistants, from Télécom SudParis and CNRS, working in the field of services, networks and telecommunications. ThereSIS is a private research laboratory involved in research, development and innovation in cloud computing, parallel and distributed computing.

This thesis represents the learning and acquired know how over a period of nearly three years (2013-2016). The first questions were raised when providers of cloud platforms realized that infrastructure costs and energy consumption in data centers can be reduced significantly through smarter resource allocation and placement. However, due to problem complexity and scalability issues, optimal placement and allocation have turned out to be more difficult to achieve than expected. This has been especially exacerbated by the amount of available resources in large data centers, the heterogeneity in terms cloud resources and services themselves and the presence of multiple stakeholders. Fully optimizing the use and exploitation of cloud infrastructure remains for these reasons a challenge that we nevertheless believe can be partially addressed by unified models and appropriate optimization methods and approaches to remove some of the hurdles and barriers.

In order to derive a unified view and/or model of the general cloud resources allocation problem, the first goal has been to analyze, understand and classify existing resource allocation mechanisms and schedulers in the current literature, in open source communities and in commercial solutions. My primary objective was consequently to analyze

optimal and suboptimal resource allocation algorithms and to develop a methodology, a model and tools for new and evolved placement algorithms that can take into account usage and exploitation costs, heterogeneous resources and quality of service.

Chapter One is an introduction to dynamic resource provisioning in cloud computing. This chapter presents the cloud computing paradigm and the general concepts of cloud schedulers and describes the context and ensuing objectives and organization of the thesis.

Chapter Two describes the background and the methods available in the literature in the first section. The chapter presents the metrics used in cloud environments, the related multi-dimensional packing and scheduling problems and the algorithms proposed by the current literature to solve them. The first section gives a complete description of multi-dimensional packing and scheduling problems. The notion of permutation in combinatorics used to resolve these problems is described in section one. Section two is a review of the metrics used in a cloud to compare the scheduling methods. The third and final section lists some relevant algorithms for the cloud resource allocation problem.

Chapter Three introduces a new and generic model for cloud placement and resource allocation. The problem is modeled as a packing problem. We outline and apply a single objective and multiple objectives to these problems while defining constraints for the allocation.

Chapter Four develops a genetic algorithm to solve the addressed cloud resource allocation problem. We express a graph breakdown method for a more efficient data structure for solving the problem. We describe all the operators of our proposed genetic algorithm and the parameters used in the model. To improve convergence time, we define our own initialization and crossover operators. We perform an evaluation and comparison of our algorithm with the metric defined in section 2.2 and with another algorithm commonly found in the industry (Section 2.4).

The closing chapter summarizes the contributions to the cloud resource allocation problem. Future directions and perspectives for further research are also described in this chapter.

Chapter 1

Introduction

Abstract. We first describe cloud computing and the problems raised by cloud resources allocation in an infrastructure as a service context. We then summarize our contributions to this allocation problem and present the organization of the manuscript.

This first section is a general overview of cloud computing to gain knowledge on its principles, properties and resource allocation and placement challenges. The notion of scheduler essential in planning end users' cloud computing jobs and tasks, and central in organizing resource allocation in the time dimension, is also presented.

1.1 Dynamic Resource Provisioning in Cloud Computing

Scheduling of jobs in shared infrastructures first appeared in mainframe computers and servers and moved thanks to advances in the IT world into computer grids involving distributed servers and hosting platforms across multiple and geographically spread sites. However, in these infrastructures the jobs and tasks scheduling and associated compute, storage and communications resources allocations do not take into account the complex business relationships and interactions between end users, consumers and providers as envisioned by cloud computing. The latter is a paradigm shift from the servers, clusters and grids services model. Cloud computing introduces more flexibility in consumers and providers interactions by introducing a dynamic and on demand resources and services provisioning and management grounded on a pay per use principle.

In cloud computing various resources such as infrastructure resources or software components are provided as a service are provided, consumed and billed on a need and use basis. In public clouds, services or capacity are leased following subscriptions or via a Pay-per-Use principle for customers using these services. Customers interact with the cloud platform through a user interface or a programming interface (API: Application Programming Interface) managed by the provider.

There are multiple definitions of cloud computing in the current literature but a definition commonly adopted is that of the National Institute of Standards and Technology (NIST). In the NIST definition, Peter Mell and Tim Grance define cloud computing as follows: "*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. This cloud model is composed of five essential characteristics, three service models, and four deployment models.*" [2]

The essential characteristics [3] of cloud model are **on-demand resources and services provisioning** with guaranteed Quality of Service (QoS). Cloud resources or services can be **automatically reconfigured**, **orchestrated** and **consolidated** to present

a single users' access to the cloud platform. Cloud computing platforms are **flexible** to adapt to various requirements of a potentially large number of users.

There are several deployment models [4] such as private cloud dedicated exclusively to single organization. Cloud technology is called a "Public Cloud" when the services are rendered over a network that is open for public use. Community clouds share infrastructure resources for several organizations from a specific community with common concerns (security, compliance or jurisdiction). Finally, hybrid cloud is a composition of two or more distinct cloud infrastructures (private, community, or public).

There are three cloud services models [5]. Software as a Service, SaaS, which provides customers with applications running on a cloud infrastructure. Platform as a Service, PaaS, allowing the customer to deploy onto the cloud infrastructure customer created or acquired applications using programming languages, libraries, services, and tools supported by the provider. Infrastructure as a Service, IaaS, that offers additional resources such as a virtual-machines, disk image library, raw (block) and file-based storage, firewalls, load balancers, IP addresses, virtual local area networks (VLANs), and software bundles [6]. IaaS-cloud providers supply these resources on-demand from their large resources pools installed in data centers. For wide-area connectivity, customers can use either the Internet or carrier clouds (dedicated virtual private networks).

Clouds involve multiple stakeholders but this thesis focuses on two key players that are the customers and providers. This focus is justified by the fact that the client server model will apply to all combinations of pairs of stakeholders in the value and business chains. The consumer typically customize their resources and services via service templates presented by resource providers. The providers share and rent their infrastructures and provide IaaS, PaaS and SaaS services. The customers rely on the providers *aaS offers to transparently design or compose their services thanks to the providers' presented abstractions.

There are, in our view and in our thesis focus, two generic players (see Figure 1.1)) involved: the provider and the consumer. We are particularly concerned by the different interests and objectives of these two key players. The users aim at reliable, resilient and guaranteed services at lowest possible cost while the providers want to minimize their own costs and maximize their revenues. These partly conflicting interests have not been addressed jointly and simultaneously in the current resource allocation and management research. The objective of the thesis is to find balanced solutions that can conciliate these potentially conflicting objectives and interests.

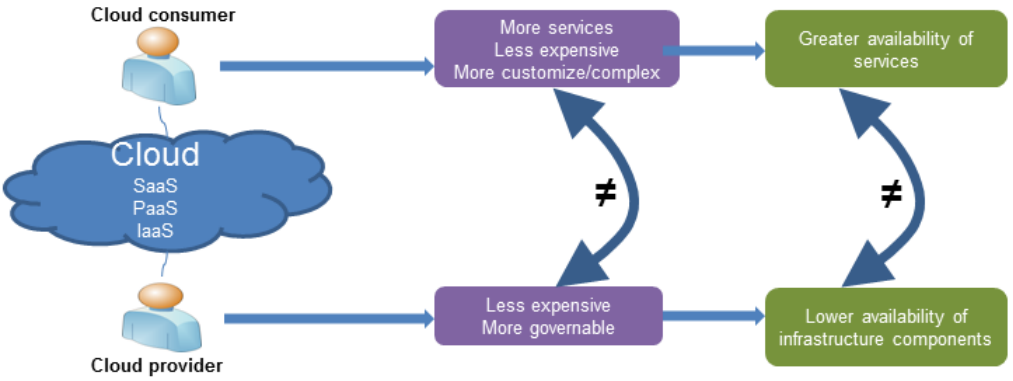

FIGURE 1.1: Various players within Cloud platform

Customer needs are constantly changing, cloud computing customers are expected to bear the costs related to their new business models. In addition to the costs of compliance, there are the costs generated by the remote infrastructure going off-line or becoming unavailable. That is the reason why some cloud customers are turning to private or hybrid infrastructures instead. The term "private" refers to the fact that a cloud computing platform is not shared and cause less concerns in terms of data protection and security. The public clouds or provider models are not well suited for certain consumers and service providers requirements. The consumers and providers relationships need to evolve beyond the current providers business models that constrain the consumers to a specific contract, restrict the consumers possibilities and impose both the service usage and the associated costs.

Therefore, demanding customers are turning to private cloud-based infrastructure solutions which allow to operate an automated service development process. Automation is an essential process in a cloud computing platform because it enables faster delivery of services to customers, while avoiding the risk of human error. This automation can be performed on different processes such as checking compliance of the service with SLAs (Service Level Agreement), patch deployment, service version upgrade or resource allocation (resource provisioning and deprovisioning).

Cloud computing where resources and services are provided on demand on a pay-per-use basis offers the advantage of avoiding capital expenses and reducing considerably operational expenses to customers. These advantages come also with the drawback of lock in to the cloud providers model and application programming interfaces. Even if some of the issues have been addressed by cloud brokerage and portability solutions to move across cloud providers and aggregate their services, the business interests of the customers have not been taken sufficiently into account. The business contracts and APIs imposed by the providers continue to constrain the users that have to adapt to the providers' models. In fact, there is a variety of customer-provider relationships with

different technical and business objectives to cater for and to include in the way cloud resources and services are acquired, used and managed. Our objective in this work is to provide cloud resources allocation solution that address the interests of the providers and customers jointly by reflecting these goals in the model. The approach consists in integrating the technical and business requirements and constraints from the customers in addition to that of the providers.

Today, in order to manage customer requests and the whole infrastructure, a cloud platform will use a "scheduler". The "Scheduler" is designed to manage allocation of resources and services on the provider layers (*aaS). For example, within IaaS layer, a placement is decided by the scheduler on the available hypervisors for each virtual machine.

If an error occurs on a given system, schedulers can trigger placement of the impacted resources onto another hosts without any human interaction. Sometimes it may be preferable to operate maintenance on a daily basis as the scheduler brings much more improvement in preserving a resource group. But how can the scheduler choose the most suitable host? And how does it react in case of failure?

In cloud platforms, schedulers on boot read out the general settings and cloud configuration files and enter standby mode until customers request services. Some schedulers take into account information coming from embedded monitoring systems. Some schedulers process requests in batch or using queuing systems but most schedulers treat the demands successively, one at a time, as they occur.

In grids and clouds, resource allocation focused primarily on consolidation and load balancing objectives. More recently there have been attempts to introduce the notion of workflows in the allocation but turned out hard to achieve in heterogeneous clouds. The same can be said of the integration of business and technical constraints despite some of the progress and the improvements in resource allocation (using for example minimum cost maximum flow approaches).

As often in the field of cloud computing, work has been carried out originally in the field of high performance computing. The aim is to successfully place statically computation tasks represented as graphs on multiprocessors grids [7]. Based on previous work, exact and heuristic methods were provided by [8] to allocate virtual networks using subgraphs within a service delivery architecture.

1.2 Research Problems and Objectives

The thesis as previously stated focuses on resource allocation onto physical infrastructures shared and offered as a service (i.e, for a IaaS context. The resource allocation problem is addressed with the consumer and provider centric objectives in mind that require answering some key underlying questions:

- **How to measure the performance of a resource allocation algorithm in a cloud computing platform?**
- **How to represent the provider's resources and the resources requested by the users?**
- **How to obtain a feasible allocation solution in a suitable/practical time for large infrastructure sizes?**

These objectives lead us to address key scientific challenges to derive and propose efficient resource allocation solutions or algorithms:

- Suggest a template definition generalizing carrier cloud infrastructures descriptions.
- Identify relevant indicators to describe the state of the carrier infrastructure and define the information aggregation process to support the decision and allocation process.
- Solve the related NP-hard optimization problem through efficient dynamic allocation algorithms taking into account consumers' and providers' intrerests.

1.3 Contributions

We summarize for the reader convenience the contributions of the thesis with respect to the consumer and provider centric resource allocation in clouds in a IaaS context. We also take this opportunity to describe, in the sequel, the manuscript chapter organization.

The first contribution is a new generic resource model capable of describing and integrating the costs of resources offered as a service in clouds and capable on reflecting both the consumers' and providers' objectives and interests and related constraints. The proposed data representation is an extension of Open cloud computing Interface (OCCI)

model for cloud resources. We mainly worked on the description of a resource component properties. The represented resources in our work are infrastructures services but the model applies equally well to the Platform as a Service (PaaS) and Software as a Service (SaaS) paradigms. The representation includes cost of management, service interruptions, workload on cloud resources and the hosted user requests.

The second contribution is the identification and expression of affinity and anti-affinity constraints between the provider resources and the users' requested resources. These affinity and anti-affinity constraints enable tradeoffs and flexibility in the overall costs and in users' quality of service. The framework facilitates differentiation and prioritization of resources allocation to take into account critical user resources by hosting for instance these sensitive virtual resources on secure provider resources. These constraints are expressed as allocation (mapping) constraints of virtual resources onto the provider infrastructure.

The third contribution, that relies on the two previous contributions to describe and define the problem, is an evolutionary algorithm, specifically a genetic algorithm, that can meet simultaneously the conflicting users' and providers' objectives and constraints. The proposed genetic algorithm can find in practical/reasonable time ($<$ 30 seconds) good solutions for the resource allocation problem for scenarios involving midsize clouds (in the order of 2000 virtual machines and 500 hypervisors). We are in favor of genetic algorithms because they can find multiple near optimal solutions, if well configured and tuned, solutions have chosen to use a genetic algorithm, in sufficiently short durations. Our genetic algorithm has been modified to improve convergence time of the produced populations towards a feasible solution. We improved the initialization process of the population so each new generated individual respects the constraints and this speeds the convergence to a viable solution.

1.4 Thesis Organization

The thesis manuscript is organized in chapters depicted in Figure 1.2.

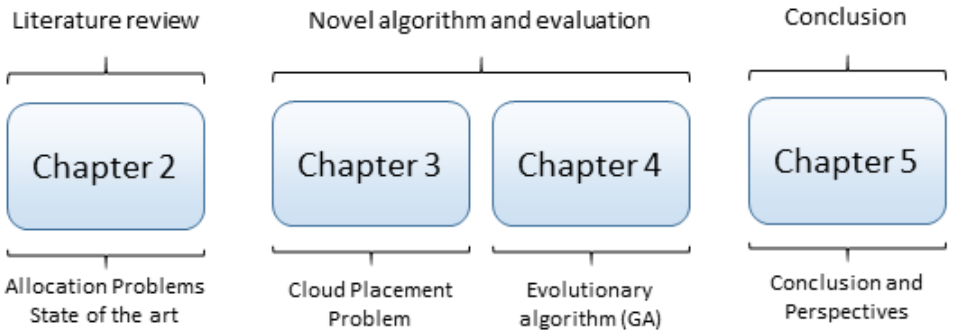

FIGURE 1.2: Thesis organization

Chapter 2 describes the state of the art resource representations and allocation methods. This state of the art is divided into three subsections: a multi-dimensional resource allocation problem description, a presentation of performance metrics, and a series of relevant algorithms for the cloud resource allocation problem. The first subsection presents various theoretical formulations of the addressed problem and resolution methods using combinatorial partial permutations. The second subsection describes as indicated the performance indicators and typically used metrics to assess efficiency and performance. The last subsection presents resource allocation methods used by large companies providing cloud computing services.

Chapter 3 describes a new algorithm using a generic model for the cloud resource allocation problem. The IaaS resource allocation problem, addressed by this thesis, is modeled as a multi-dimensional bin packing with both single and multi-objectives and associated constraints that reflect the users' and providers' interests.

Chapter 4 presents our proposed approach that uses a genetic algorithm to find solutions to the resource allocation problem while taking into account the stakeholders' objectives and constraints. The chapter describes the decomposition of the consumer or user requests and the provider physical infrastructure into graphs for a more efficient matrix representation of the problem. The genetic algorithm parameters and modified operators are specified along with our introduction of our own genetic algorithm will be described along with our own initialization and crossover operators to improve the solutions. We assess performance and compare with a state of the art commercial and industrial solution (section 2.4) using the metric defined in Section 2.2.

In the final chapter 5 we summarize our contributions to the cloud resource allocation problem and sketch perspectives for future research and investigations.

Chapter 2

Placement and Scheduling Problems - State of the art

Abstract. This chapter reviews the relevant state of the art for the resource allocation and scheduling problem addressed in this thesis. The multi-dimensional packing and scheduling problems and their current academic and industrial implementations and combinatorial partial permutations used to treat such resource allocation problems are presented. Typical performance indicators used to assess and evaluate solutions for cloud resources allocation and scheduling are described. The indicators are classified into three main categories: technical, business and pricing indicators. We present the most commonly used exact and heuristic algorithms with or without constraints to set the background for our proposals and include the commercial or cloud services providers solutions.

2.1 Allocation Problem Definition and Resolution

Since resource allocation and scheduling both rely on combinatorial optimization, we review briefly the history of this field. Indeed, allocation and scheduling have been the subject of combinatorial optimization for quite a long time in past work such as [9] [10] [11] [12]. The resource allocation and scheduling problems had a major role at the beginning of World War II for instance where there was a need to optimally place military equipment in cargos. Linear programming was used for the purpose in the 1930s [13] [14]. In the late 1930s, applications of linear programming were first seriously attempted by Leonid Kantorovich (a Soviet mathematician) and by Wassily Leontief (an American economist) respectively for manufacturing schedules in economics.

Leonid Kantorovich obtained the 1975 Nobel Prize in economics for his contribution to the optimal resource allocation problem [15]. The Sveriges Riksbank Prize in Economic Sciences, in Memory of Alfred Nobel, was awarded jointly to Leonid Vitaliyevich Kantorovich and Tjalling C. Koopmans "for their contributions to the theory of optimum allocation of resources" [16] [17]. They were able to solve these problems with linear programming which consequently gained momentum in 1947 with the introduction of the simplex method by George Dantzig that has since simplified considerably the resolution of linear programs [18] [19].

The mathematical definition of these problems has evolved over time and several approaches and mathematical tools can be used to address resource allocation and scheduling such as: linear programming, integer programming, decision analysis, dynamic programming or network analysis with critical path method to cite a few.

A typical mathematical problem is to maximize profits or reduce costs by using a single objective function with an associated set of constraints. In the case of a linear program, the objective function and constraints are all linear functions. The allocation problem of virtual resources in a cloud computing platform is a more recent topic and challenge that is similar to the problems of load balancing and storage in a multidimensional knapsack. The objective functions and constraints are becoming increasingly complex, however, in the context of clouds characterized by allocation and provisioning flexibility and highly dynamic operation.

We now specialize the description, analysis and presentation to our central problem: resource allocation and scheduling in d-dimensions corresponding to the cloud computing resource provisioning and management. We outline a general definition for the resource allocation and scheduling and present vector bin packing, vector scheduling and view the entire problem as "Packing Integer Problem". We pay special attention to combinatorial partial-permutations that are used to address these problems.

2.1.1 Multi-dimensionnal Packing Problems

In this thesis, we study multi-dimensional generalizations of bin packing, load balancing and knapsack problems. Indeed, the generalization of resource allocation problems within a cloud computing platform is an aggregate of these well-known generalizations. We provide a definition of the multi-dimensional bin packing and the load balancing generalizations. We examine the generalization of the knapsack problem. For each definition, we give an example using an integer programming formulation.

2.1.1.1 Vector Bin Packing

Vector bin packing, very well known in operational research [20] [21], is a static model representing resource allocation problem with known service demands and a set of servers with known capacities [22]. The capacity of the servers and the service demands span several dimensions. There is a renewed interest in this problem because it adapts well to the virtual machine allocation problem in clouds. The objective is to minimize the number of bins needed to allocate n resources. Resources must be allocated only once and the allocation must comply with the bins capacity limits.

Definition 2.1. Vector Bin Packing. Given a set of n rational vectors $V = \{v_1, \ldots, v_n\}$ from $[0,1]^d$ where d is the number of dimension. We have to find a partition of the set into sets $B = \{B_1, \ldots, B_m\}$ whose sum of its vectors is at most equal to 1 in every dimension, $\left\| \sum_{j=1}^{m} v \in B_j^v \right\| \leq 1 \ \forall v \in V$. The objective is to minimize the number of bins needed to allocate n resources.

We can write this problem with an integer programming formulation [23]. We use boolean variable x_{ij} to indicate that resource i is allocated on bin j and set $x_{ij} = 1$, $\forall i \in [0,n]; \forall j \in [0,m]$. We have two capacity variables for items and for bins. $W_{d,m}^I$ representing the weight of item n on the dimension d. $W_{d,m}^B$ representing the weight of bin m on the dimension d. $U_j = 1$ if bin j is used. Table 2.1 lists the Vector Bin Packing Problem variables:

TABLE 2.1: Vector Bin Packing Problem Variables

$x_{i,j}$	Boolean variable indicating whether item i is assigned to bin j
$W_{d,i}^I$	Weight of item i on the dimension d
$W_{d,j}^B$	Weight of bin j on the dimension d
U_j	Boolean variable indicating that Bin j is used

We obtain a description of Vector Bin Packing problem as an integer program:

$$min \sum_{j=1}^{m} U_j \tag{2.1}$$

subject to:

$$\forall j, d \sum_{i=1}^{n} W_{d,i}^{I} x_{i,j} \leq W_{d,j}^{B} U_j \tag{2.2}$$

$$\forall i \sum_{j=1}^{m} x_{i,j} = 1 \tag{2.3}$$

$$x \in \{0,1\}, U \in \{0,1\}$$

2.1.1.2 Vector Scheduling

Scheduling is commonly used to address allocation problems in many areas such as allocating tasks to processors [24] or managing gates at airports [25]. The objective is to find an assignment of the tasks that minimizes load over all bins.

Definition 2.2. Vector Scheduling [26]. Given a set of n rational vector of tasks $T = \{t_1, \ldots, t_n\}$ from $[0,1]^d$ where d is the number of dimension, we have to find a partition of the set into sets $B = \{B_1, \ldots, B_m\}$ with maximum load on any bin in any dimension is at most equal to 1, $\left\| \sum_{t \in T_j}^{m} T \right\|_{\infty} \leq 1$ where T_j is the set of tasks assigned to bin j. The objective is to find an assignment of the tasks that minimizes load over all bins.

We can write this problem in the form of a generalized load balancing problem with an integer program formulation. Let $T = \{t_1, \ldots, t_n\}$ be a set of tasks and $B = \{B_1, \ldots, B_m\}$ be a set of bins with d dimensions $[0,1]^d$. Define x_{ij} as a boolean variable with $x_{ij} = 1$ if task i is allocated on bin j. Matrix $x \in \{0,1\}^{|T| \times |B|}$ represents the assignment matrix. If task i is assigned to bin j, there is non-negative cost $C_{ijj'}$ associated to machine j'. The objective is to minimize load over all bins. Table 2.2 gives an example of the Vector Scheduling Problem variables:

TABLE 2.2: Vector Scheduling Problem variables

$x_{i,j}$	Boolean variable indicating whether task i is assigned to bin j
$C_{ijj'}$	Non-negative cost of machine j' if task i is assigned to bin j

We obtain a description of Vector Scheduling problem as an integer program [27]:

$$min_x max_{j'} \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} C_{ijj'} \tag{2.4}$$

subject to:

$$\forall i \sum_{j=1}^{m} x_{i,j} = 1 \tag{2.5}$$

$$x \in \{0,1\}, \forall i \in T, j \in B$$

2.1.1.3 Packing Integer Program

The Packing Integer Problem is equivalent to the knapsack problem with multiple dimensions [8] [28]. The knapsack problem consists in filling a bag with items. A natural constraint is that the aggregated size of all selected items cannot exceed the capacity of the knapsack. Another constraint is that each item must be allocated only once. The objective is to transport the maximum number of items to maximize profit. The multidimensional knapsack problem arises in several practical contexts such as capital budgeting, cargo loading, cutting stock problems and processors allocation in huge distributed systems.

Definition 2.3. Packing Integer Program [29]. Given $A \in [0,1]^{n \times m}$, $b \in [1,\infty)^n$ and $c \in [0,1]^m$ with $max_j\ c_j = 1$, a packing (resp. covering) integer program PIP (resp. CIP) seeks to maximize (resp. minimize) $c^T.x$ subject to $x \in Z_+^m$ and $A_x \leq b$ (resp. $A_x \geq b$). Furthermore if $A \in \{0,1\}^{n \times m}$, we assume that each entry of b is integral. We also define $B = min_i b_i$.

We can formulate this problem as an integer program. There are n items available for selection each with a profit and a weight. The set of items is noted $I = \{i_1, \ldots, i_n\}$ with profit P_i and weight W_i^I for item i. $K = \{k_1, \ldots, k_m\}$ is a set of knapsack. W_j^K is the capacity of knapsack j. The boolean variable x_{ij} is set to $x_{ij} = 1$ to indicate that item i is packed into knapsack j. The objective is to maximize the profit while respecting the capacity constraints.Table 2.3 provides an example of the Packing Integer Program variables:

TABLE 2.3: Packing Integer Program variables

$x_{i,j}$	Boolean variable indicating whether item i is assigned to knapsack j
P_i	Profit of item i
W_i^I	Weight of item i
W_j^K	Capacity of knapsack j

The Packing Integer problem can be represented as an integer program [30]:

$$max \sum_{j=1}^{m} \sum_{i=1}^{n} P_i x_{i,j} \tag{2.6}$$

subject to:

$$\forall j \sum_{i=1}^{n} W_i^I x_{i,j} \leq W_j^K \tag{2.7}$$

$$\forall i \sum_{j=1}^{m} x_{i,j} \leq 1 \tag{2.8}$$

$$x \in \{0,1\}, \forall i \in T, j \in B$$

Packing Integer Problem is an important class of integer problems because it generalizes the notion of filling in various containers and helps in solving complex problems such as graph isomorphism or hypergraph matching.

2.1.2 Partial Permutations

Multi-dimensionnal Packing can be solved through combinatorial computation. Combinatorial analysis [31] is the mathematical study of permutations and combinations of finite sets of objects. These problems can be modeled through partial permutations [32] [33]. By extension we can solve scheduling problems with permutations [34]. The aim of this subsection is to list the methods for counting sets in various conditions. Many books and articles describe the different combinatorial representations [35] [36] [37] [38]. We focus on Partial Permutation and more specifically on Partial Permutation with repetition that is used to represent the cloud resource allocation problem.

First, we give a definition of Partial Permutation without repetition with proofs. We show that this representation does not perfectly match with our allocation problem.

2.1.2.1 Partial Permutation without repetition

A partial permutation [Def. 2.10] is an arrangement of a number of k items chosen from n distinct items without repetition. For example, the words 'top' and 'pot' represent two different permutations (or arrangements) of the same three letters without repetition.

Definition 2.4. Partial Permutation without repetition. Let $E = \{e_1, e_2, \ldots, e_n\}$ be a set of n distinct items, we call partial permutation without repetition of these n items p to p, any group ordered of p items $(e_{i1}, e_{i2}, \ldots, e_{ip})$ chosen from n items of E without repetition. Let p as $1 \leq p \leq n$. The number of partial permutations on n item p to p without repetition is $\frac{n!}{(n-p)!}$ with $p \leq n$.

For $p > n$ we have $A_n^p = 0$, which represents the pigeonhole principle. This principle states that if n discrete objects occupy p containers, with at least one container hosting

at least $\left\lceil \frac{n}{p} \right\rceil$ objects. The number of partial permutations without repetition is given by:

$$A_n^p = n(n-1)\dots(n-p+1) = \frac{n!}{(n-p)!} \tag{2.9}$$

TABLE 2.4: Example solutions of partial permutation without repetition

$\{vm_1, vm_2\}$	$\{vm_2, vm_3\}$	$\{vm_3, vm_1\}$
$\{vm_2, vm_1\}$	$\{vm_3, vm_2\}$	$\{vm_1, vm_3\}$

For an example with one group $VMS = \{vm_1, vm_2, vm_3\}$, we have $\frac{n!}{(n-p)!} = 6$ partial permutations without repetition. Partial permutations without repetition of two selected items from VMS are shown in Table 2.4. Note that this representation does not correspond to the structure of our resource allocation problem in a cloud platform.

2.1.2.2 Partial Permutation with repetition

A partial permutation [Def. 2.10] is an arrangement of a number of k items chosen from n distinct items with repetition. For example, the words 'top' and 'oot' represent two different permutations (or arrangements) of the same three letters with repetition.

Definition 2.5. Partial Permutation with repetition Let $E = \{e_1, e_2, \dots, e_n\}$ be a set of n distinct items, we call partial permutation with repetition of these n items p to p, any group ordered of p items $(e_{i1}, e_{i2}, \dots, e_{ip})$ chosen from n items of E with repetition. The number of partial permutations on n item p to p with repetition is n^p.

The expression of Partial permutations with repetition is given as:

$$A_n^p = n^p \tag{2.10}$$

Let us consider the example of a resource allocation within a cloud platform where we want to place three virtual machines $VMS = \{vm_1, vm_2, vm_3\}$ on two hypervisors $HYPS = \{hyp_1, hyp_2\}$ in an IaaS platform. We have $2^3 = 8$ partial permutations with repetition.

TABLE 2.5: Example solutions of partial permutation with repetition

$\{hyp_2, hyp_2, hyp_2\}$	$\{hyp_1, hyp_1, hyp_1\}$	$\{hyp_1, hyp_2, hyp_2\}$
$\{hyp_1, hyp_1, hyp_2\}$	$\{hyp_2, hyp_1, hyp_1\}$	$\{hyp_2, hyp_2, hyp_1\}$
$\{hyp_2, hyp_1, hyp_2\}$	$\{hyp_1, hyp_2, hyp_1\}$	

This partial permutation with repetition is a design that best represents our resource allocation problem in a cloud platform.

2.2 cloud computing: Key Performance Indicators

It is difficult to precisely measure the performance of cloud platform components because the key performance indicators (KPIs) are heterogeneous and do not use the same metrics. The KPIs are used to make decisions about moving to the cloud based on the data measurement capabilities (quality of service, availability and reliability) [39]. Obviously, these indicators have an essential role as they enable measurement of the impact of decisions made by IT managers. Key Performance Indicators come up with WHAT the results of the measurement are, but not HOW the measurement was performed. As mentioned, the difficulty lies in the fact that the indicators are heterogeneous with definitions that may vary from one provider to another. As of today, there are several indicators for energy management, formalized and defined differently, [40] [41], QoS [42], costs [43][44] or business [45] [46] .

There is an attempt by NIST to standardize KPI and metrics in cloud environments [47]. NIST defines an abstract measurement standard used to assess resource properties. This abstraction is interesting since it aims at the integration of these metrics in a unified model describing a cloud platform. But this attempt is only a draft at this time.

What are the performance indicators within a cloud environment? We classify these indicators into three major categories:

- Technical indicators (network access, availability of resources);
- Strategic and business indicators;
- Tarification indicators.

2.2.1 Technical indicators

The technical section is the simplest to assess because these indicators are following the traditional IT. For solution providers who offer hosted applications and services, measuring and tracking utilization rates is crucial. We present is this sub-section Infrastructure capacity indicator with energy and bandwidth, availability rates, and convergence and reconfiguration time of scheduling process.

2.2.1.1 Infrastructure Capacity

We can measure the capacity of an infrastructure with hardware capacity and utilization rates. Hardware capacity is measured by storage, CPU cycles, network bandwidth

or workload memory capacity as indicators of performance. Infrastructure utilization is measured by uptime availability and volume of usage as indicators of activity and usability.

Managing capacity ensures that there is always available capacity to meet service demands. Service providers develop horizontal and vertical scalability [48] with business forecast, project pipelines and use data from capacity monitoring to avoid resource shortage. Horizontal scaling consists of adding more machines into your pool of resources whereas "Vertical scaling" implies the addition of more power (CPU, RAM) to your existing machines.

In a cloud world horizontal-scaling is often based on adding new hypervisors, in vertical-scaling increasing the capacity is done through a multi-core process, balancing the load over both the CPU and RAM resources of machines or adding such ability to the original features.

Horizontal-scaling often makes it easier to scale dynamically by adding more machines into the existing pool - Vertical-scaling is often limited to the capacity of a single machine, scaling beyond that capacity often involves downtime and comes with an upper limit.

In short, we can cite some examples of capacity indicators:

- Hypervisors utilization;
- Application utilization;
- Bandwidth utilization between hypervisors;
- Bandwidth utilization between application.

2.2.1.2 Availability Rates

High-availability is the ultimate aim of a cloud platform. Indeed, the idea to be able to access our services, our tools or our data from anywhere at anytime, it is the essence of the cloud computing paradigm. But it is very difficult to define high-availability in a cloud environment. What is the availability? Is it consistent with network devices reliability? Does it correspond to downtime of virtual resources?

In the case of Infrastructure as a Service (IaaS), high-availability is the availability of provider's hardware platform: in other words, we are dealing with failures in the datacenter. But it can also be downtime of virtual resources. Generally speaking, the availability is displayed as a percentage. In its classic form, availability is represented

as a fraction of total time that a service needs to be up. Let us take a 99.99% SLA, for instance. This means that the service can only be offline for about 50 minutes per year.

We can cite some examples of availability indicators:

- reliability rates;
- downtime rates.

2.2.1.3 Convergence Time

A rapid provisioning of resources provides a higher elasticity within cloud platform. This means that customers will have faster access to their services. Resources are scaled up and down to follow business activity as it expands and grows or is redirected. Provisioning time can go from hours to minutes.

When the convergence time of an allocation algorithm exceeds execution time of a reconfiguration plan, then it means that the cloud platform will run in a degraded state where resources will appear as unstable.

2.2.1.4 Reconfiguration Time

Current virtualization techniques can support dynamic reconfigurations of virtual machines. Reconfiguration plans implement the required configurations actions on the selected hosting cloud platforms. It includes the provisioning and instantiation of the proposed solution by the allocation optimization process. The reconfiguration time is the time taken by the reconfiguration plan to execute completely.

This is an important indicator because if the reconfiguration plan takes too much time to perform compared to the feed-back loop, then the cloud platform will be in a state of permanent configuration. This will cause downtime and unwanted side effects.

2.2.2 Strategic and business indicators

Key performance indicators can help the managers to track their business. This is what is called business intelligence. Business intelligence is the ability to slice and dice the information contained within a business into meaningful data. It is a way for entrepreneurs to drill through the organization and get the information that is needed to efficiently manage their business. Business indicators cannot be found that easily due to both infrastructure heterogeneity and the fact that these concepts are subject to

personal interpretation. The business indicators we quote are only samples chosen for their relevance.

2.2.2.1 Security Awareness

If you wish to store information virtually, you must consider the added risk that your information may be accessible to other people who you do not wish to have access. We also need to ask the right questions to find out whether your information is safe.

We can cite some examples of security indicators:

- Risk level shows to what extent intrusion or service halt may happen following an incorrect operation or status vulnerability;
- Incident level shows to what extent intrusion or service halt have already occurred;
- Influence range level shows how much one incident of a component will affect other components over the cloud.

2.2.2.2 Cost Of Goods Sold

Cost of goods sold (COGS) is the consolidated total of all direct costs used to produce and deliver a product or service, which has been charged. In a service business, the cost of goods sold is considered to be the manpower, payroll taxes, and benefits of those people who generate billable hours (though the term may be changed to "cost of services"). A cloud service provider must know what their operational costs are for their physical IT resources, so as to assess the COGS.

2.2.2.3 Return On Investment

Cloud computing is changing the IT purchase from the investment, or capital expenditure (CAPEX) to operating expenses (OPEX). RoI measures or assesses, for a service provider, whether the investment spent in infrastructure does not exceed the benefits (profit) provided by the pricing of services paid by customers.

2.2.2.4 Earnings before interest and taxes

Earnings before interest, taxes, depreciation, and amortization (EBITDA) [49]. To calculate EBITDA, a business must know its income, expenses, interest, taxes, deprecation

(the loss in value of operational assets, such as equipment) and amortization, which is expenses for intangible assets, such as patents, that are spread out over a number of years. The problem is that EBITDA is not a standardized indicator, the content can differ from one company to another.

2.2.3 Tarification indicators

Usually regarded as less obscure than the technical or business indicators, price indicators are in reality equally complex to handle. If we take into consideration all parameters, including storage, virtual machines or network availability, there are thousands of possible prices attribuable onto the use of a cloud platform. Considering offers all different from one provider to another (template servers, size and performance of storage drives, sizing virtual machines), the comparison looks either very difficult or simply impossible. The billing method issue appears to be a crucial point: do customers pay for power consumption per hour, per month or per year?

An IaaS provider can determine the IaaS pricing for their services by first calculating their hourly operational costs per server, per hour and then adding the desired amount of mark-up, to meet organizational revenue targets.

2.3 Problem Resolution and Algorithm Implementation

We list in this section the algorithms we have studied in this thesis and that can be found in popular cloud platforms.

2.3.1 Branch and Bound

The Branch and Bound [50] algorithm is classified as an exact method. This method is one of the most popular methods for the resolution of NP-hard combinatorial optimization problems. They use the divide and conquer strategy by partitioning the search space solutions into sub-problems to process individually. This method is based on a tree search of an optimal solution with separation and evaluation. A sub-problem can also be divided into sub-problems so that this process is similar to the construction of a tree.

Note that the list of all the elements is frequently unrealistic due to the size of search space to be covered.

The Branch and Bound is based on three principles:

- Separation principle
- Evaluation principle
- Path strategy principle

The separation principle is, to split the problem into sub-problems. Each of them presenting a set of feasible solutions. By solving all sub-problems and retaining the best solution we end up solving the initial problem. The principle of separation is applied sequentially or recursively to each subset while there are solutions. The process for separating a branch stops when we know the best solution of all sub-problems or when we know that one sub-problem does not contain any feasible solution.

The evaluation principle relies on the quality of nodes. If a node is considered inappropriate, improved algorithms do not explore, anymore, other solutions associated with this node.

This method was used to solve common small problems and many researchers have improved the algorithm over time. Other algorithms such as the Branch and Cut (Padberg et Rinaldi, 1991) [51], Branch and Price (M. Desrochers, F.M. Solomon. 1989) [52] were proposed. Combining these two techniques leads to the "Branch and Cut and Price" method.

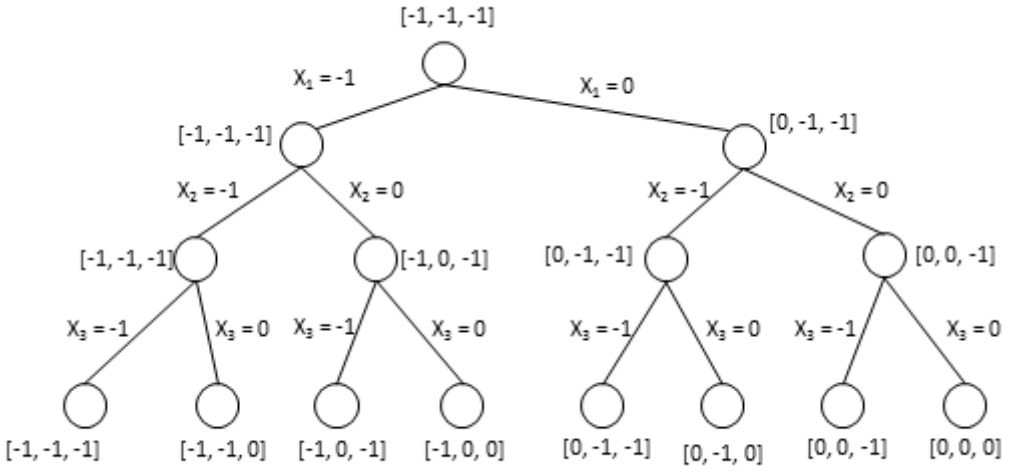

FIGURE 2.1: Branch And Bound Decomposition

Figure 2.1 shows a branch and bound example of the cloud resource allocation resolution with three consumer and two provider resources.

2.3.2 Round-Robin

The Round-Robin algorithm is very interesting as it has given birth to many more sophisticated algorithms . Round-Robin [53] [54] is one of the oldest, simplest, fairest and most widely used scheduling algorithms, especially designed for time-sharing systems. It can be viewed as a load balancing algorithm. This is the simplest algorithm; a request is sent to the server first, then the second and so on and so forth up to the last, and then a round is started with a first query to the first server... It's designed to give a better convergence time but the worst turnaround and waiting time. The basic algorithm uses time quantum to execute a process for a given time. A time quantum [55] is a time slice.

The algorithm can be modified to allocate resources within a cloud platform. The Round-Robin scheduling algorithm for cloud resources is given by following steps:

1. The scheduler maintains a queue of ready virtual machine to allocate and a list of hypervisors hosting resources;
2. It scans hypervisors one by one and allocates resources when possible;
3. Algorithm stops when the queue is empty or when hypervisors are full.

This is an allocation and scheduling algorithm so simple but it can be improved to address workloads or other more complex models [56] [57] [58] [59].

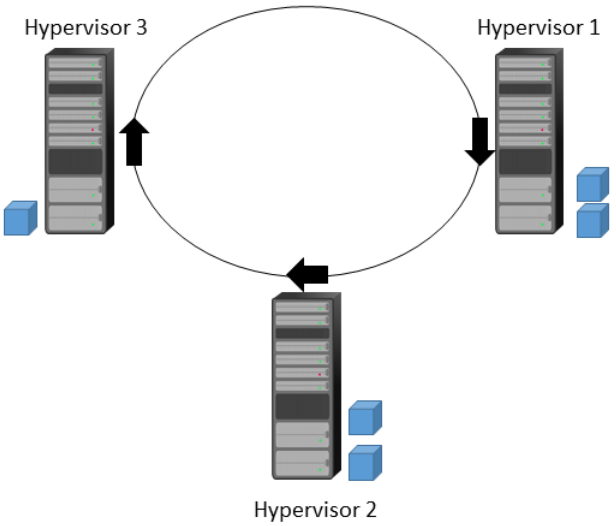

FIGURE 2.2: Round Robin concept

In the context of the resolution of our allocation problem, Figure 2.2 shows an example of the resolution of our algorithm with five customer resources and three provider resources.

2.3.3 MatchMaking

For the scope of our problem, a "rating system" for a resource allocation problem will refer to any method used to assess the containers and assign them a numerical value. Matchmaking system are typically paired with a "rating systems", which will refer to a system that attempts to select high rated containers during the allocation process. A rating system does not necessarily need to be very complex or have a statistical basis.

For example, each hypervisor could simply be assigned a rating equal to their ratio of acceptance/rejection user requests. These simplistic methods, however, might not accurately reflect the true capability of a hypervisor to host resources. Naturally, there are very complex models of resource allocation problems using matchmaking algorithms. Some are decentralized matchmaking algorithms [60] [61] [62].

The general idea of this algorithm is to allocate consumer resources or user requests one by one on infrastructure. Then, algorithm iterates on filters using as parameters the infrastructure resources and user request. Filters are generally associated with one and only one characteristic to be filtered. For example, there are filters managing RAM capacity, other filters managing downtime, etc... Each filter associates scores to containers. The worst containers are removed after the scores are calculated by the filters. Containers with the best scores are selected.

Figure 2.3 shows an example of the resolution with four provider resources.

2.3.4 Constraint Programming

Constraint programming is a powerful concept for solving combinatorial search problems that draws on a wide range of techniques from operations research, algorithms or graph theory. The basic idea in constraint programming [63] is that the user states the constraints and a solver is used to solve them. Constraints are just relations, and a constraint satisfaction problem (CSP) states which relations should hold among the given decision variables. More formally, a constraint satisfaction problem consists of a set of variables, each with some domain of values, and a set of relations on subsets of these variables [64]. The goal in constraint satisfaction problems is to assign values to variables such that all constraints are satisfied.

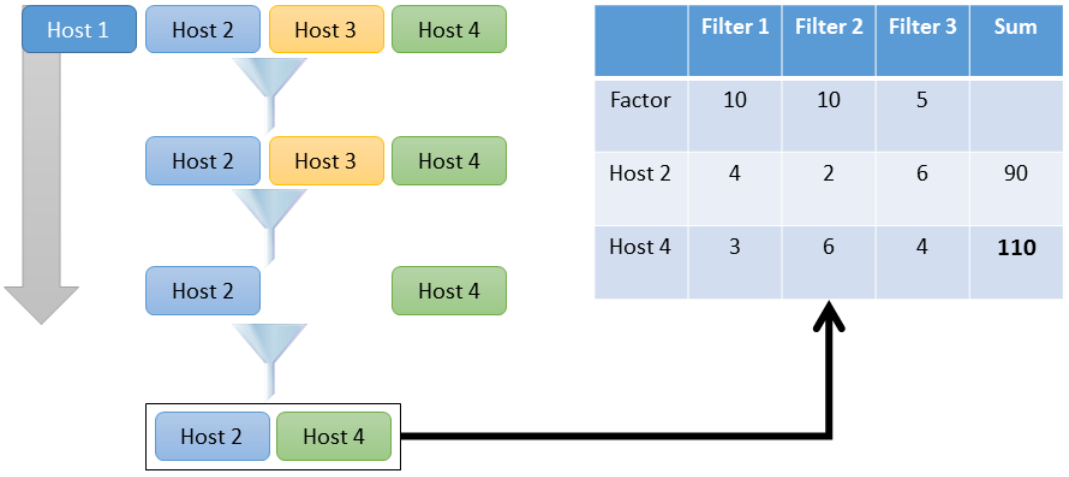

FIGURE 2.3: Filter algorithm concept

We can take as an example, the definition of a basic bin packing problem. Let B a set of bins and I a set of items. Each bin j is associated with the capacity C_j^B and each item i is associated with the capacity C_i^I. In the basic model membership variables are used, that is for each bin j and each item i, a boolean variable $x_{i,j}$ is defined. If the variable is equal to 1 then it means that iteam i is assigned to bin j, else the variable is equal to 0. The variable U_j is a boolean variable stating that the bin j has been used. Table 2.6 below describes variables of Constraint Programming Problem example:

TABLE 2.6: Variables description of Constraint Programming Problem

$x_{i,j}$	Boolean variable indicating whether item i is assigned to bin j
C_i^I	Capacity of item i
C_j^B	Capacity of bin j
U_j	Boolean variable indicating that Bin j is used

The problem can be formulated as follows:

$$min \sum_{j=1}^{m} U_j \tag{2.11}$$

$$\forall j \sum_{i=1}^{n} C_i^I x_{i,j} \leq C_j^B U_j \tag{2.12}$$

$$\forall i \sum_{j=1}^{m} x_{i,j} = 1 \tag{2.13}$$

$$x \in \{0,1\}, U \in \{0,1\}$$

To solve this problem, we can use constraint propagation methods. Constraint propagation is central to the process of solving a constraint problem. Constraint propagation is a very general concept that appears under different names depending on both periods and authors. Among these names, we can find constraint relaxation, filtering algorithms, local consistency enforcing, narrowing algorithms, constraint inference, or rules iteration.

2.3.5 Standard Evolutionary Algorithm

Physical and biological phenomena have been the source and inspiration of many algorithms. Simulated annealing [65] [66] is inspired by thermodynamics, artificial neural networks [67] [68] by the human brain and Evolutionary Algorithms (the most famous are the genetic algorithms) of Darwinian evolution of biological populations [69] [70]. Evolutionary algorithms are mainly stochastic global optimization methods.

Evolutionary algorithms are said to be flexible as they allow to define irregular objective functions [71] [72] within non-standard search spaces [73] [74] such as vectors or d-dimensional graphs.

Genetic Algorithms (GA) were initially developed by Bremermann in 1958 [75] but popularized by Holland who applied GAs to formally study adaptation in nature to apply the mechanisms in computer science [76]. This work led to the development of the principle starting in 1968[76] which was explained in detail in his 1975 book Adaptation in Natural and Artificial Systems [77].

Genetic Algorithms encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes and the values of genes are called alleles. Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to evolve solutions to the search problem using the following steps [78]:

1. **Initialization**. The initial population of candidate solutions is usually generated randomly across the search space. However, domain-specific knowledge or other information can be easily incorporated

2. **Evaluation**. Once the population is initialized or an offspring population is created, the fitness values of the candidate solutions are evaluated.

3. **Selection**. The principle of selection is to choose individuals to evolve the population composed of better individuals. The main idea of selection is to choose better solutions over worse ones, and many selection procedures have been proposed to accomplish this idea, including roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection, some of which are described in the next section.

4. **Crossover**. Crossover combines parts of two or more parental solutions to create new, possibly better solutions. There are many ways of accomplishing this and competent performance depends on a properly designed recombination mechanism.

5. **Mutation**. While crossover operates on two or more parental chromosomes, mutation locally but randomly modifies a solution.

6. **Replacement**. Replace the parent population with new generation.

7. **Repeat steps**. Go through steps 2 to 6 until the termination criteria are met. Evolution stops when the desired level of performance is achieved, or a fixed number of generations has passed without improving the most powerful individual.

The key aspect for the algorithm is the choice of the representation of individuals or the choice of the search space. At each step of the algorithm, there is generally a trade-off between exploring the search space, to avoid getting stuck in a local optimum, and exploit the best individuals obtained in order to produce better populations.

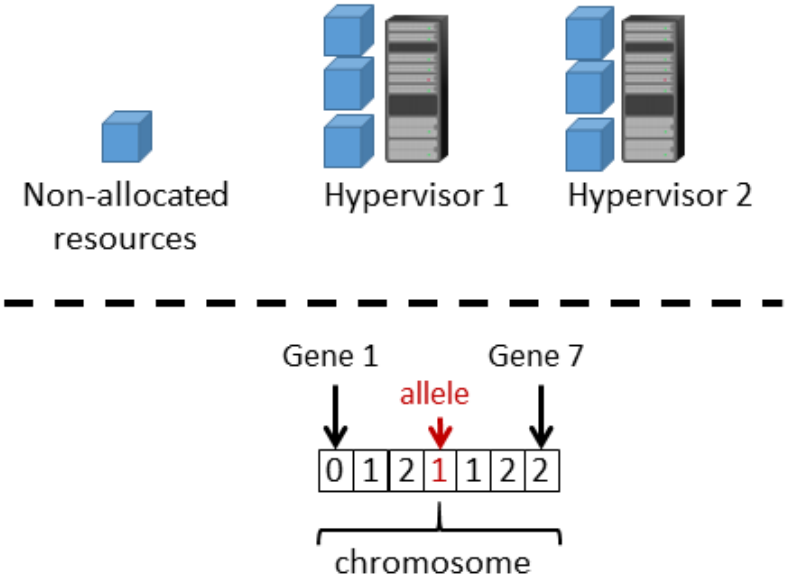

FIGURE 2.4: Allocation Problem Representation with GA

To solve a cloud resource allocation problem, we need to relate resources to the notion of chromosomes [79] [80]. Figure 2.4 illustrates how virtual resources are represented in a chromosome. Genes are the virtual machines to be allocated on infrastructure. Alleles are the hypervisor hosting resources.

It is often difficult to find the parameter values of a genetic algorithm. There are various parameters such as population size, mutation rate, number of generations and many more. For example, if the population is too small, the improvement per iteration in the fitness function will be low (measured as the best candidate solutions or the average of solutions). If you increase the population size and the fitness function increases faster then you have a sign that the result is suboptimal. There is also a point where increasing the population size does not improve the rate of increase in the fitness function. The number of generations is related to improvement in the fitness function. A fitness function usually shows major improvement in early generations and then asymptotically approaches an optimum. The algorithm parameter settings have to be configured correctly to find good solutions quickly. Finding the right age settings is a very complex problem subject of "meta-optimization" [81] [82]. Where the objective is to find suitable parameter settings for a particular evolutionary search algorithm (for a specific problem), and the search is done by another "upper-level evolutionary" process.

2.3.6 Decentralized Scheduling

Usually, industrial and business companies IT infrastructure are hosted in multiple data centers, utilizing state-of-the-art practices for fault tolerance at each level of the system infrastructure, including power, cooling and backbone connectivity. As the infrastructure is composed of several data centers, each data center is responsible for resource allocation in its own realm. But using several algorithms in the different zones leads to inconsistencies in the global resource allocation [83].

To solve this problem and provide global coherence in their infrastructure, businesses decided to search for algorithms that can solve the resource allocation process across different data centers [84]. These algorithms must be coherent and independent from each other [85] [86]. This is what we can call decentralized and distributed algorithms. In decentralized scheduling, organizations maintain (limited) control over their schedules. Jobs are submitted locally, but they can be migrated to another cluster, if the local cluster is overloaded. A distributed system is a collection of independent computers that appear to the users of the system as a single consistent system.

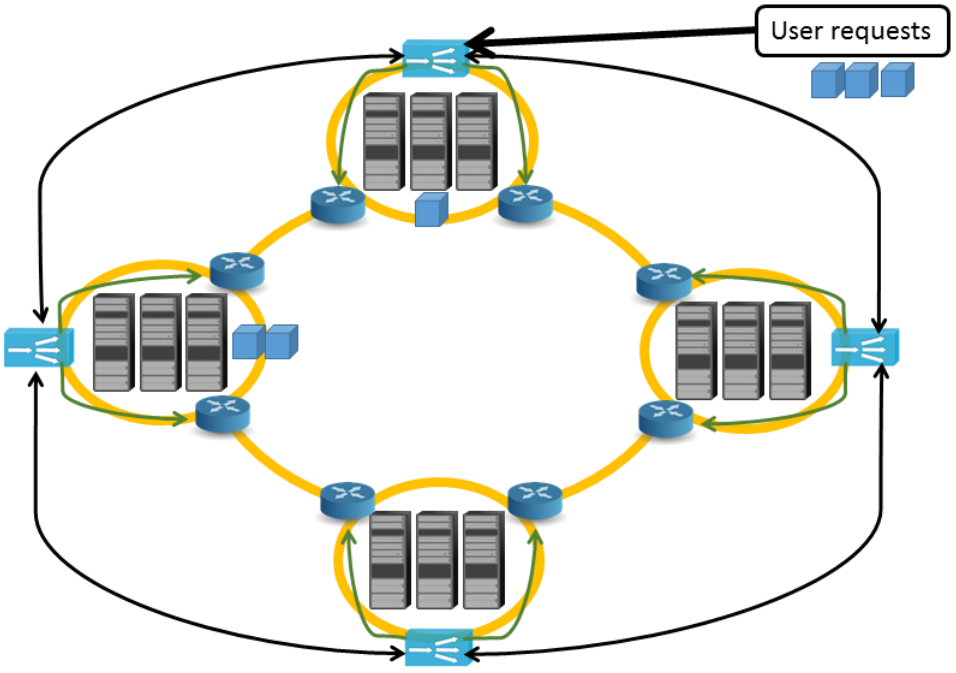

FIGURE 2.5: Distributed algorithm concept

Decentralized and distributed algorithms communicate with each other through protocol such as the peer to peer protocol [87]. Figure 2.5 shows an example with four data centers and one user request composed of three resources. This request is processed by a serving node and resources are allocated in a transparent manner on different data centers.

2.3.7 Summary of Existing Algorithms

In the tables below we summarize the advantages and disadvantages of existing algorithms :

TABLE 2.7: Comparison of allocation algorithms

	Round Robin	Constraint Programming	NSGA-II	Filtering Algorithm	Decentralized Scheduling
Compliance with constraints	✗	✓	✗	✓	✓
Resource Scalability	✗	✗	✓	✗	✓
Integrity of customer requests	✗	✗	✗	✗	✓
Control over infrastructure	✗	✓	✗	✗	✗

2.4 Scheduling in the industry

Resource management and scheduling are very important processes in industry that affect cost, performance and functionality that are the three basic criteria for the evaluation of a system. Resource allocation and scheduling is an essential element of the automation process. Scheduling decides how to allocate system resources such as CPU cycles, memory, secondary storage space, I/O and network bandwidth, between users and tasks.

Cloud resource management requires complex policies and decisions for single or multi-objective optimization. This process can be affected by unpredictable interactions with the environment, system failures, attacks, for example. Cloud service providers are faced with large fluctuating loads that challenges meeting their claim of cloud elasticity. The scheduling process must be able to manage these load fluctuations.

2.4.1 OpenStack scheduler

The main component of OpenStack is Nova. Nova seeks to provide a framework for the large-scale provisioning and management of virtual compute instances. Similar in functionality and scope to Amazon's EC2 service, it allows you to create, manage, and destroy virtual servers based on your own system images through a programmable API.

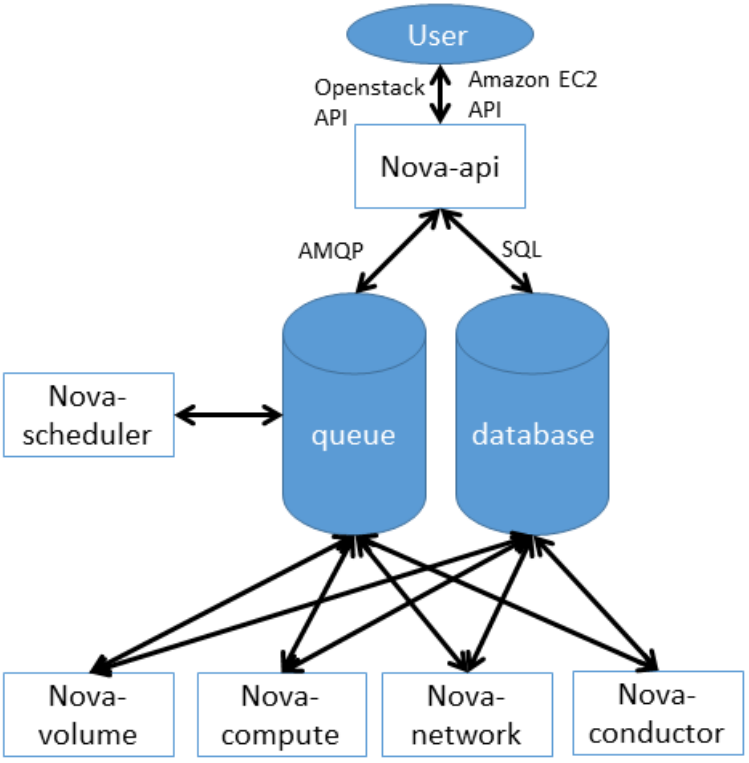

FIGURE 2.6: Nova Logical Architecture

The nova-scheduler [88] process is conceptually the simplest piece of code in Nova: it takes a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on). In practice, however, this will grow to be the most complex component, as it needs to factor in the current state of the entire cloud infrastructure and apply complicated algorithms to ensure efficient usage. To that end, nova-scheduler implements a pluggable architecture [89] that lets you choose (or write) your own algorithm for scheduling.

There are three types of scheduling algorithm implemented in nova-scheduler [90]. Simple algorithm that attempts to find the least loaded host. This is the default scheduler. Chance algorithm that chooses randomly available hosts from a service table. Finally, Zone algorithm that picks randomly hosts from within an availability zone.

Simple and Zone algorithms are equivalent to a matchmaking algorithm. During its work Filter Scheduler iterates over all found compute nodes, evaluating each against a set of filters. The list of resulting hosts is sorted by weighers [Fig. 2.7]. The Scheduler then chooses hosts for the requested number of instances, choosing the most weighted hosts. Filter Scheduler uses the so-called weights during its work. A weigher is a way to select the best suitable host from a group of valid hosts by giving weights to all the hosts in the list. In order to prioritize one weigher over another, all the weighers have to define a multiplier that will be applied before computing the weight for a node. For

a specific filter to succeed for a specific host, the filter matches the user request against the state of the host and other criteria as defined by each filter. Filter Scheduler makes a local list of acceptable hosts by repeated filtering and weighing. We can focus on these

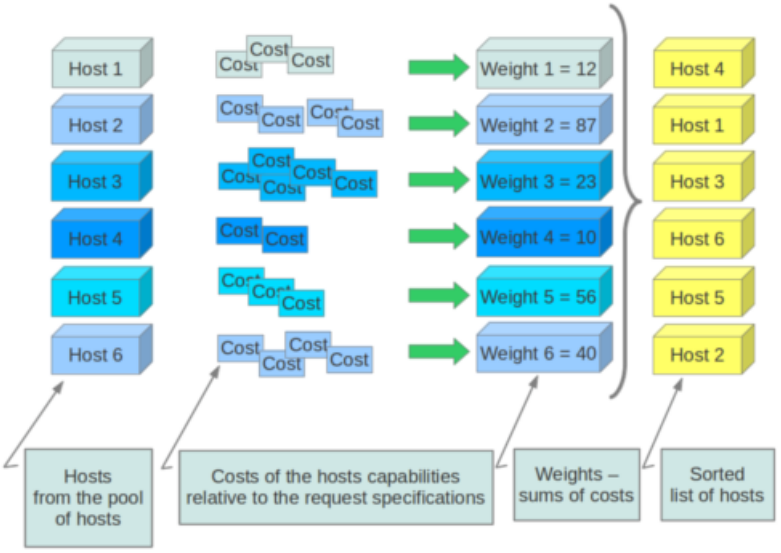

FIGURE 2.7: Filter Scheduler in nova-scheduler

standard filter classes such as AllHostsFilter, CoreFilter and RamFilter because their functionality is relatively simple. But there are many other more complex filters such as subnet mask filter for example.

The list of filters and their explanation is given in the official documentation [90].

2.4.2 Amazon EC2 Scheduling

Scheduling process within Amazon Elastic Compute cloud (Amazon EC2) is very important because it must quickly allocate resources and respond to millions of queries per hour. An EC2 instance is a virtual server in Amazon's Elastic Compute cloud (EC2) for running applications on the Amazon Web Services (AWS) infrastructure. AWS is a comprehensive, evolving cloud computing platform; EC2 is a service that allows business subscribers to run application programs in the computing environment. The EC2 can serve as a practically unlimited set of virtual machines.

Amazon provides a variety of types of instances with different configurations of CPU, memory, storage and networking resources to suit user needs. Each type is also available in two different sizes to address workload requirements. Instance types are grouped into families based on target application profiles. These groups include: general purpose,

compute-optimized, GPU instances, memory optimized, storage optimized and micro instances.

Amazon EC2 is hosted in multiple locations world-wide. These locations are composed of regions and Availability Zones [91]. Each region is a separate geographic area [Fig. 2.8]. Each region has multiple, isolated locations known as Availability Zones. Amazon EC2 provides you the ability to place resources, such as instances, and data in multiple locations. Resources are not replicated across regions unless you do so specifically.

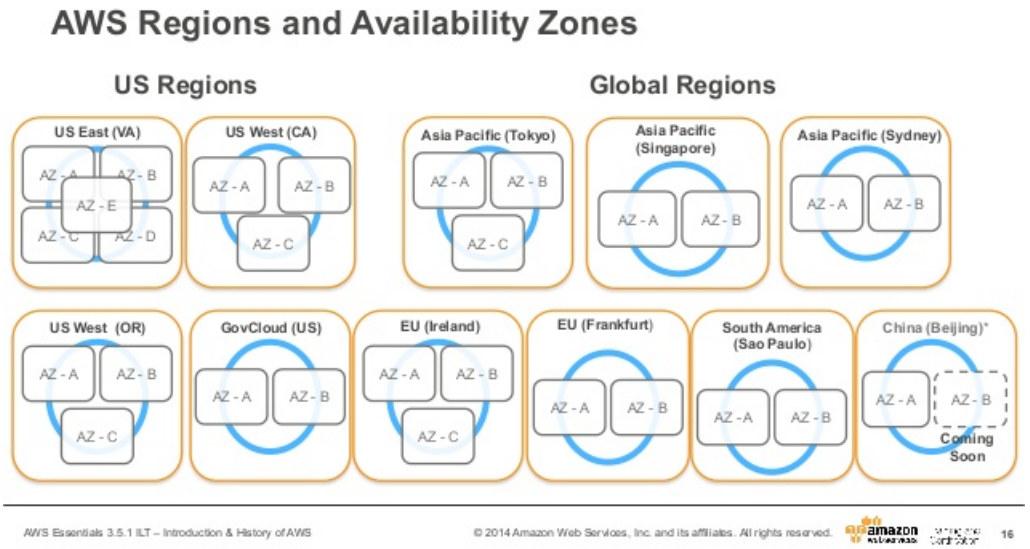

FIGURE 2.8: AWS Regions and Availability zones

A user making a resource request to EC2 chooses their region and their area Availability where they wish to allocate these resources. Then in the Availability Zones, the user request should be allocated among the thousands of servers that host user resources. It is very difficult to know how the user requests are provisioned within an Availability Zones. These mechanisms are not made public as the cost of a VM or the scheduling algorithm. However, we obtained indications by viewing keynotes [92] and participating in sessions AWS re: Invent [93].

Through the descriptions provided in the AWS keynotes, the input parameters requested from users and the response time to queries in relation to the size of the Regions and Availability zones, we can suspect that Amazon EC2 has set up a scheduling process using a matchmaking algorithm.

2.4.3 VMware, Distributed Resource Scheduler

VMware is based in Palo Alto, California and was founded in 1998. The company is a producer of virtualization software and cloud services [94]. The company has the

notable accomplishment of having produced the first x86 architecture virtualization that was workable and suitable for consumer use. VMware virtualization is based on the ESX/ESXi bare metal hypervisor, supporting virtual machines. ESXi (Elastic sky X Integrated) is also the VMware's enterprise server virtualization platform. The term "VMware" is often used in reference to specific VMware Inc. products such as vSphere and VMware Workstation.

To allocate user resources to infrastructure, system administrators can manually allocate the resources on hosts. But system admins cannot predict the future workloads nor can they keep their eyes tracking ups, downs and imbalanced distribution of loads in a virtual infrastructure. Dynamic Resource Scheduler (DRS) [95] is the VMware solution to this problem that aims to give every virtual machine the resources it needs, and not necessarily to make the utilization equal on all hosts in the Dynamic Resource Scheduler cluster.

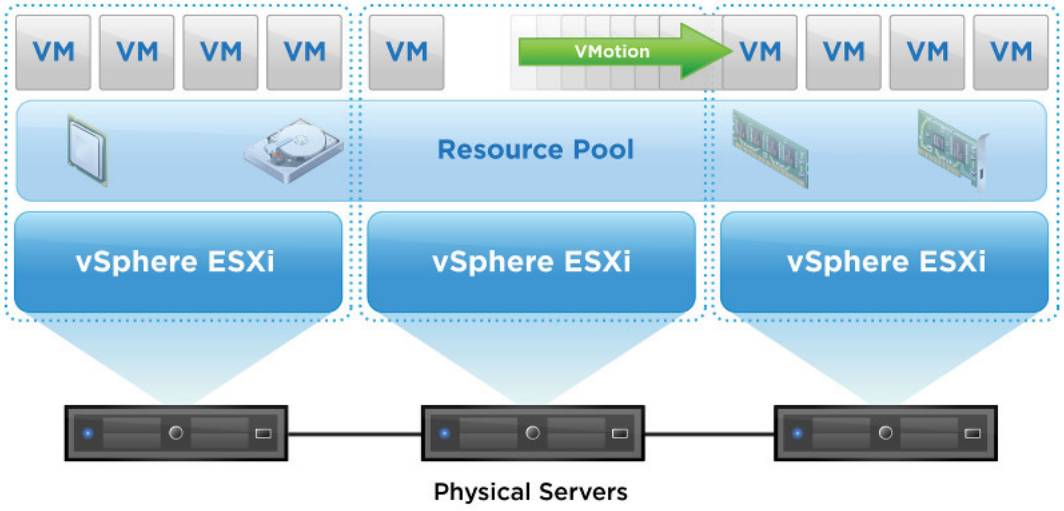

FIGURE 2.9: vSphere Dynamic Resource Scheduler principle

What we are concerned with is the scheduling algorithm. Does DRS use a load balancing or filter algorithm? DRS maps VMs to hosts and performs intelligent load balancing in order to improve performance and to enforce both user-specified policies and system-level constraints. It is easier to figure out the type of allocation algorithm used in DRS compared with Amazon EC2. Paper [95] describes the scheduling process with a filtering algorithm to achieve load balancing between hypervisors. Finally, VMware DRS uses a filter algorithm as Openstack or Amazon EC2.

Chapter 3

Mathematical model for Cloud Placement Problem

Abstract. This chapter presents single and multiple objectives, criteria and constraints algorithm for cloud resource allocation taking into account both the tenant (customer) and infrastructure provider interests. Current cloud placement algorithms are provider centric or biased, limit the optimization scope and force customers to accept the providers' business models. We propose a matrix-based model to represent both the customer needs and provider infrastructure to come up with a problem formulation that leads to joint customer and provider based optimization.

Our goal is to propose a sufficiently generic model for the cloud resource allocation problem even if the analysis and performance evaluation will be focused on the provisioning of IaaS services. The selection of adequate hypervisors will be emphasized with respect to the service request, the compute resources (virtual machines), storage and networking services.

3.1 Cloud Resource Mapping Problem

The approach adopted to ensure joint customer and provider centric resource allocation is to use matrices to describe the service requests and service offers in a form compatible with evolutionary algorithms known to be more appropriate for multi-objectives and multi-constraints optimization problems.

Figure C.3 depicts the modules that compose a typical cloud resource allocation system.

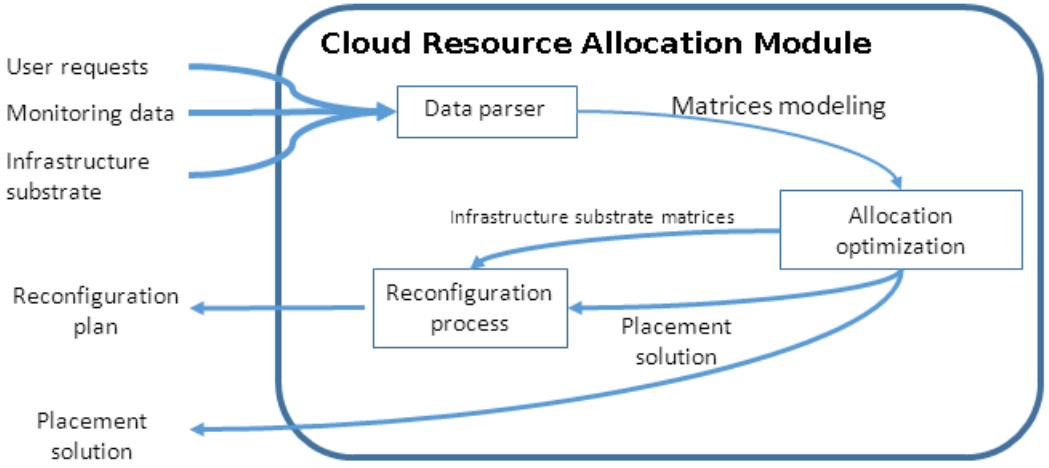

FIGURE 3.1: Cloud Resource Allocation process

- **System input** that consists of user requests for infrastructure resources and collected monitoring data key performance indicators and metrics and logs necessary for describing completely the infrastructure or substrate state, using the proposed matrix-based representation;
- **Data parser** that analyzes and transforms requests from users, such as required resources and relationships, into the matrices that will be used by the model to represent the demand and the available resources from the providers' infrastructures;

- **Output Matrices** composed of a set, of integers and real matrices, that describes the request and infrastructure resources, their relationships and their associated constraints;
- **Allocation optimization** that find solutions for the resource allocation problem using exact or heuristic algorithms;
- **Placement solution** is a feasible solution for the resource allocation problem that fulfills the multiple objectives and respects the resources relationships and constraints;
- **Reconfiguration plan** that implements the required configurations actions on the selected hosting cloud platforms. It includes the provisioning and instantiation of the proposed solution by the allocation optimization process or block.

In this thesis we operate on both the customer and the provider layers and describe the resources in each layer in matrix form that can be easily used by evolutionary algorithms. The user service requests are also broken down into matrices and stay in line with a matrix-based model for the cloud resource allocation problem.

Most operating networks today are based on the so-called 3-Tier model where the data traffic is assumed to follow a north to south path and meant to leave the data center. For some applications that require est-west communications within the data center or between data centers, the 3-Tier model is not appropriate since it induces many hops. Consequently, other architectures have been proposed to balance the number of hops or leaps and achieve homogeneous latency across all servers [96].

Advanced data centers use an architecture that is far more suitable for managing both redundancy [97] and bandwidth [98] known as the Core/Leaf-Spine distributed network architecture [99] as described in Figure 3.2.

Data centers, when hosting cloud computing platforms, tend to opt for this type of architecture with stronger resilience through three access levels:

- **Spine layer** acting a datacenter distribution system. As suggested by its name, a spine layer serves as backbone to an infrastructure. Each node is connected to each switch on the Leaf layer;
- **Leaf layer** represents the layer that manages access to different hypervisors and connections to the distribution network. This way, each switch is connected to each node on the distribution spine layer, but each resource from Spine and Leaf layers are not directly connected to each other;

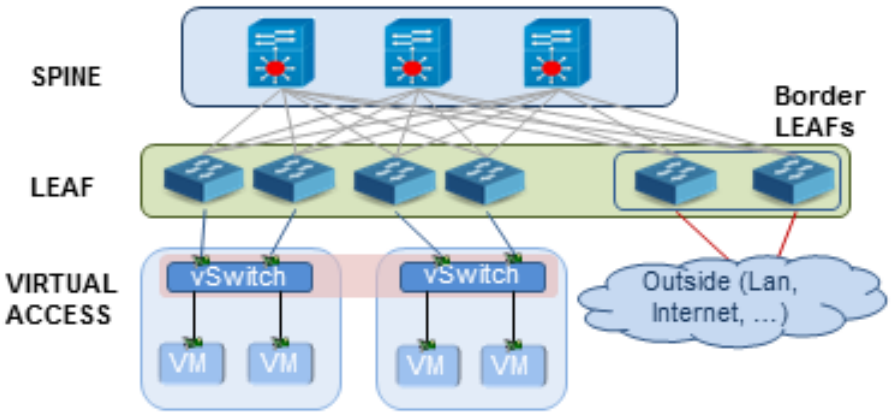

FIGURE 3.2: Spine and Leaf Modern Architecture of datacenter

- **Virtual access layer** encompasses hypervisors and virtual resources for computing or networking.

Note that the Core/Leaf-Spine architecture avoids the use of the Spanning Tree Protocle(STP) [100] and facilitates the addition of extra hardware peripherals and enhances system capacity.

Datacenters provide ever-upgrading architectures in the face of new challenges to take [101] such as mobile datacenters. Mobile datacenters are designed to be deviced to a mobile environment, like a shipping cargo container [102][103] or a sea freighter [104].

As stated previously, a cloud computing platform offers services to customers. This platform is installed on a infrastruture managed by a provider. This service provider strives to meet the requirements from the users' perpesctive while ensuring the overall sustainability of the provided services. Thanks to virtualization, a single server can simultaneously run multiple virtual components, each embedded in a virtual machine, if their total demand in each resource does not exceed the server capacity.

Figure 3.3 shows the composition of an IaaS-type cloud computing system with its different layers. An IaaS platform consists of a layer of hardware resource and another one of virtual resource. The hardware layer consists of servers and switches connected together using the leaf-spine architecture in most cutting-edge datacenters. The virtual layer is made up of virtual machines, virtual switches and networks most often in the form of VLAN. These virtual resources are assigned on hypervisors from the hardware layer. Virtual switches are allocated through calculation nodes and are set to accommodate different VLAN users. You can define connections between resources like affinity or anti-affinity constraints while allocating.

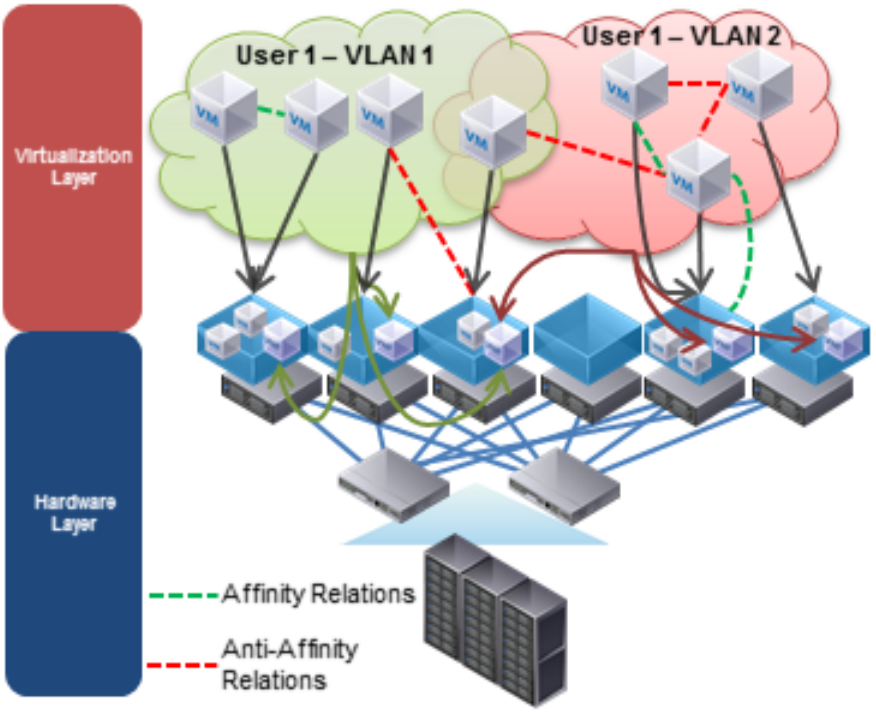

FIGURE 3.3: Virtual resource allocation through Cloud Computing layers

This new modeling with business relations allows implementation of services while keeping an overview of the system and this leads to easier end-to-end automation of failure-free services production.

Our goal is to reduce operating costs for an IaaS platform. Matrices are used here to represent and manage these multiple constraints and objectives. The use of a matrix representation to define the operating costs, performance objectives and constraints all lead us naturally to the use of a constraint solver in order to settle our general resource allocation issue within a cloud platform and to validate our model.

Therefore we offer a method that reflects the requests from customers and we describe a provider's infrastructure shown in the form of matrices in order to solve the cloud resource allocation issues.

Our algorithm solves the cloud resource allocation problem using the users' and providers' layers with the objective of placing (optimally or near optimally) the demands to minimize cost, service unavailability and the need to migrate or move resources in the infrastructures. Without loss of generality we assign equal weights to these objectives, that can otherwise be tuned and configured differently by the stakeholders. The objectives are specified as follows:

- **Operating cost** determines the full cost of an infrastructure integrating operating (consumption) cost of customer resources and the operating cost for the provider;

- **Downtime cost** represents the cost due to the unavailability of customer resources;
- **Migration cost** represents the cost of migrating customer resources using the reconfiguration plan.

We identify five major relationships and constraints to describe the user affinity/anti-affinity requirements between resources expressed by the users and providers:

- **Co-localization in same datacenter** tenants and applications impose the co-location in the same data center of their virtual resources;
- **Co-localization on same server** this constraint imposes the location of the customer or tenant virtual resources in the same server or host;
- **Total separation** customer resources have to be separated and mapped onto distinct servers and distinct datacenters for security reasons for example or simply due to tenant requirements or application constraints;
- **Separation on different datacenters**: this is a separation of customer resources onto different datacenters;
- **Separation on different servers** the customer resources need to be separated into different servers.

There is also one last constraint, the resource capacity constraint related to the maximum amount of available resources from the providers.

- **Resource capacity constraint** this is a limit used to make sure that the total amount of allocated resources does not exceed available maximum capacity

Figure 3.4 describes the resource attributes to a datacenter. Attributes to servers and network liaisons and switches come to be detailed here. This way, modeling a server will be expressed in CPU number and capacity, RAM and HDD capacities, and NIC maximum bandwith. The server will be linked to specific operating costs. Network liaisons are described through bandwidth and flow rate. Switches feature attributes like RAM capacity, switching bandwith and forwarding rate.

Figure 3.5 shows resource attributes to a user request. A request is a user query involving virtual machines, virtual switches and interrelations between resources. Attributes to a virtual machine and a server are alike. Relations between resources mean affinity/anti-affinity constraints or a described VLAN-like newtork. A virtual switch takes the same attributes as a physical switch do.

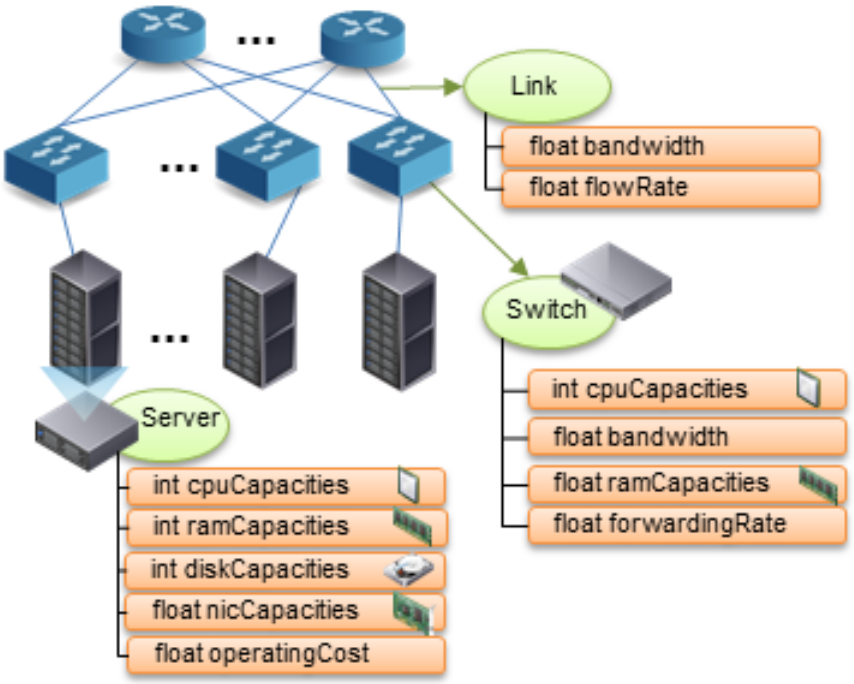

FIGURE 3.4: Infrastructure Resources Attributes

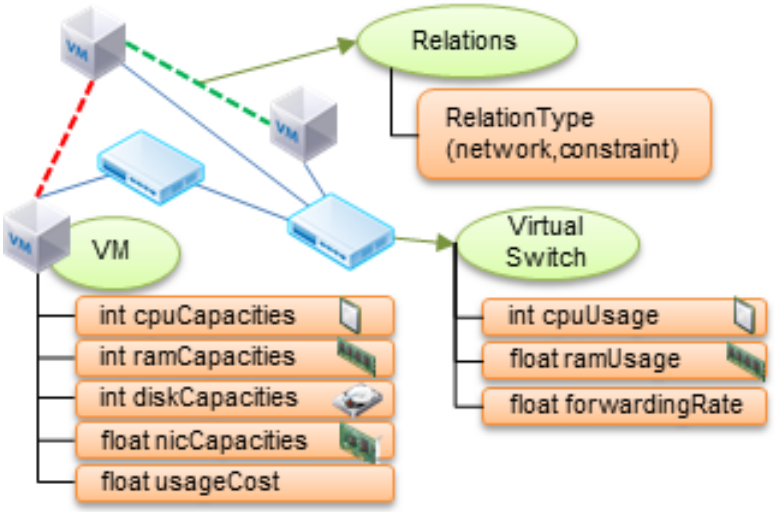

FIGURE 3.5: Virtual Resources Attributes

Our study mainly focuses on virtual machine allocation onto servers in order to compare performance benchmarks from various placement algorithms (tabu search, constraint programming, evolutionary algorithms)

The modeling of the user resource requests and the provider physical infrastructure through matrices is meant to be generic enough to be reused and generalized to other types of services.

3.2 Single Objective Definition

The number of provider datacenters is noted g, the number of servers defined as m, total requested resources n and the number of attributes is represented by h. In our model, the number of provider's resource attributes and the number of customer's resource attributes are the same. The mapping is on identical attributes with respect to the demand. The resource capacities (maximum available resource per node or link) are represented also by two matrices. The first matrix [Eq. 3.1] represents the capacity of each attribute of the provider resources.

$$\begin{array}{cc} P_{jl} & 1 \leq j \leq m, 1 \leq l \leq h \\ P_{jl} \in \mathbb{R}^{+} & m \times h \end{array} \tag{3.1}$$

The second matrix [Eq.3.2] represents the capacity limit of each customer resources attribute.

$$\begin{array}{cc} C_{kl} & 1 \leq k \leq n, 1 \leq l \leq h' \\ C_{kl} \in \mathbb{R}^{+} & n \times h' \end{array} \tag{3.2}$$

The number of attributes for provider and customer resources should be the same for both types of resources ($h = h'$). We also define a factor on the attributes that corresponds to the ratio between the consumed physical capacities of an attribute and the virtual capacities. This factor is also described using the matrix of [Eq.3.3] of size $m \times h$.

$$\begin{array}{cc} F_{jl} & 1 \leq j \leq m, 1 \leq l \leq h \\ F_{jl} \in \mathbb{R}^{+} & m \times h \end{array} \tag{3.3}$$

We note X_{ijk}, a boolean variable, which is true when a customer resource k is hosted by provider server j in datacenter i. We define a constraint on the capacity limit on provider resources not to be exceed in [Eq.3.4]. This constraint includes the virtual to

physical resource consumption factor for each requested attribute.

$$\sum_{k=1}^{n} C_{kl} \times X_{ijk} \leq P_{jl} F_{jl}, \forall l = 1...h, \forall i = 1...g \tag{3.4}$$

To represent the operation expenditure of each provider resource, we introduce the vector [Eq.3.5] whose elements are costs representing the direct costs associated to the use of a server: power, floor space, storage, and IT operations for resource management.

$$E_j \quad 1 \leq j \leq m, \quad E_j \in \mathbb{R}^+ \tag{3.5}$$

In addition to operating expenditure, provider resources have an operating cost for each customer resource they host. We use a vector [Eq. 3.6] for these operating costs.

$$U_j \quad 1 \leq j \leq m, \quad U_j \in \mathbb{R}^+ \tag{3.6}$$

Each attribute on each provider resource has a maximum load before quality of service deterioration or degradation occurs for the hosted virtual resources. We note L_{jl}^M the maximum loading before degradation and L_{jl} the current load on server j for an attribute l. These servers have a current and a maximum achievable quality of service Q_{jl} and Q_{jl}^M respectively. This level of service guarantee is noted C_k^Q and if it is not respected the provider pays a downtime penalty defined as a cost C_k^U. The service provider must guarantee that there will be no interruption in service. We define these load and quality of service matrices in [Eq.3.7]:

$$\begin{array}{ll} L_{jl}^M & 0 \leq L_{jl}^M < 1 \\ L_{jl} & 0 \leq L_{jl} < 1 \\ Q_{jl}^M & 0 \leq Q_{jl}^M < 1 \\ Q_{jl} & 0 \leq Q_{jl} < 1 \end{array} \tag{3.7}$$

All the variables and constants used in our model are listed for easy reference in Table 3.1.

The model is augmented with all the required relationships and constraints expressed in the user requests and the valid equalities and inequalities relative to the demand and hosting infrastructures.

The **co-localization on same datacenter** constraint expressed in [Eq.3.8] ensures that resources subject to this constraint are hosted in the same data center:

$$\sum_{i=1}^{g} \prod_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{3.8}$$

TABLE 3.1: Cloud resources allocation model variables

	Provider substrate and requested resources
G	Set of available datacenters $G = \{1, ..., g\}$
M	Set of available servers $M = \{1, ..., m\}$
N	Set of requested resources $N = \{1, ..., n\}$
H	Set of available attributes for each resource $H = \{1, ..., h\}$
	Capacities matrices
P_{jl}	Capacity of provider resource j for attribute l
C_{kl}	Capacity of requested resource k for attribute l
F_{jl}	Capacity factor of attribute l on provider resource j
	Mapping model
X_{ijk}	Boolean variable indicating whether requested resource k is assigned to provider resource j in datacenter i
	Quality of service matrices and vectors
L_{jl}	Load of attribute l on provider resource j
L_{jl}^{M}	Maximum load of attribute l on provider resource j
Q_{jl}^{M}	Maximum quality of service for attribute l on provider resource j
Q_{jl}	Quality of service for attribute l on provider resource j
C_{k}^{Q}	Quality of service guaranteed by the provider on customer resource k
	Costs vectors
E_j	Operational expense for a provider resource j
U_j	Usage cost for each customer resources hosted on a provider resource j
C_{k}^{U}	Cost of downtime for a resource k
M_k	Migration cost of a customer resource k

The **co-localization on same server** constraint is achieved if the sum of the products on the customer allocated resources meets [Eq.3.9]. This guarantees co-localization in the same server:

$$\sum_{i=1}^{g}\sum_{j=1}^{m}\prod_{k=1}^{n} X_{ijk} = 1 \tag{3.9}$$

The **total separation** constraint is expressed via the sum of resources, allocated to all datacenters and servers, that has to be equal to one as indicated in [Eq. 3.10]:

$$\sum_{k=1}^{n} X_{ijk} = 1, \forall i = 1...g, \forall j = 1...m \tag{3.10}$$

The **separation on different datacenters** constraint is substantially identical to the previous one, except that the sum on the datacenters and requested resources as well has to meet [Eq. 3.11]:

$$\sum_{i=1}^{g}\sum_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{3.11}$$

The **separation on different servers** constraint involves the sum of the datacenters, servers and requested resources as expressed in [Eq. 3.12].

$$\sum_{i=1}^{g}\sum_{j=1}^{m}\sum_{k=1}^{n} X_{ijk} = 1 \tag{3.12}$$

All these relationships and constraints are implemented in our linear programming model that will be used to provide solutions to the cloud resource allocation problem addressed in this work.

3.3 Allocation problem solving

We solve the resource allocation problem according to the operating costs, the affinity/anti-affinity relations between resources and the required quality of service on the assigned cloud computing resources. The approach takes into account multiple constraints, relationships and objectives but the multiple objectives are aggregated in the overall objective function as described in the sequel. Since our optimization problem is similar to the multidimensional bin packing and the knapsack problems that have been shown to be NP-Hard, our problem is also NP-Hard. The multidimensional bin packing problem is a vector scheduling problem with d dimensions proven to be NP-Hard in [105]. As our objective is to minimize costs, service interruptions and reconfiguration plan sizes, we introduce several objective functions to compute namely, the costs in [Eq. 3.13], the quality of services of resources through [Eq. 3.14] and the approximate evaluation of the reconfiguration plan sizes using [Eq.3.15]. Our resource allocation problem can be summarized by lumping all the objective functions with all the constraints and conditions into the following set of equations:

$$min\ Z = min[\sum_{j=1}^{m} E_j \times X_{ijk} + \sum_{k=1}^{n} U_j \times X_{ijk}, \forall i=1...g \tag{3.13}$$

$$+ \sum_{k=1}^{n} C_k^U(\lfloor \frac{Q_{jl}}{C_k^Q} \rfloor) \times X_{ijk}, \substack{\forall i=1...g \\ \forall j=1...m \\ \forall l=1...h} \tag{3.14}$$

$$+ \sum_{k=1}^{n} (X_{ijk}^t - X_{ijk}^{t+1}) M_k, \substack{\forall i=1...g \\ \forall j=1...m}] \tag{3.15}$$

Subject To according to the request:

$$\sum_{k=1}^{n} C_{kl} \times X_{ijk} \leq P_{jl} F_{jl}, \forall l = 1...h, \forall i = 1...g \tag{3.16}$$

$$\sum_{i=1}^{g}\prod_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{3.17}$$

$$\sum_{i=1}^{g}\sum_{j=1}^{m}\prod_{k=1}^{n} X_{ijk} = 1 \tag{3.18}$$

$$\sum_{k=1}^{n} X_{ijk} = 1, \forall i = 1...g, \forall j = 1...m \tag{3.19}$$

$$\sum_{i=1}^{g}\sum_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{3.20}$$

$$\sum_{i=1}^{g}\sum_{j=1}^{m}\sum_{k=1}^{n} X_{ijk} = 1 \tag{3.21}$$

Clearly, we could solve the problem as an integer linear programming using the model proposed to minimize the costs in equations 3.13 to 3.15 subject to the constraints specified in equations 3.16 to 3.21. But we favor the use of heuristic and evolutionary algorithms since the ILP solutions will not scale with problem size.

The objective functions and the metrics are all converted to an equivalent monetary cost so they can be aggregated into a global objective function. We detail all objective functions. We aggregate all objective functions by converting these different values in pecuniary cost. The first function is the full cost calculation given by [Eq.3.13]. This function comprises operating costs. For each provider resource, E is the operating cost of provider resource j. For each customer resources, variable U represents the customer resource custompton cost k.

Second objective [Eq.3.14] represents the costs of temporary service disruptions or penalty for lower quality of service experienced in hosting resources from the providers. On the basis of performance assessments, authors [106] [107] prove with empirical investigations that the quality of service of customer resources decreases exponentially with increasing workload in provider resources. The total dowtime cost is the sum of each downtime cost C_k^U on customer resources k when the quality of service guarantee C_k^Q is not respected.

We need to calculate the quality of service [Eq. 3.22] for each attribute l on provider resource j. We note this variable as Q_{jl}. The quality of service as a function of load

behaves as a piecewise function where each provider resource has a maximum load before deterioration L_{jl}^{M}. This is reflected in [Eq.3.22]:

$$Q_{jl} = \begin{cases} Q_{jl}^{M} & if\ L_{jl} \leq L_{jl}^{M} \\ Q_{jl}^{M} e^{\frac{L_{jl}^{M} - L_{jl}}{1 - L_{jl}}} & if\ L_{jl} > L_{jl}^{M} \end{cases} \tag{3.22}$$

The workload on provider resource j for a specific attribute l is calculated [Eq.C.17] as the sum of the values of the attributes of the resources located on the servers divided by the provider resource capabilities.

$$L_{jl} = \frac{\sum_{k=1}^{n} C_{kl} \times X_{ijk}}{P_{jl}} \begin{matrix} \forall i=1...g \\ \forall j=1...m \\ \forall l=1...h \end{matrix} \tag{3.23}$$

The last objective is the sum of the costs of implementing the reconfiguration plan [Eq.3.15]. The size of reconfiguration plan is an estimate based on the current allocation X_{ijk}^{t} and the next allocation X_{ijk}^{t+1} provided by the optimization process.

3.4 Multi Objectives Definition

Multi-objectives problems have the characteristic of being much more difficult to process than their single-objective equivalent. Thus, we initially described our problem in the form of a single-objective problem using a scalar approach where we used an aggregation of goals with equal weight. The multi-objectives definition challenge lies in the lack of ordered relations between the solutions. One solution may be better than another on certain objectives and worse on others. So there is generally no one solution that simultaneously provides the optimum solution for all objectives. That is why the concept of optimal solution becomes less relevant in multi-objective optimization. In this case the optimal or good solution is not a single solution, but a set of compromise solutions between the different objectives to optimize. It is vital to identify the best compromise to define an ordered relationship between these elements. The most famous and most used is the dominance relation in the Pareto sense. All the best compromise is called the Pareto front, the surface of compromise or all effective solutions. This set of solutions is a balance because no improvement can be made on an objective without degradation of at least one other objective. Pareto solution is to get the Pareto front PF or approximate the Pareto frontier $PF*$.

We convert our current single objective genetic algorithm in multi-objective optimization problem [Eq. 3.24]. This consists in writing a vector containing the previous objectives.

$$minZ = \begin{pmatrix} \sum_{j=1}^{m} E_j \times X_{ijk} + \sum_{k=1}^{n} U_j \times X_{ijk}, \forall i=1...g \\ \sum_{k=1}^{n} C_k^U (\lfloor \frac{Q_{jl}}{C_k^Q} \rfloor) \times X_{ijk}, \substack{\forall i=1...g \\ \forall j=1...m \\ \forall l=1...h} \\ \sum_{k=1}^{n} (X_{ijk}^t - X_{ijk}^{t+1}) M_k, \substack{\forall i=1...g \\ \forall j=1...m} \end{pmatrix} \tag{3.24}$$

where the first row of the vector represents the calculation of costs [Eq. 3.13], second row, the quality of services of resources [Eq. 3.14] and the last line is approximate evaluation of the reconfiguration plan size [Eq. 3.15].

In the context of this thesis we only talk about the search for best compromise solutions. To select the best compromise on the Pareto front, we use the Goldberg NSGA Ranking.

Chapter 4

Cloud Resource Allocation Algorithm

Abstract. This chapter presents our proposed cloud (IaaS) resource allocation algorithm to address jointly the requirements and interests of consumers and providers. We first describe the graph representing the user request and the provider infrastructure and decompose in a second stage the graph into the matrices proposed in the previous chapter. The chapter describes finally our genetic algorithm (based on NSGA III) and the associated operators and reports the results of performance evaluation and comparison with state of the art algorithms such as Round Robin and Branch-and-Bound.

This chapter presents the initial problem model in the form of a user-modifiable graph used as a starting point by our genetic algorithm based on NSGA II. The initialization operator proposed to improve convergence of the population towards a viable solution is also presented. The chapter describes the used Crossover and mutation operators, specifically simulated binary crossover (SBX) and polynomial mutation (PM), and their generic operation. The performance of the proposed genetic algorithm is compared with state of the art resource allocation algorithms.

This chapter is divided into three sections. Section 4.1 describes how a graph is broken down into the modeling matrices as explained in Chapter 3. The implementation of the NSGA II and NSGA III genetic algorithms is explained in Section 4.2. Section 4.3 provides an assessment of performance of selected algorithms to be compared with our NSGA III algorithm enhanced with a tabu search. The performance evaluation of each algorithm is reported on a per section basis for the Round Robin, the Constraint Programming, NSGA II, NSGA III, NSGA III with constraint solver and NSGA III enhanced with a tabu search (our advocated and retained solution and also one of our main contributions). The last section conducts the actual comparison of the algorithms, readers familiar with the basic algorithms can skip the first sections and proceed directly to Section 4.3.4 for the comprehensive performance comparison between our enhanced NSGA III algorithm and the other algorithms.

4.1 Graph Decomposition

To describe the consumer requests and the provider infrastructure resources we transform the initial description of the virtual and physical resources as depicted in Figure 4.1. This graph provides the connectivity and the topologies of the requests and infrastructure graphs. The graph representation specifies three types of relationships between resources: standard connectivity between nodes, affinity and anti-affinity relationships between nodes. This graph representation (right side of Figure 4.1) is in turn converted into adjacency matrices as shown in the matrices example below that reflect the connectivity and topology of the example depicted previously in 4.1. The algorithms use the matrix representations as input to find a viable resource allocation solution to host the requests in the provider infrastructure.

To facilitate to the users the specification of their requests, we designed a graphical user and administrator interface as shown in Figure 4.2. The interface uses the Neo4j NoSQL (Not only SQL) data base to visualize the infrastructure and hosted resources graphs and charts comparing quality of a new resource allocation with previous allocations.

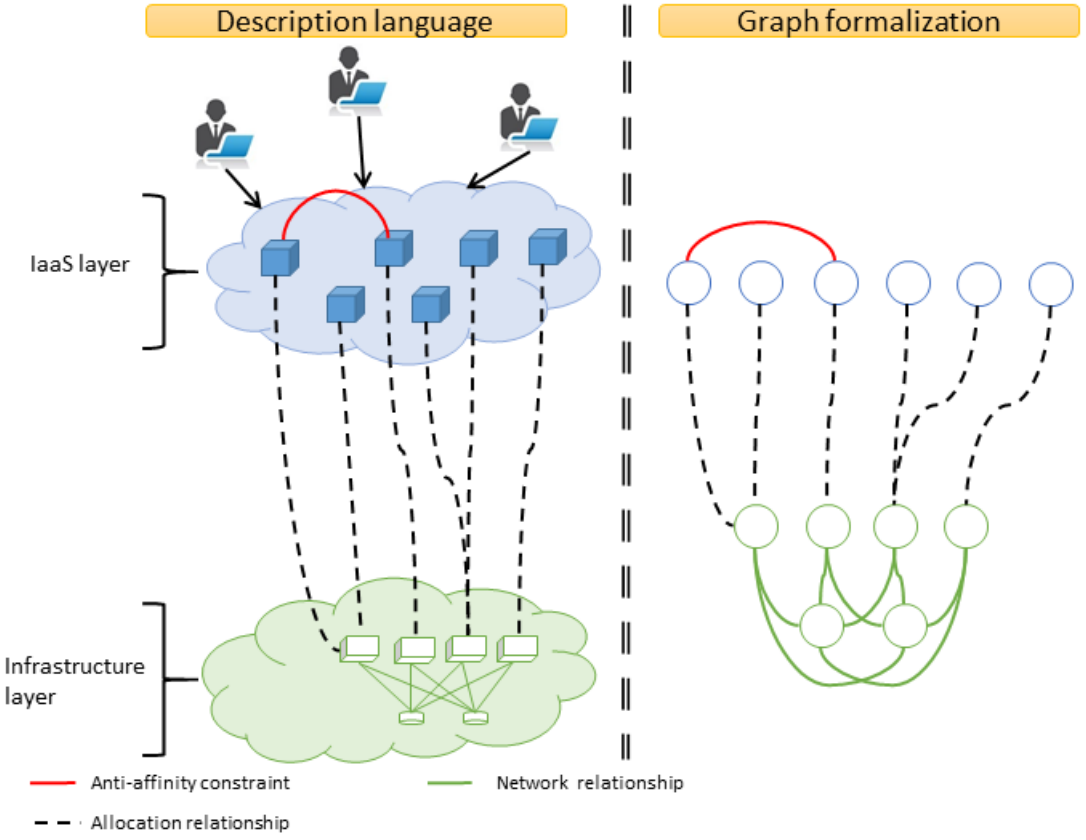

FIGURE 4.1: Graph representation example

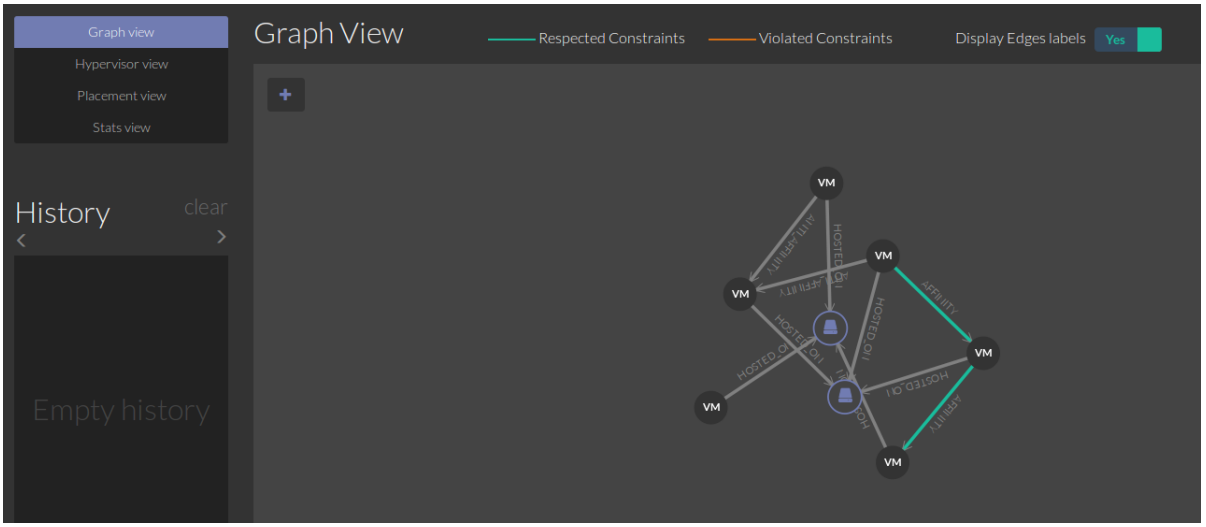

FIGURE 4.2: Our interface with graph

Consequently the first step for the resource allocation framework is to transform the specified user request via this interface to the representations of 4.1. The next step consists in producing the adjacency matrices shown in Figures 4.1, 4.2 and 4.3.

The first matrix 4.1 represents the allocation matrix for Figure 4.1.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

The second matrix 4.2 is an adjacency matrix for anti-affinity relations.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.2}$$

The third matrix 4.3 takes into account the connectivity and relationships in the provider's infrastructure. Note that connectivity of the servers to switches in the provider infrastructure are also included in the representation.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{4.3}$$

Besides the description of the topologies and connectivity in the networks requested by the users and in the provider infrastructure, we also need to describe the resources themselves to find matching nodes and links to the requested resources. The resources are specified using attributes and the notion of characteristics to indicate their type and provided service such as compute, storage and communications resources or services. This representation is illustrated in Figure 4.3.

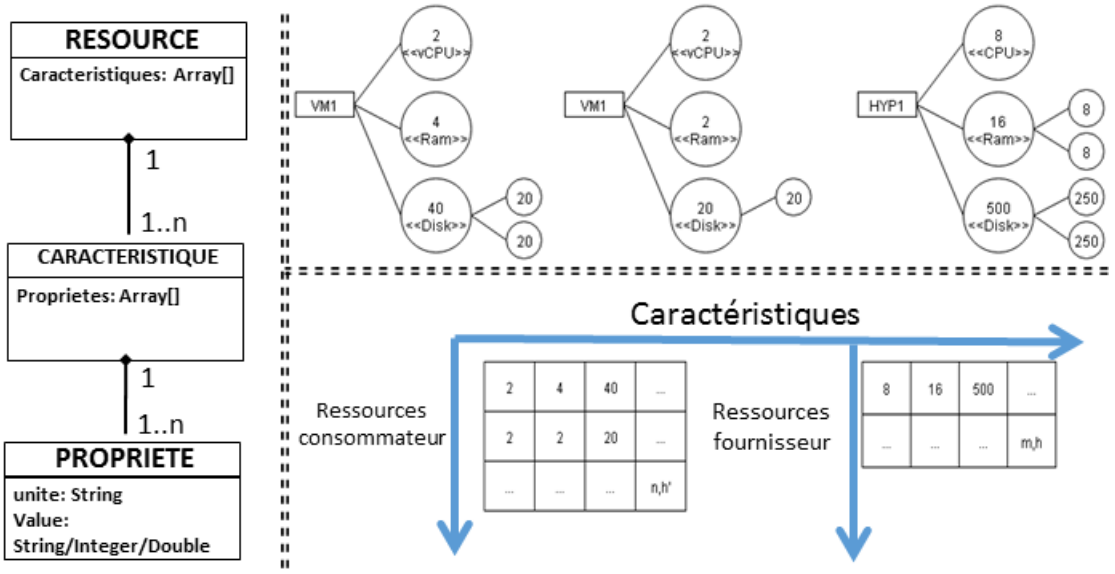

FIGURE 4.3: Matrix representation of resources

4.2 Cloud Resource Placement Algorithm

Since solving the cloud resource allocation problem using an exact integer linear program or exact methods leads to high (exponential) complexity and encounters scalability issues, we do not use the inter program formulation and tools such as CPLEX to solve the problem. Besides, multiple objectives have to be taken into account and this typically exacerbates the complexity of exact algorithms.

We prefer good suboptimal solutions that can be found in reasonably short execution times and are capable of fulfilling practical requirements. To scale and provide solutions in reasonable resolution and execution times, we resort to the set NSGA II [108] genetic algorithm to solve the cloud resource allocation problem addressed in this work. NSGA II is a popular non-domination based genetic algorithm for multi objective optimization and is hence an appropriate choice for the considered problem.

The NSGA II complexity is $O(MN^2)$ where M is the number of objectives and N is the population size. To accelerate the execution time of the algorithm and reduce the impact of this complexity, we modify the algorithm as depicted in Fig.4.4 and develop a specific initialization algorithm (Algorithm 1) based on the observations on the outcomes of random drawing of solutions and their tuning according to the success rate in finding viable and acceptable solutions, those that meet all the capacity constraints. We use a probability derived from these observations to control and drive future drawings and thus speed up the exploration towards good solutions. Basically, we exploit the past observations in order to fine-tune future explorations.

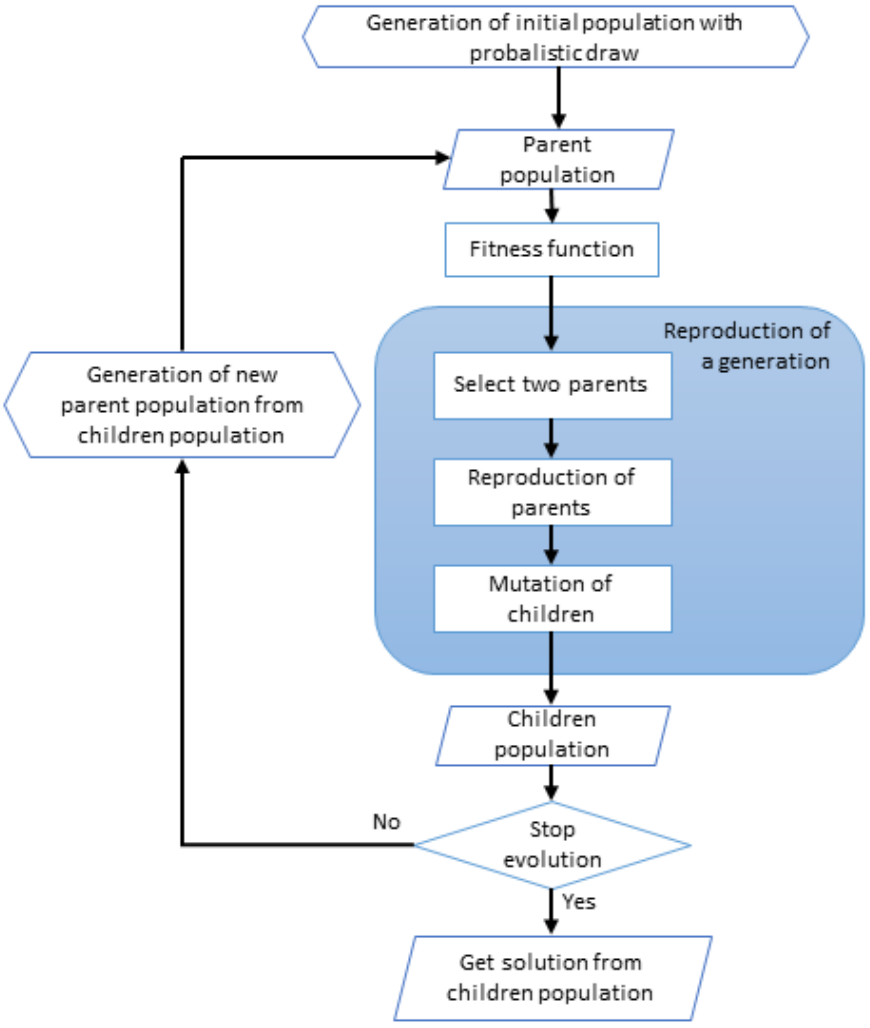

FIGURE 4.4: Genetic Algorithm Principle

Figure 4.4 shows the general structure of our modified NSGA II. Initialization of the population is based on random drawing. The selection operator is a binary tournament selection operator using Pareto dominance. The crossover operator is the Simulated binary crossover (SBX) operator. The SBX crossover operator simulates the offspring distribution of binary-encoded single-point crossover on real-valued decision variables.

We use as mutation operator the polynomial mutation (PM) operator. The PM operator simulates the offspring distribution of binary-encoded bit-flip mutation on real-valued decision variables. Similar to SBX, PM favors offspring nearest to the parents. An individual consists of one chromosome and is a resource placement solution. A chromosome corresponds to boolean variable X_{jk} indicating whether the requested resource k is assigned to the provider resource j. The chromosome is composed of genes with each gene being a real number representing the id of the provider resource j. We want an algorithm that converges faster to an approximate solution and that respects the constraints (Fig.4.5). To this end, we check at population initialization all the constraints

described in Algorithm 3. The size of the population is fixed and is an input to the algorithm.

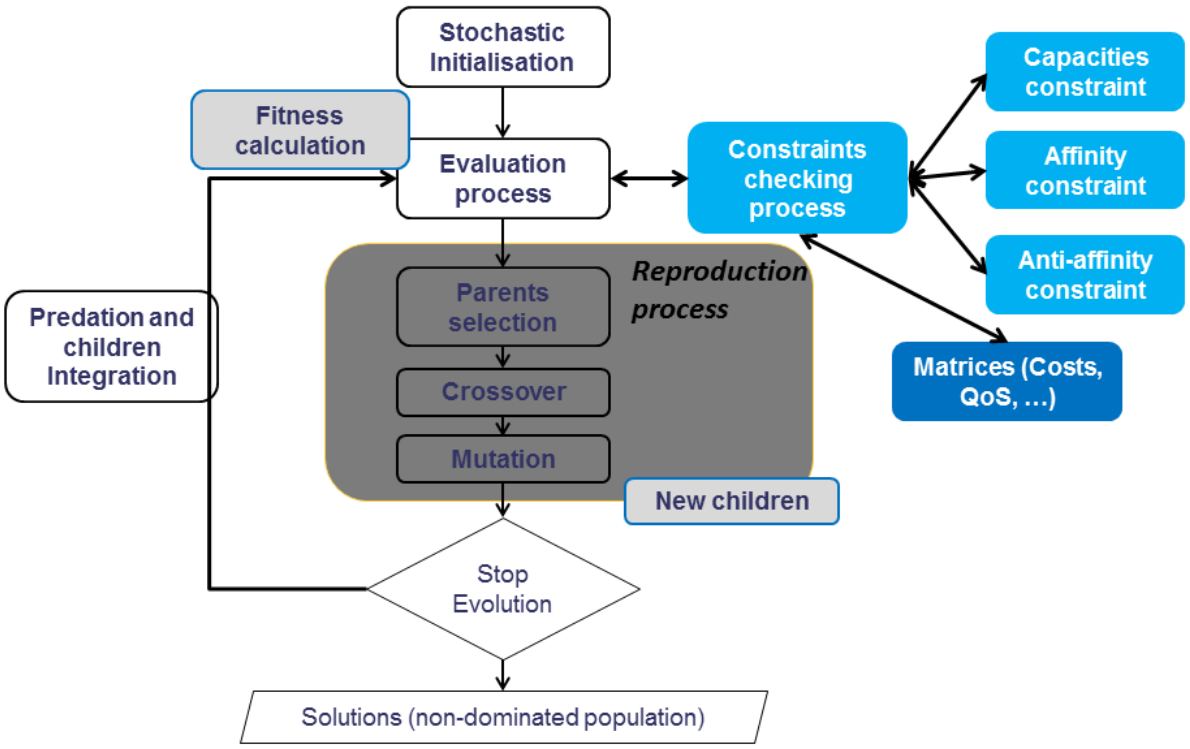

FIGURE 4.5: New Genetic Algorithm Principle

We note Pop the population, p an individual and S the variable set of one solution. Variable s is just one allocation solution. Variable P_{jl} represents a matrix of provider resource capacities and C_{kl} is a matrix of requested resource capacities.

We introduce three variables: DC_j specifies the datacenter where server j resides. The booleans $NODEused_j$ and $DCused_i$ indicate if a node or a datacenter is used. These variables are used to implement the constraints and relationships between resources.

The variables used in the definition of our algorithm are listed for easy reference in Table 4.1:

TABLE 4.1: Variables used in our algorithm

	Algorithm variables
Pop	Set of individual $Pop = \{1, ..., p\}$
S	Chromosome: Set of gene $N = \{1, ..., s\}$
DC_j	specifies the datacenter from server j
$NODEused_j$	Boolean variable if a node is used
$DCused_i$	Boolean variable if a datacenter is used

The next subsections describe the initialization operator that we have improved and the used crossover and mutation operators.

4.2.1 Initialization Operator

Population initialization is achieved through an initialization operator. The initialization operator is a crucial task in evolutionary algorithms because it can affect the convergence speed and also the quality of the final solution. Random initialization is the most commonly used method to generate candidate solutions (initial population) but this method does not meet the constraints. We propose a random drawing initialization (Fig.4.6), guided by the ability of provider resources to host customer requests, to generate the initial population including the satisfaction of constraints.

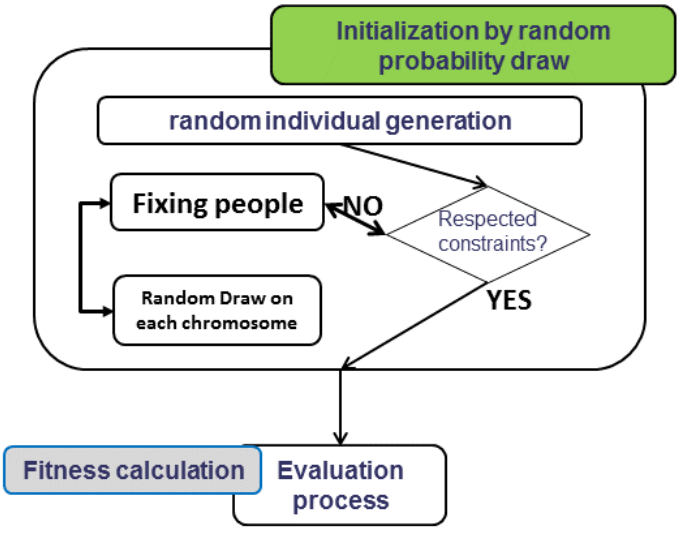

FIGURE 4.6: New Initialization Operator Principle

Algorithm 1 Initialize population

Require: Pop, S, R_j
  for each $p \in Pop$ **do**
    initialize R_j
    for each $s \in S$ **do**
      repeat
        $s = randomDraw(p, R_j)$
      until isPossible(p)
    end for
  end for

First time, we want to obtain random solutions which respect constraints. To initialize the population, we proceed for each individual to a random drawing (Algorithm 1) on their genes, by randomly selecting the resource that will host each gene. As long as the obtained solution is not valid, does not respect the constraints and relationships

expressed in the original requests, the process is repeated, i.e. the drawing continues until valid solutions are found. The drawing is not a simple random drawing since it is guided by the ability of provider resources to host customer requests. We introduce a vector R_j that stores the number of failed tests on provider resource j to capture and estimate this capability and use it to drive ensuing explorations.

Algorithm 2 Random draw

Require: R_j
Ensure: $p \in Pop$
  List greatLocation
  for each $j \in P_{jl}$ **do**
    if j == Collections.min(R_j) **then**
      $greatLocation.add(j)$
    end if
  end for
  $p = Random.select(greatLocation)$
  return p

Algorithm 2 provides as output a viable provider resource (one that can host the request and respect the constraints and the relationships) to be used for the initialization process in Algorithm 1. We retain and store in a list the best provider resources, those that are part of the minimum values of R_j. The R_j values keep updating during the constraints checking process (Algorithm 3). Finally, we randomly draw a resource from the produced list of best provider resources and send it to the initialization process.

Algorithm 3 Checks whether the solution is valid

Require: p, P_{jl}, R_j
Ensure: $p \in Pop$

```
    // Capacity constraint [Eq. 3.4] below
  for each q ∈ Q_p do
    capacityProvider = getVariable(q)
    for each j ∈ P_jl do
      capacityProvider = capacityProvider - capacityCustomer
    end for
    if capacityProvider < 0 then
      increaseAmount(R_j); return failureConstraint
    end if
  end for
    // All relationship constraints below
  switch RelationshipConstraints do
    case ColocalizationSameDC : // [Eq. 3.8]
      for each q ∈ Q_p do
        if DC[q] ! = anyDC then
          return failureConstraint
        end if
      end for
    end case
    case ColocalizationSameNode : // [Eq. 3.9]
      for each q ∈ Q_p do
        if q ! = anyNode then
          return failureConstraint
        end if
      end for
    end case
    case TotalSeparation : // [Eq. 3.10]
      for each q ∈ Q_p do
        if DCused[DC[q]] == true OR
        NODEused[q] == true then
          return failureConstraint
        else
          NODEused[q] = true
        end if
      end for
    end case
    case SeparationOnDifferentNode : // [Eq. 3.12]
      for each q ∈ Q_p do
        if NODEused[q] == true then
          return failureConstraint
        else
          NODEused[q] = true
          DCused[DC[q]] = true
        end if
      end for
    end case
```

4.2.2 Crossover Operator

We use a Simulated Binary Crossover (SBX) for our crossover operator. SBX is based on One-point crossover. A single crossover point on both parents string is selected. All data beyond that crossover point in either parent is swapped between the two parent organisms. There are two important properties of One-Point Crossover :

- the average of the decoded parameter values is the same before and after the crossover (Figure 4.7);

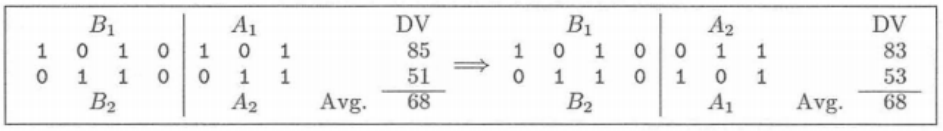

Figure 1: The action of single-point crossover on two random strings is shown. DV stands for the decoded parameter value. Notice that the average of the decoded parameter values is the same before and after the crossover operation.

FIGURE 4.7: From the article Simulated Binary Crossover for Continuous Search Space [1] describing the crossover process

- The Spead factor β is defined as the ratio of the spread of offspring points to that of the parent points [Eq. 4.4];

In order to implement this crossover operator for any two parent solutions $p1$ and $p2$, a dimensionless spread factor β has been defined as the ratio of the spread of created children solutions $c1$ and $c2$ to that of the parent solutions as follows:

$$\beta = \left| \frac{c1 - c2}{p1 - p2} \right| \tag{4.4}$$

We use the Simulated Binary Crossover (SBX) proposed by Deb and Agrawal in 1995 [1]. SBX was designed with respect to the one-point crossover properties in binary-coded GA.

- **Average Property**: The average of the decoded parameter values is the same before and after the crossover operation;
- **Spread Factor Property**: The probability of occurrence of spread factor $\beta \approx 1$ is more likely than any other β value;

We implemented this Simulated Binary Crossover (SBX) function in compliance with the philosophy of the proposed method in 1995. SBX uses a probability distribution of β in SBX [Eq.4.5] that should be similar to the probability distribution of β in Binary-coded crossover.

$$\rho(\beta) = \begin{cases} 0.5(n+1)\beta^n, & if \beta \leq 1; \\ 0.5(n+1)\frac{1}{\beta^{n+2}}, & otherwise \end{cases} \tag{4.5}$$

SBX is a real-coded crossover operator that create two children solutions from two parent solutions. It uses a probability distribution centering the parent solutions and two children solutions are created based on that probability distribution. Creating children solutions using a fixed probability distribution, which does not depend on the location of the parent solutions, makes the search adaptive. In order to create two children solutions $c1$ and $c2$ from the parent solutions $p1$ and $p2$ using the above probability distribution, the following procedure is used.

- Create a unified random number u between 0 and 1.
- Find a $\beta^{'}$ for which the cumulative probability respects [Eq. 4.6]

$$\int_0^{\beta^{'}} \rho(\beta) d\beta = u \tag{4.6}$$

- Knowing the value of $\beta^{'}$, the children points are calculated using [Eq. 4.7] :

$$\begin{aligned} c1 &= 0.5[(p_1 + p_2) - \beta^{'} |p_2 - p_1|] \\ c2 &= 0.5[(p_1 + p_2) - \beta^{'} |p_2 - p_1|] \end{aligned} \tag{4.7}$$

As previously mentioned, our implementation of SBX is based on the article written by the authors in 1995 [1] and which is implemented in NSGA II by Deb. We observed that the crossover operator SBX is most appropriate for our resource allocation problem (See section 4.3).

4.2.3 Mutation Operator

Deb and Agrawal [109] suggested a polynomial mutation (PM) operator with a user-defined mutation probability parameter. We use them in our implementation of the mutation operator with a value of 0.20 for the mutation probability. In this operator, a polynomial probability distribution is used to perturb a solution in a parent's vicinity. The probability distribution in both left and right of a variable value is adjusted so that

no value outside the specified range [a, b] is created by the mutation operator where a and b are lower and upper bounds of the variable. Polynomial mutation (PM) operator attempts to simulate the offspring distribution of binary-encoded bit-flip mutation on real-valued decision variables. Similar to SBX, PM favors offspring nearer to the parent. The distribution index controls the shape of the offspring distribution. Larger values for the distribution index generates offspring closer to the parents.

4.2.4 Constraints validation

The point with evolutionary algorithms is, they can hardly handle strict constraints. Technically, there exist different possible methods to have his issue bypassed: A weakness of evolutionary algorithms is: they can hardly handle strict constraints. Technically, there exist different methods to address this weakness:

- by excluding the individuals that are not in line with (or violate) the constraints;
- by fixing faulty individuals through a repair process;
- by preserving the correct individuals through special operators;
- by modifying the search space and guiding the algorithm all the way through (via hyperplanes).

This work focused on the first two methods and assessed their efficiency to eventually select them. Method 1, which excludes out-of-limit individuals, reveals to be inefficient as too many individuals end up excluded. Method 2, that we retain, is designed to fix faulty individuals by running a tabu search algorithm on them (Figure 4.8). The repair process through a tabu search enhanced with a genetic algorithm (NSGA III) is described on Figure 4.9.

The repair process through the tabu search is made possible through the browsing of a list of potential hosts for the virtual machines (Figure 4.10) currently hosted on overloaded servers. The repair process is launched whenever invalid individuals are identified. The fixing method aims at making them comply with the constraints. Every faulty gene found within an individual will then be processed and modified accordingly.

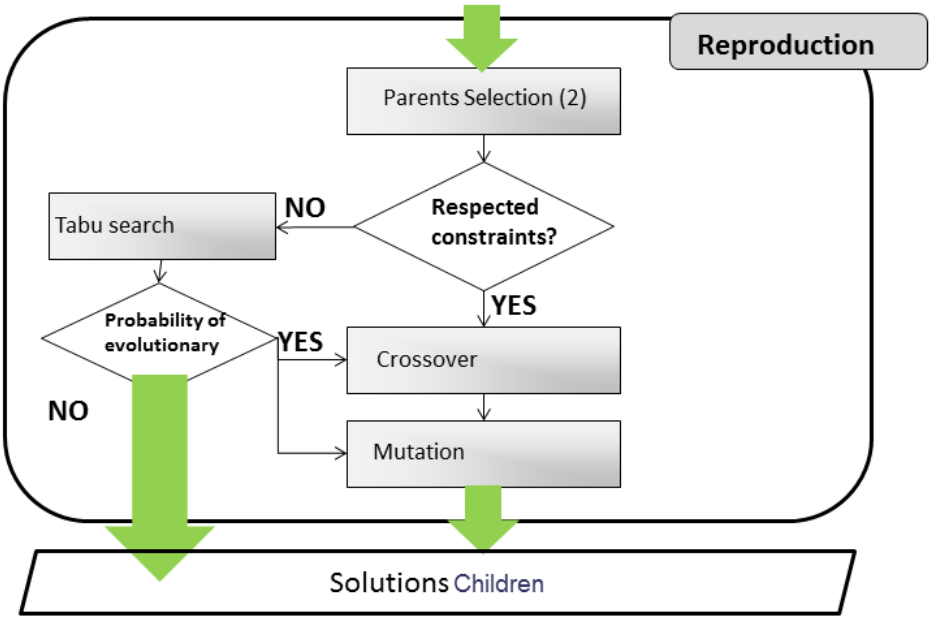

FIGURE 4.8: Tabu Search integration within NSGA

1: **procedure** REPAIR(I) ▷ I is an individual
2: $serversError \leftarrow exceedingDetection(I)$ ▷ we are scanning for servers where constraints are exceeded
3: **for** $i \neq numberOfVM()$ **do**
4: $server \leftarrow getServerOfVM(I, i)$
5: **if** $server \in serversError$ **then**
6: $I(i) \leftarrow findNeighbour(I, i)$
7: **end if**
8: **end for**
9: **end procedure**

FIGURE 4.9: Tabu search for the repair process individuals of NSGA

1: **procedure** FINDNEIGHBOR(I, i) ▷ i is an ID of a VM
2: **for** $j \neq numberOfServer(I)$ **do**
3: **if** $isValidAllocation(i, j)$ **then**
4: $return(j)$ ▷ We return a valid server
5: **end if**
6: **end for**
7: **end procedure**

FIGURE 4.10: Finding the nearest valid neighbor

4.3 Performance Evaluation

We evaluate the performance of the proposed algorithm at high load when the system is under stress in order to reveal the limitations and capabilities of our genetic algorithm. We assess performance in terms resolution time, downtime cost, global resource usage, exploitation cost and the size of the induced reconfigurations plans. These simulations are performed on an Intel NUC computer with Intel Celeron 847 (1.1 GHz - 2 MB Cache) and 4GB DDR3 RAM.

For each algorithm, we perform simulations on linear and random scenarios. The linear scenarios consists in allocating consumer resources (Virtual Machines) on provider resources (Hypervisors) that can host only two consumer resources. The random scenarios is to host virtual machines that have a random capacity on random capacity hypervisors.

4.3.1 Round Robin Evaluation

We evaluate in this section the Round Robin algorithm. This algorithm checks the pool of hypervisors by cycling through them and allocating virtual machines on each hypervisor during each visit. For each scenario, we study the resolution time, operating and usage costs generated, the number of allocated customer resources, the number of hypervisors used and the filling rates of the hypervisors.

4.3.1.1 Linear scenario

First we study the resolution time for a solution. Under the linear scenario, we present the resolution time of the Round Robin algorithm in Fig. 4.11. We observe a resolution or execution time that grows exponentially with problem size and hence an algorithm that does not scale.

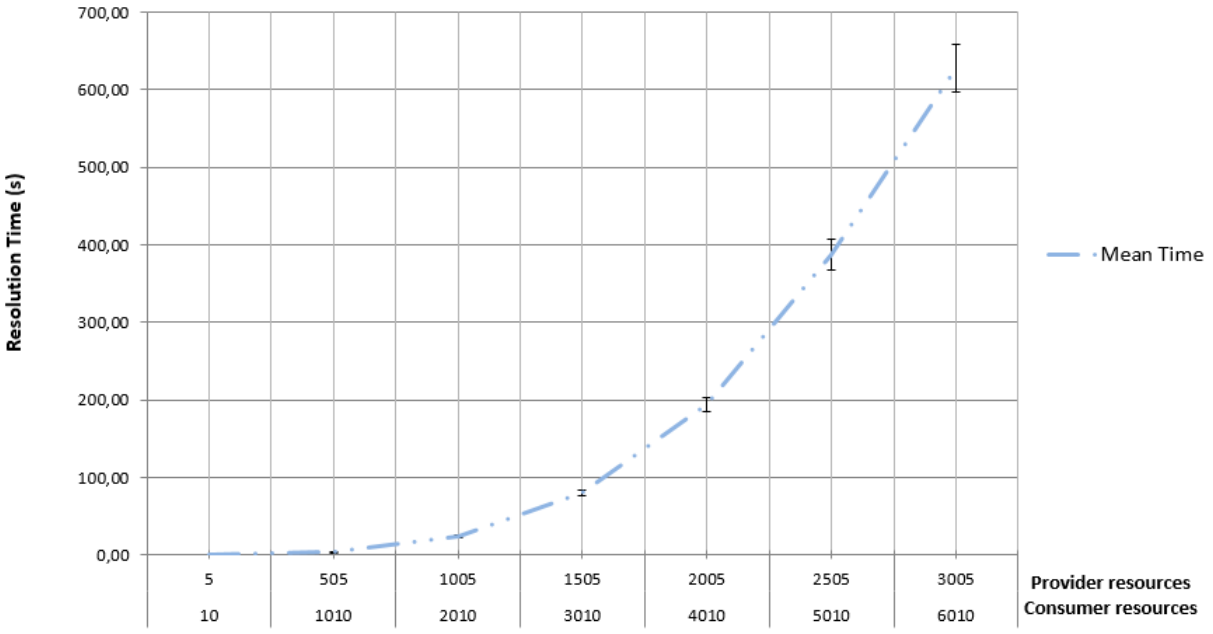

FIGURE 4.11: Round Robin Resolution Time with linear scenarios

Next, we highlight the generated costs related to the scenarios in Fig. 4.12. The global costs are the operating and usage costs of the resources. All customer resources are hosted and there are no downtime costs.

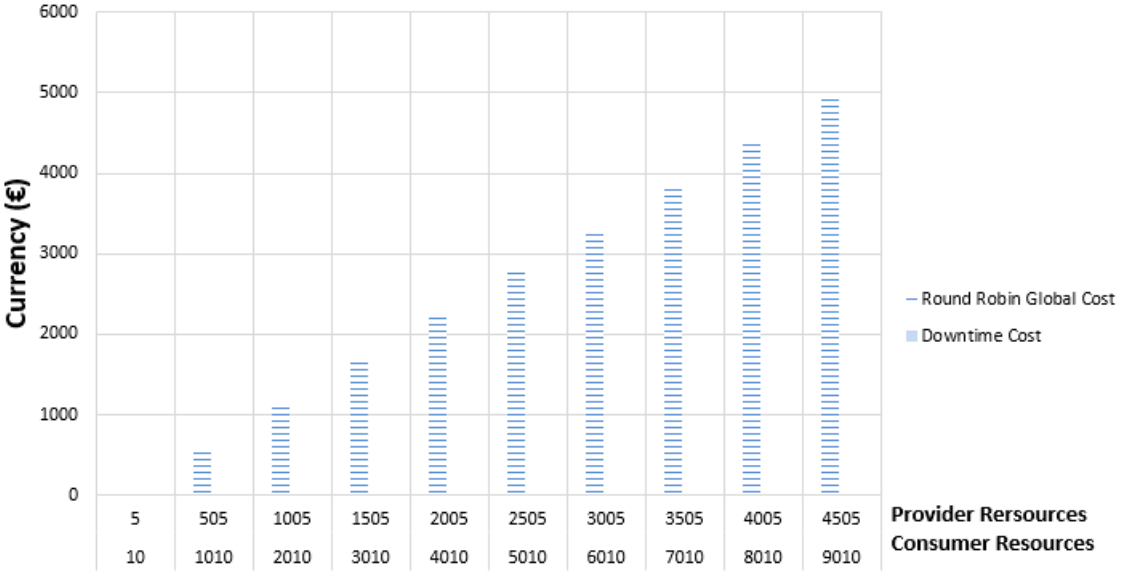

FIGURE 4.12: Round Robin Global Cost with linear scenarios

Finally, we show the proportion of hosted virtual machines and the percentage of hypervisors used to host these machines (Fig. 4.13). All resources are used and hosted.

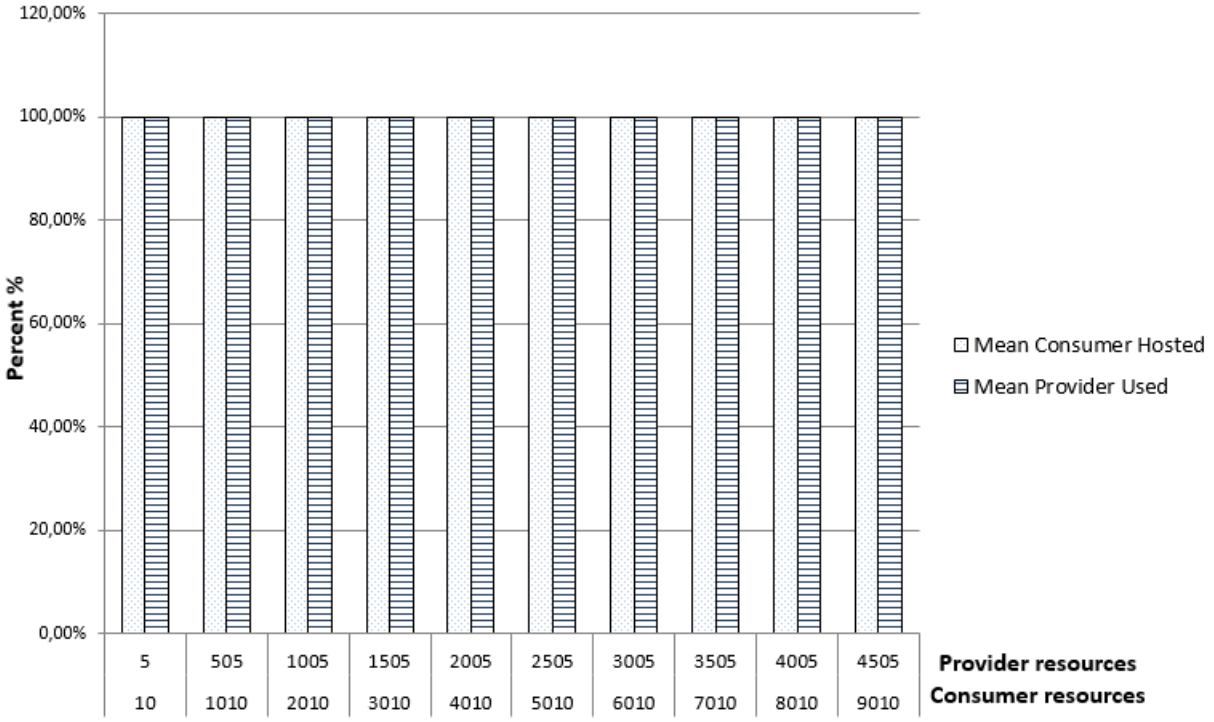

FIGURE 4.13: Round Robin Resource Utilization with linear scenarios

4.3.1.2 Random scenario

For the random scenario, we present the resolution (execution) time of the Round Robin algorithm (Fig. 4.14). The time needed to allocate resource rises exponentially with problem size confirming the inability of the Round Robin to scale.

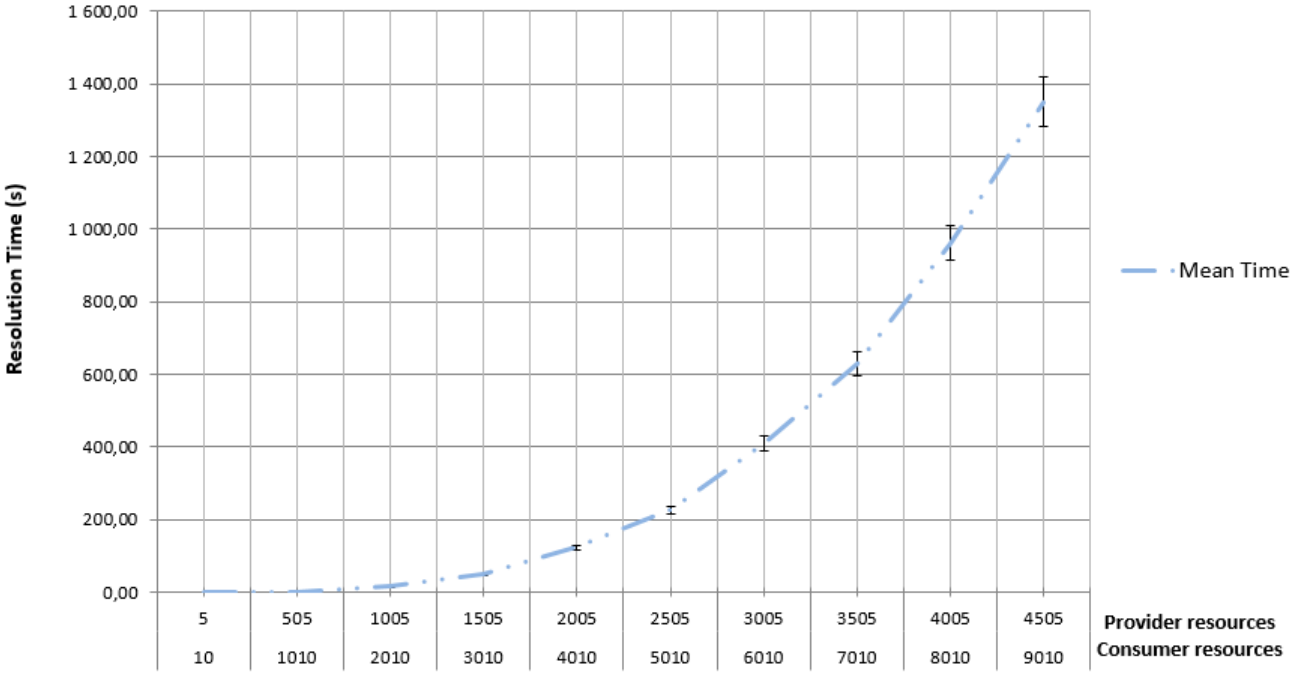

FIGURE 4.14: Round Robin Resolution Time with random scenarios

Next, we highlight the costs generated for the random cases in Fig. 4.15. The global costs are the operating and usage costs of the resources. All customer resources are hosted and there are no downtime costs experienced.

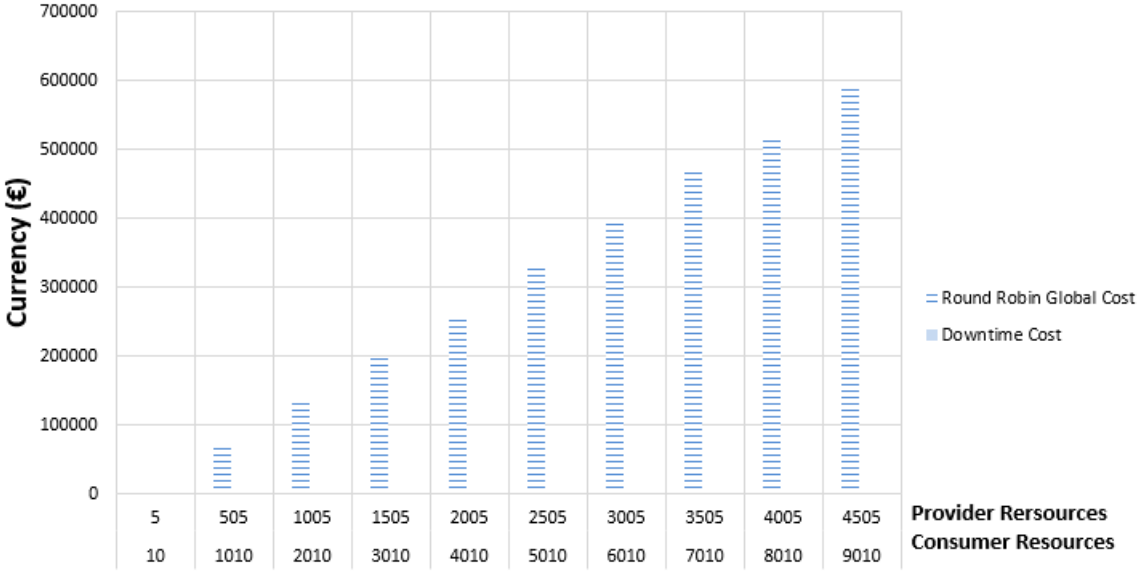

FIGURE 4.15: Round Robin Resolution Time with random scenarios

Finally, we show the percentage of hosted virtual machines and the proportion of hypervisor consumed for hosting in Fig. 4.16. We find that all customers resources are hosted but the provider infrastructure is not consolidated. Indeed, under normal circumstances

the round robin algorithm must use all the provider resources. This is explained by the fact that resources are heterogeneous and that the algorithm does not allocate optimally the resources.

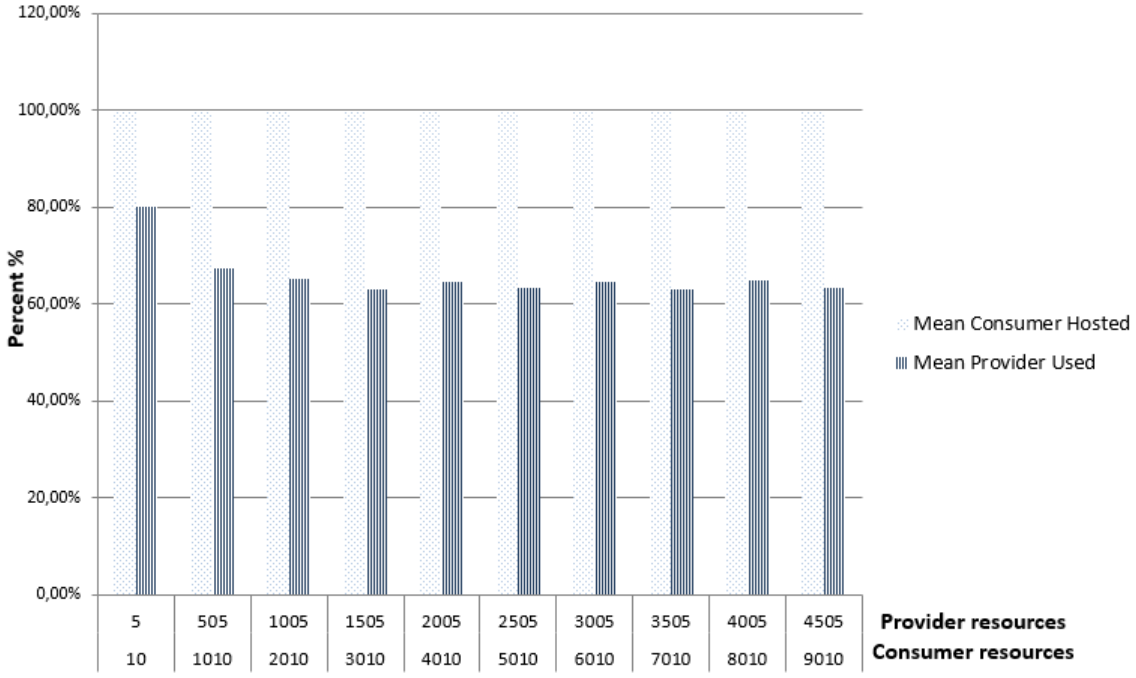

FIGURE 4.16: Round Robin Resolution Time with random scenarios

4.3.1.3 Conclusion

The advantage of the Round Robin algorithm is that it can quickly respond to customer requests but unfortunately, the allocation of virtual resources makes it impossible to manage costs efficiently if the demands are heterogeneous. Moreover with 3000 virtual machines and 1500 hypervisors, resolution time is exponential compared to smallest values. The Round Robin algorithm does not efficiently allocate resources in the case of random scenarios either.

4.3.2 Constraint Programming Evaluation

We evaluate in this section the constraint programming method. For each scenario, we study the resolution time, operating and usage costs generated, the number of allocated customer resources, the number of hypervisor used and the filling rates of the hypervisors.

4.3.2.1 Linear scenario

We report the resolution time of the constraint programming algorithm Fig. 4.17 for the linear scenario and observe as expected an exponential increase in execution time with problem size and consequently the typically scalability issues.

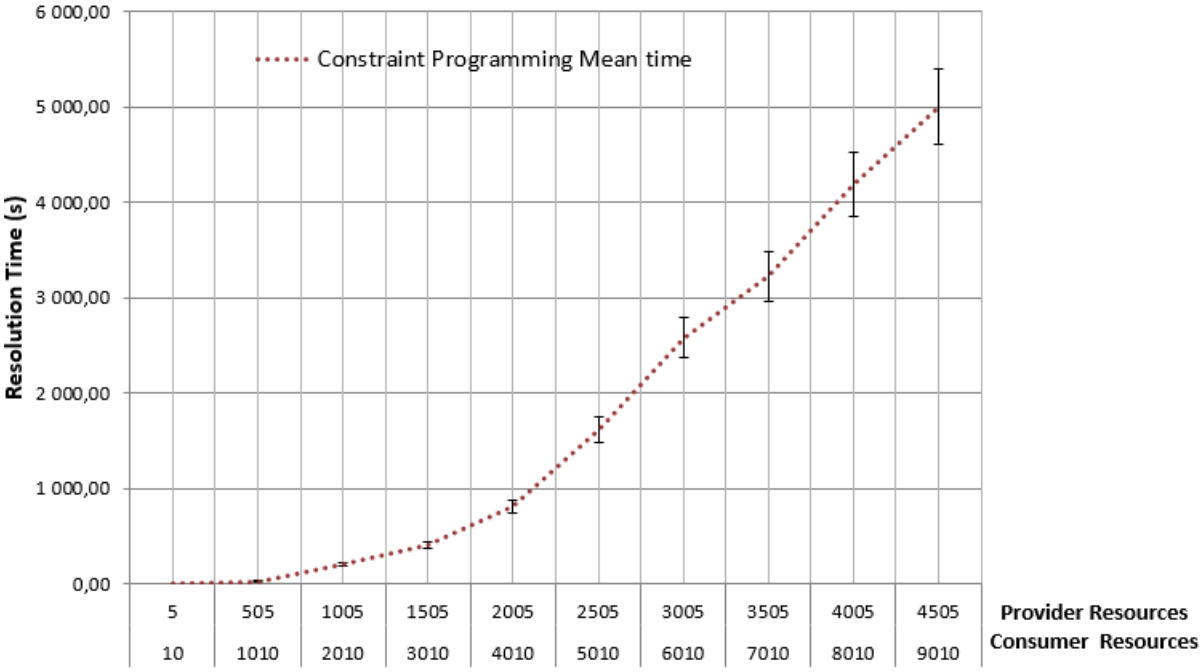

FIGURE 4.17: Constraint Programming Resolution Time Comparison with linear scenarios

Next, we present the results of cost assessments for constraint programming in Fig. 4.18. The reported global costs are the operating and usage costs of resources. For the evaluated scenarios, all customer resources are hosted and we see that there are no downtime costs.

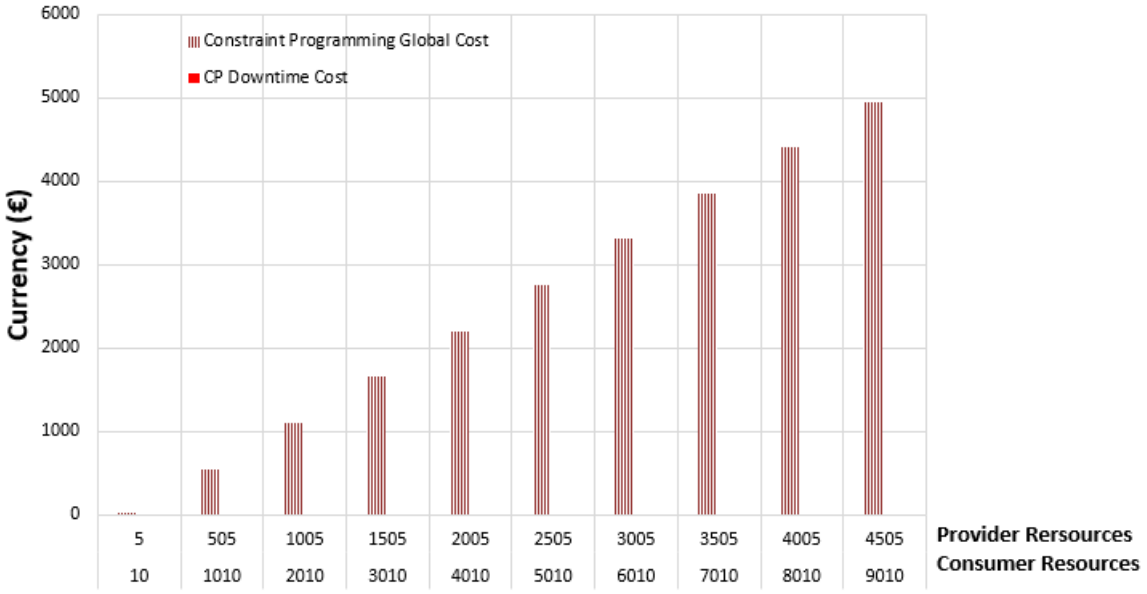

FIGURE 4.18: Constraint Programming Global Cost with linear scenarios

Fig. 4.19 depicts the amount of consumer resources that are succefully hosted and the proportion of used hypervisors to serve the requests. All the hosts are used and all requests are served.

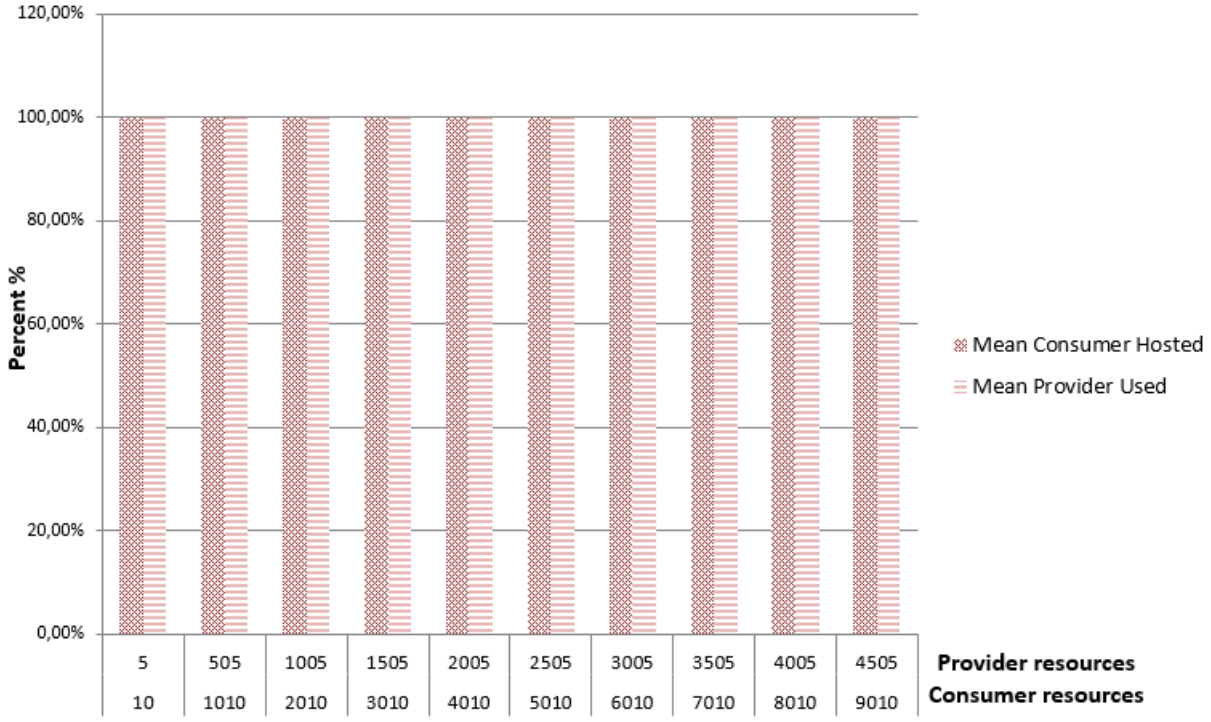

FIGURE 4.19: Constraint Programming Resource Used with linear scenarios

4.3.2.2 Random scenario

For random scenarios, Fig. 4.20 depicts the results for the resolution time for constraint programming and shows that this time grows exponentially with problem size. Constraint programming experiences scalability problems similar to Round Robin.

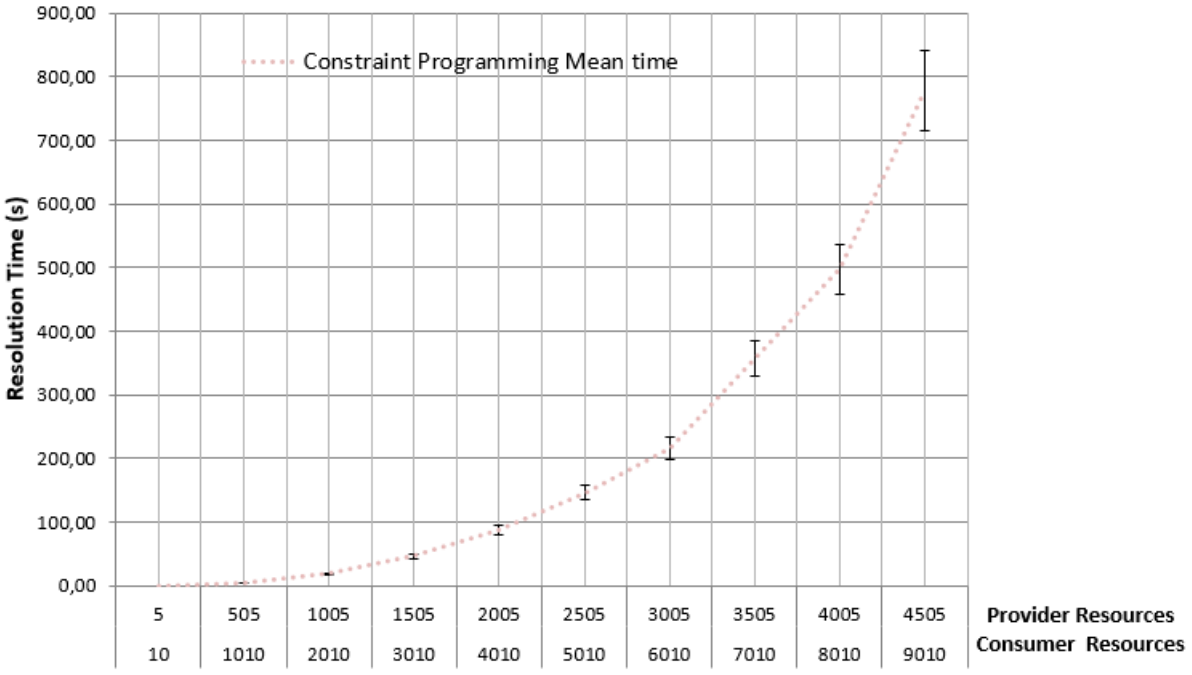

FIGURE 4.20: Constraint Programming Resolution Time with random scenarios

The global costs composed of the operating and usage costs of resources is provided by Fig. 4.21 for the random scenarios. No downtime costs are encountered for these scenarios.

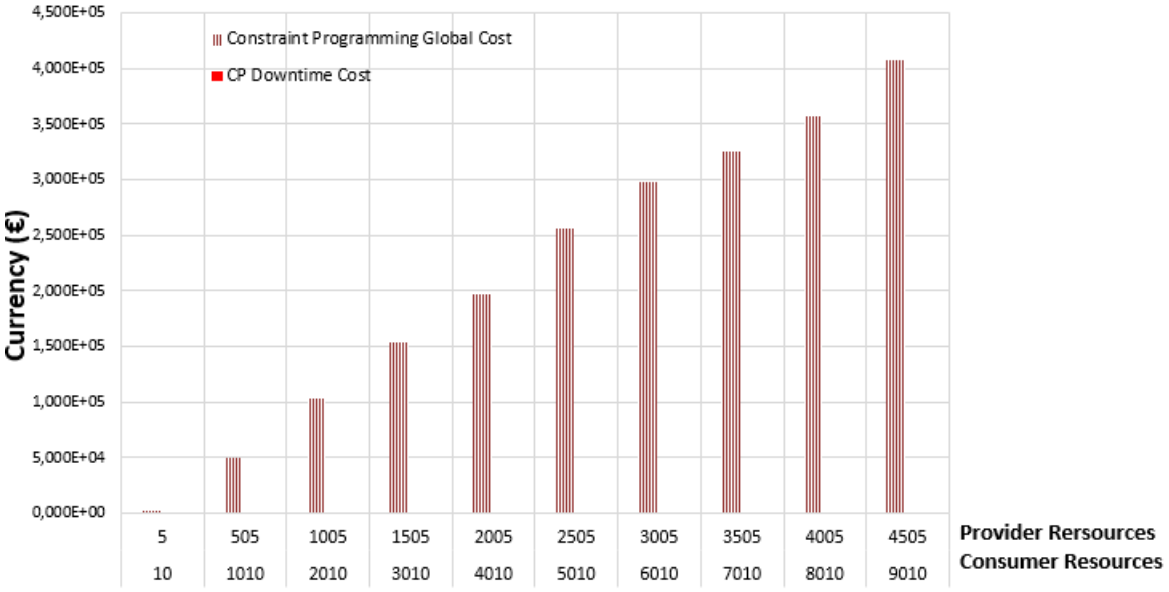

FIGURE 4.21: Constraint Programming Global Cost with random scenarios

Fig. 4.22, that reports the proportion of consumer resources that are hosted and the proportion of used hypervisors, shows that all customer requests are hosted hypervisors used . We find that all customers resources are hosted. Constraint programming does not use all the hypervisors for the evaluated scenarios.

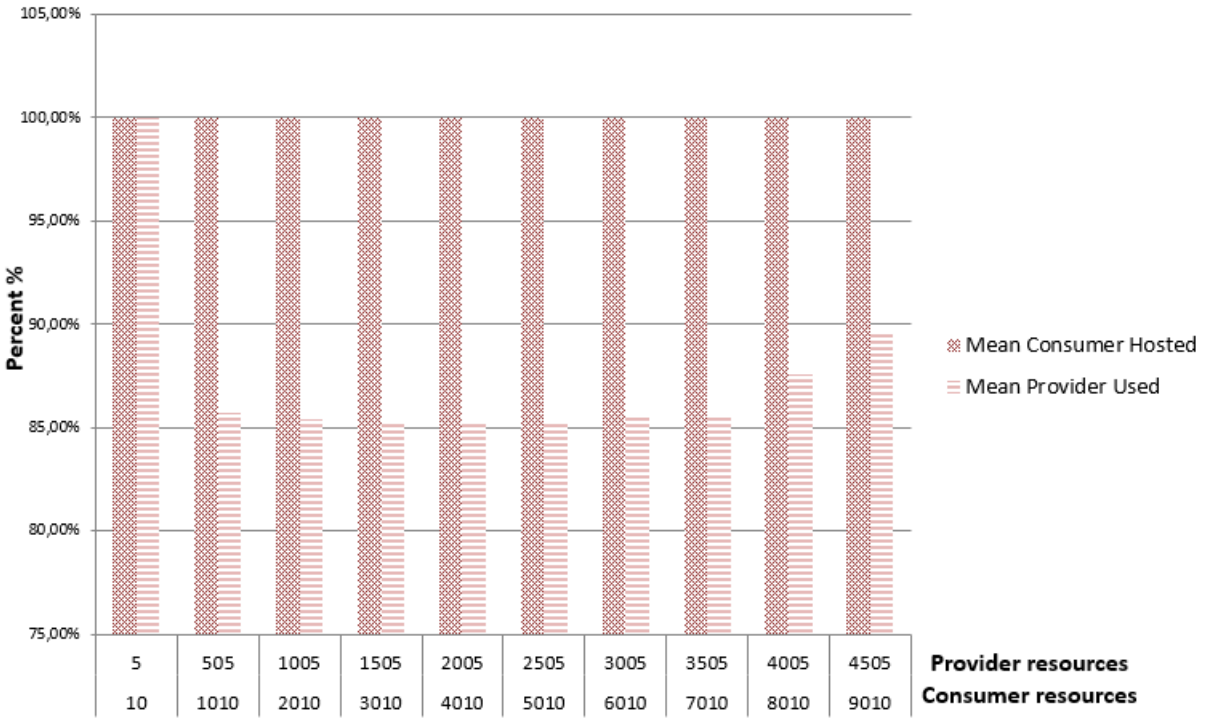

FIGURE 4.22: Constraint Programming Resource Used with random scenarios

4.3.2.3 Conclusion

The Constraint Programming method provides an exact solution to the allocation problem but the time resolution is exponential as a function of the problem size. The resolution of the resource allocation problem using constraint programming improves this result but has the same scalability problems. These algorithms provide an exact solution in the case of linear and random scenarios. The limit of constraint programming is approximately around 10000 virtual machines and 5000 hypervisors. Beyond this limit resolution time is excessive for practical purposes and typical applications. The algorithm requires also more memory to run. In addition, these algorithms are not suitable for solving optimization problems with multiple objectives.

4.3.3 Genetic Algorithm Evaluation

This section reports the results of performance evaluation of our modified NSGA II algorithm. The evaluated performance indicators are again: the resolution time, the generated operating and usage costs, the number of allocated customer resources, the number of hypervisor used and the rates at which these hypervisors are consumed. Table 4.2 describes the parameters settings used for this evaluation of NSGA II.

TABLE 4.2: NSGA II Default parameter

Parameter	Value
populationSize	100
sbx.rate	0.70
sbx.distributionIndex	15.00
pm.rate	0.20
pm.distributionIndex	15.00
maxEvaluations	10000

We similarly conduct the evaluations for linear and random scenarios.

4.3.3.1 Linear scenario

Fig. 4.23 depicts the results of performance evaluation for the NSGA II resolution time for the linear scenario. The resolution time is quasi-polynomial in problem size and this is in line with its $O(MN^2)$ complexity where M is number of objectives and N the size of the population.

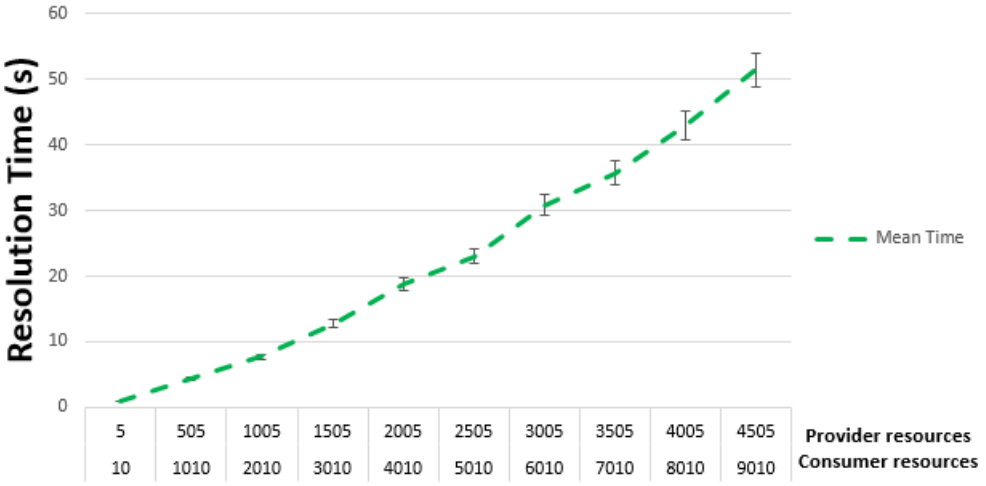

FIGURE 4.23: Modified NSGA II Resolution Time with linear scenarios

The generated costs or cost performance of NSGA II is depicted in Fig. 4.24 for the linear scenarios. Some customer resources are not served since some hosting resources are not used by the algorithm. NSGA II does not always find optimal solutions.

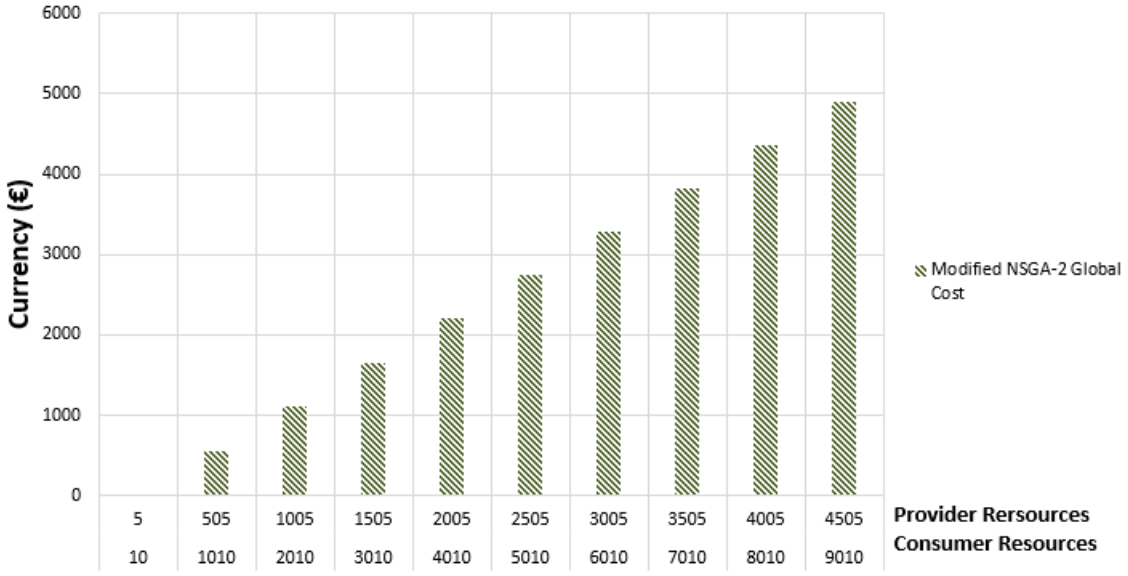

FIGURE 4.24: Modified NSGA II Global Cost with linear scenarios

Fig. 4.25 indicates thet NSGA II does not use all provider resources when the problem size increases and shows a 2% degradation for the simulated and evaluated linear scenarios.

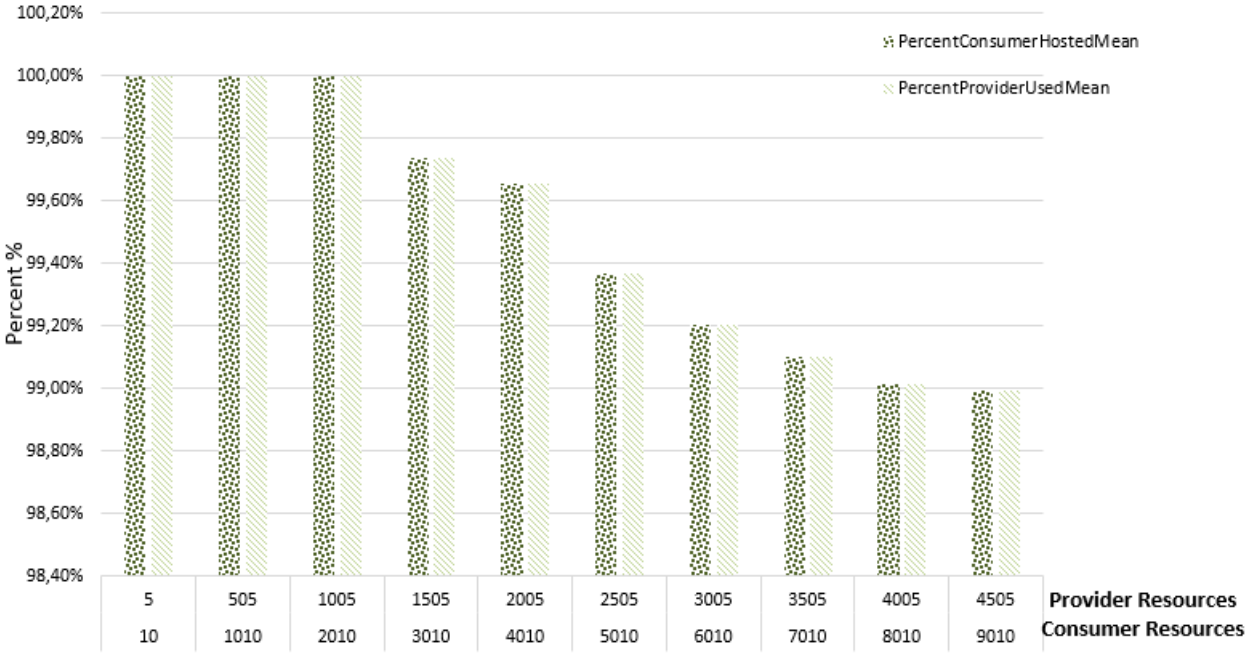

FIGURE 4.25: Modified NSGA II Resource Used with linear scenarios

4.3.3.2 Random scenario

For random scenarios Fig.4.26 shows that the resolution time remains quasi-polynomial with increasing problem size.

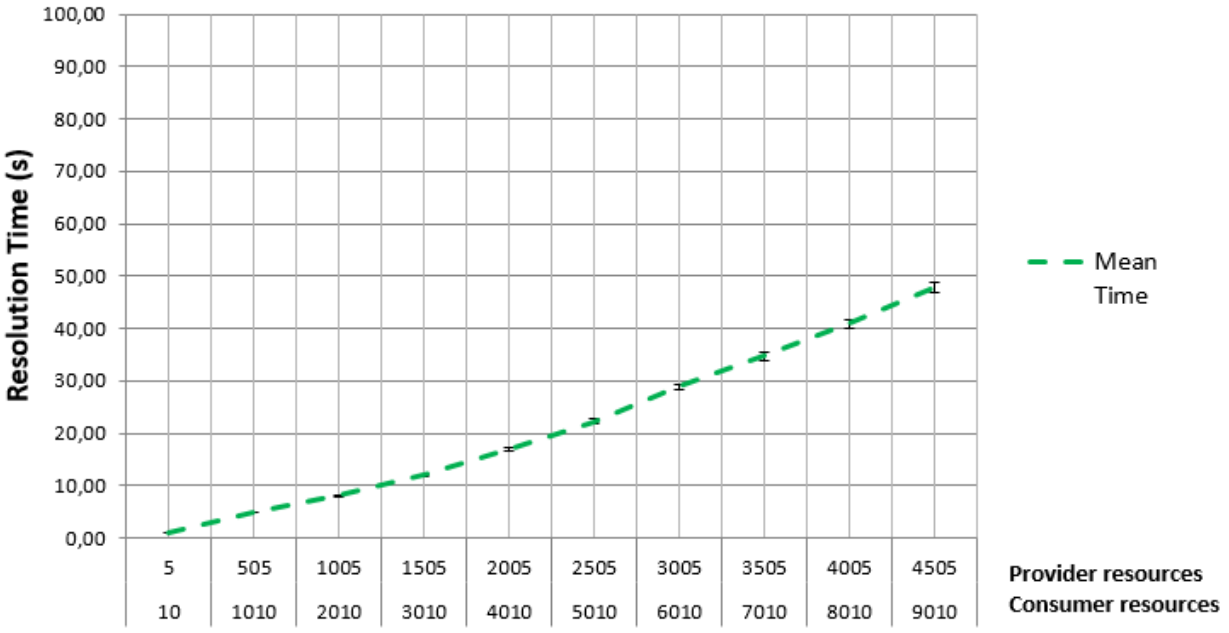

FIGURE 4.26: Modified NSGA II Resolution Time with random scenarios

The costs penalty for the NSGA II for the random scenarios is reported in Fig.4.27. Some customer requests are not served because the algorithm does not always find solutions.

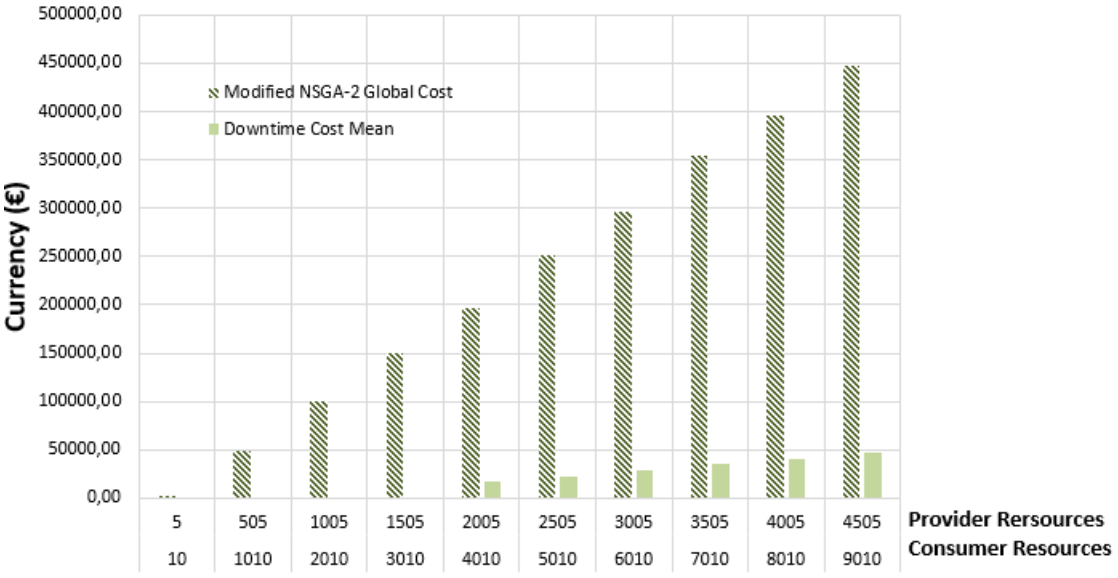

FIGURE 4.27: Modified NSGA II Global Cost with random scenarios

Fig.4.28 reports the resource utilization or usage and the hosting capabilities of NSGA II for the random scenarios. Since NSGA II does not always find solutions, some user requests are not served. For low load, small problem sizes, less than half the hypervisors are used by the algorithm highlighting the efficiency of the algorithm for reasonable loads. at higher loads or problem sizes NSGA II rejects some requests and does not use all the hypervisors indicating some weakness of the algorithm at higher loads for the random scenarios. This led us to propose enhancements to the genetic algorithms to improve their performance. This will be shown in the performance evaluation of NSGA III with the Tabu search to eliminate infeasible candidates or poor genes to accelerate resolution time and find more solutions.

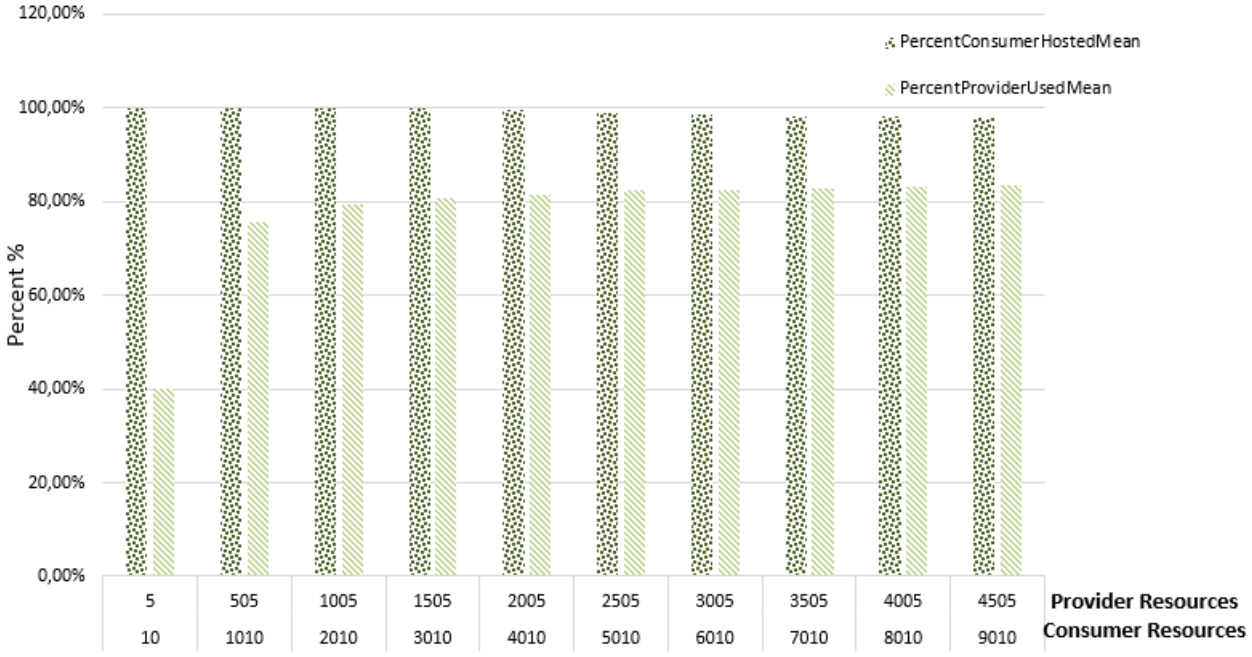

FIGURE 4.28: Modified NSGA II Resource Used with random scenarios

4.3.3.3 Conclusion

In conclusion, the NSGA II algorithm has a quasi-polynomial resolution time and is more appropriate for heterogeneous environments.

4.3.4 Algorithms comparison

We compare the performance of all the algorithms for the linear and random scenarios.

4.3.4.1 Linear scenario

The relative resolution time performance of the Constraint programming, the Round Robin and NSGA II is depicted in Fig. 4.29. NSGA II is faster than both Round Robin and Constraint Programming for the linear scenarios. Recall that NSGA II in very few cases does not find a hosting solution while the other algorithms accepted all the user requests as reported in the previous sections.

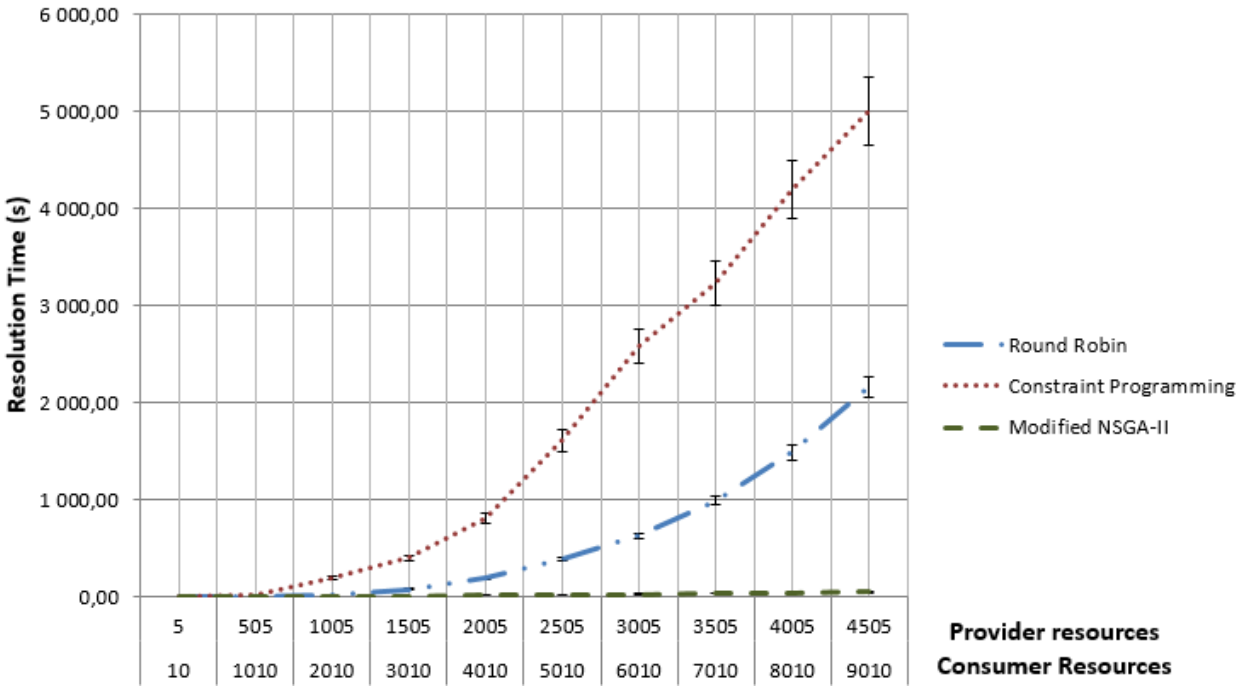

FIGURE 4.29: Modified NSGA II Resolution Time comparison with linear scenarios

The costs of the solution found by the algorithms are compared using Fig. 4.30 where the Round Robin and the Constraint Programming find solutions at the same lower costs than NSGA II. NSGA II cost increases with problem size.

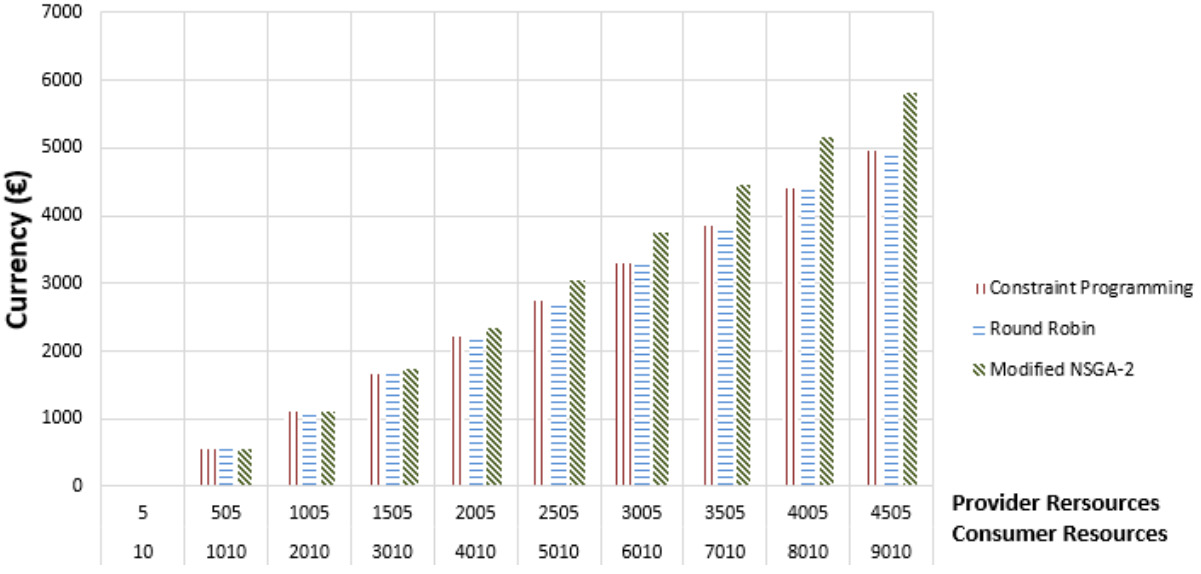

FIGURE 4.30: Modified NSGA II Global Cost comparison with linear scenarios

The percentage of customer requests accepted by the provider in the linear scenario is depicted in Fig. 4.31 that reveals better performance for the Constraint Programming and Round Robin and accept all the requests. NSGA II rejects a negligible fraction of the

requests, 1% compared to the other two algorithms confirming that the genetic algorithm does not always find solutions and needs to be enhanced and fine tunes to be more efficient. Note, however, that the genetic algorithm can handle multiple objectives while the two other algorithms use a single objective in their models. NSGA can potentially achieve the best possible tradeoff between multiple criteria and hence can address the conflicting users and providers interests.

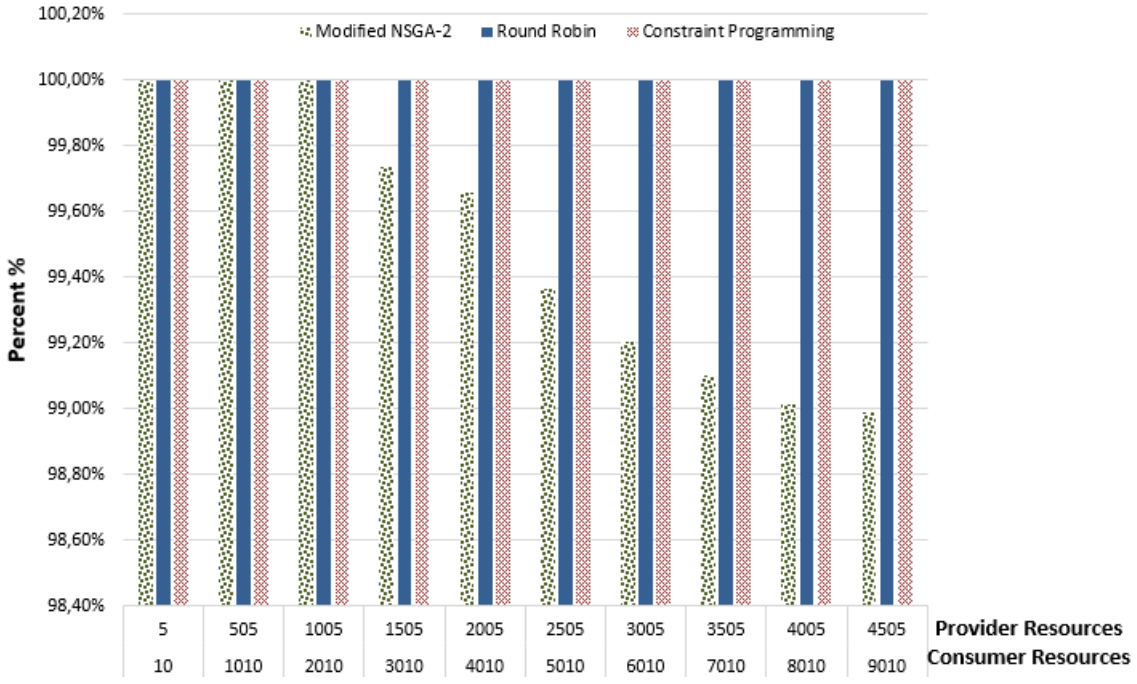

FIGURE 4.31: Modified NSGA II Customer Resource Hosted comparison with linear scenarios

The same behaviour for the algorithms can be observed in Fig. 4.32 where both the Constraint Programming and the Round Robin use all resources and host all the demands while NSGA II becomes less efficient when increasing problem size.

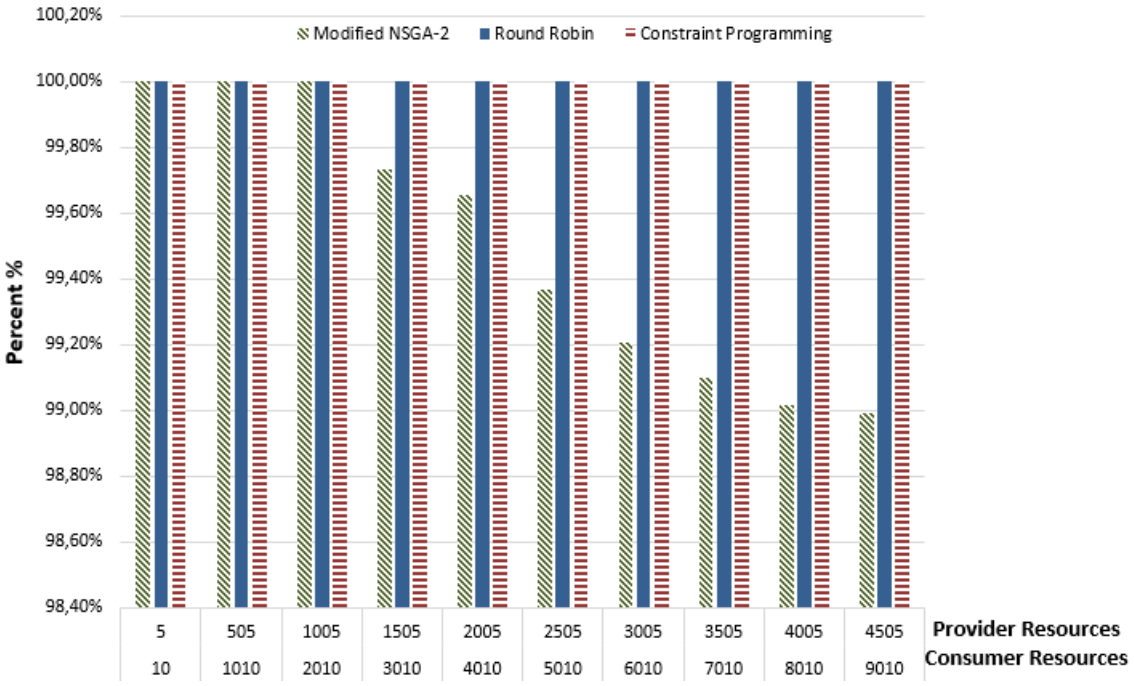

FIGURE 4.32: Modified NSGA II Provider Resource Used comparison with linear scenarios

4.3.4.2 Random scenario

In the random scenario, the provider resources and customer requests are drawn with random requested resource requirements and the infrastructure with random capacities. We use the same metrics to make the comparison: resolution time, cost and resource utilization or consumption.

Fig. 4.33 depicts the results of the resolution time (execution time) comparison and shows that the Round Robin becomes less efficiency than in the linear case. Constraint Programming and Round Robin have exponential resolution time and do not scale. The NSGA II resolution time is polynomial with problem size and has improved scalability especially if we recall that it uses multiple objectives while the other algorithms are single objective algorithms.

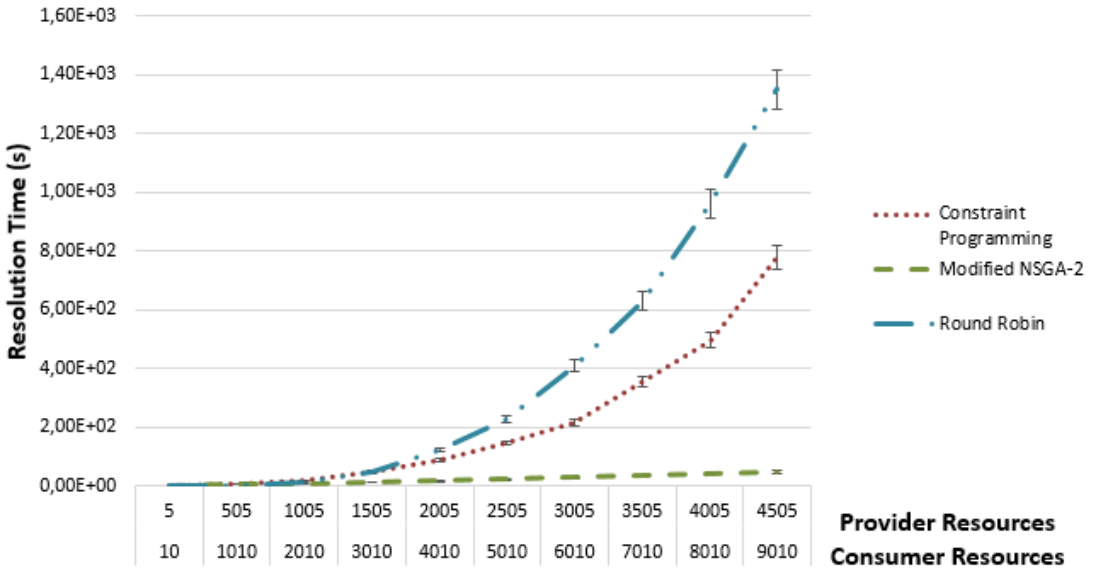

FIGURE 4.33: Resolution Time comparison with random scenarios

In terms of induced cost, the algorithms behave as shown in Fig. 4.34. The Round Robin algorithm incurs the highest cost and uses more resources than required while the Constraint Programming outperforms all algorithms. The NSGA II performance is nevertheless close to that of the Constraint Programming even if the gap increases with problem size (NSGA II costs around 13% more than Constraint Programming).

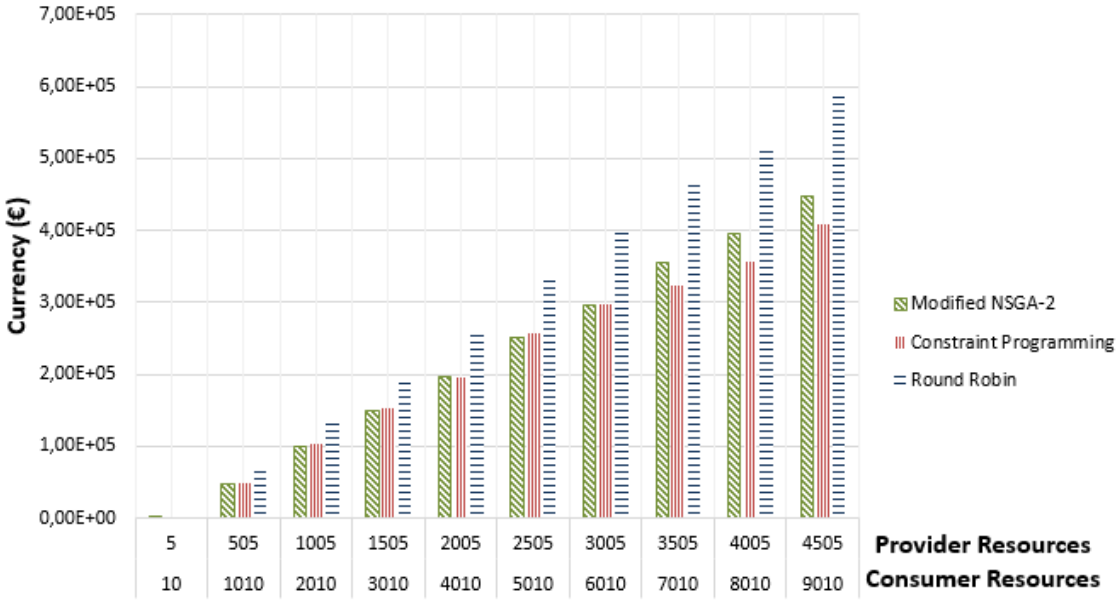

FIGURE 4.34: Global Cost comparison with random scenarios

Fig. 4.35 compares the percentage of request or resources hosted by the infrastructure, in essence this represents the acceptance rate or equivalently the rejection rate, as a function

of problem size (increasing requets and infrastructure sizes). Constraint Programming and round Robin accept all the requests for the evaluated scenarios and the simulation settings while NSGA II rejects up to 2% of the requests for large problem sizes (above 1500 provider resources and 3000 requested resources).

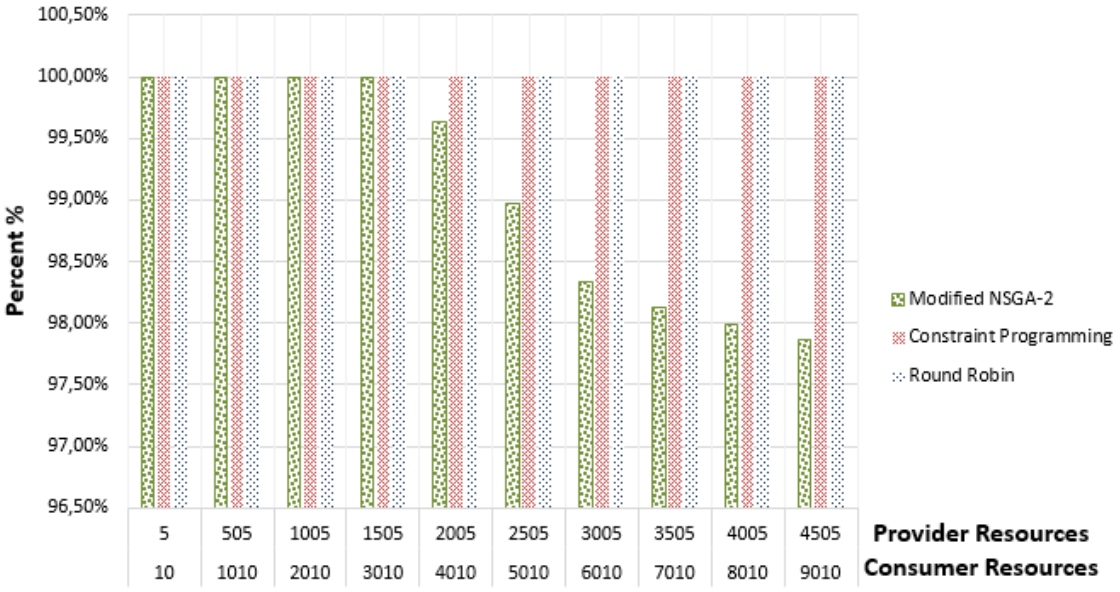

FIGURE 4.35: Customer Resource Hosted comparison with random scenarios

The amount of provider resources used to host the user requests is reported in Fig. 4.36 for the three algorithms. Constraint Programming uses the largest amount of resources to host the requests while Round Robin uses the smallest amount since it allocates resources in a systematic and cyclic fashion whereas the Constraint Programming will search over the entire space for solutions and consequently uses more resources. NSGA-II performance lies in between (actually closer to Constraint programming) but uses multiple objectives and criteria to make hosting decisions. NSGA II explores the space also more than Round Robin and can find more hosting solutions and ends up using the provider hypervisors efficiently.

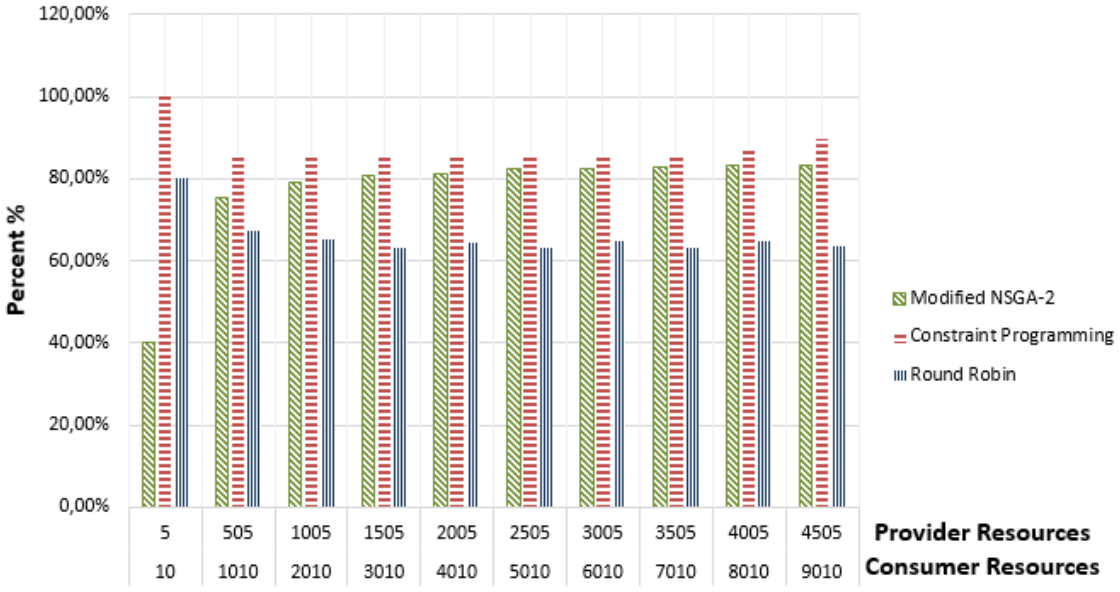

FIGURE 4.36: Provider Resource Used comparison with random scenarios

4.3.5 Extended performance comparison

The performance analysis and evaluation is extended by including the NGSA III contribution of the thesis with the previously evaluated algorithms. The scenarios used for the evaluation involve up to 800 servers (provider resources) and 1600 virtual machines (consumer requested resources). This setting is selected because it corresponds to typical infrastructure sizes most commonly managed by known and popular cloud computing platforms such as: OpenStack, OpenNebula, CloudStack, etc. Providers tend to cluster their infrastructures into domains and data centers and will not use the complete or entire cloud (in thousands of nodes) when searching for placement solutions. they will typically decompose the problem into reasonably sized domains or clusters. We consequently selected viable infrastructure sizes to pursue the performance evaluation. We also include the genetic algorithm approaches and variants in the assessment and performance comparison:

1. Constraint Programming
2. Round Robin
3. NSGA II
4. NSGA III
5. NSGA III with tabu search
6. NSGA with constraints solver

The parameters used by NSGA II and NSGA III are listed in Table 4.3.

Parameter	Value
populationSize	100
Number of evaluations	10000
sbx.rate	0.70
sbx.distributionIndex	15.00
pm.rate	0.20
pm.distributionIndex	15.00

TABLE 4.3: NSGA II and III settings

The performance results are reported in Figures 4.37, 4.38, 4.39, 4.40, 4.41 and 4.42 for respectively resolution (or execution) time, percentage of used infrastructure resources (or hosts/servers), amount of hosted consumer resources (VMs), number of constraints violations (when limits are exceeded), rejection rate and costs induced by each algorithm.

Figure 4.37 indicates that the Constraint Programming is the fastest algorithm capable of finding solutions in fractions of seconds since it uses constraints that considerable speed up execution or resolution time. The Round Robin method is slower because it cycles through the providers resources to achieve allocation of resources in a systematic fashion and hence the process takes more time. The genetic algorithms are even slower because they need to generate populations iteratively. The NSGA III with the tabu search, our advocated and proposed algorithm, improves performance compared to the NSGA III with Constraints Programming with execution times in the order of tens of seconds (12.5 sec for the largest problem size). This is a price penalty to pay for the integration of multi objectives, criteria and constraints in the optimization process compared to Constraint Programming and Round Robin for which the integration of such features is not only complicated but will also degrade execution time for very diminished return. The genetic algorithms are inherently build for the purpose and turn out to be globally more efficient and capable of trade-offs between the various performance metrics. This is illustrated in Figures 4.38, 4.39 and 4.40 where the genetic algorithms do not use all the infrastructure resources to serve the user requests, host all the requests and for the NSGA III with Constraints Programming and NSGA III with the Tabu search do not violate any constraint. This is confirmed by Figures 4.42 and 4.41 where the NSGA-III with the Tabu search achieves the best performance metrics trade off since its induced cost performance is close to the Constraints Programming cost (the smallest cost) and more importantly NSGA III has the smallest rejection rate (around 10% for NSGA III versus 40% to 90% for all other algorithms. Note also that the genetic algorithms embed multiple objectives in their optimization process and thus fulfill the requirements for the joint optimization of customers and providers objectives whose interests are taken into account in the model.

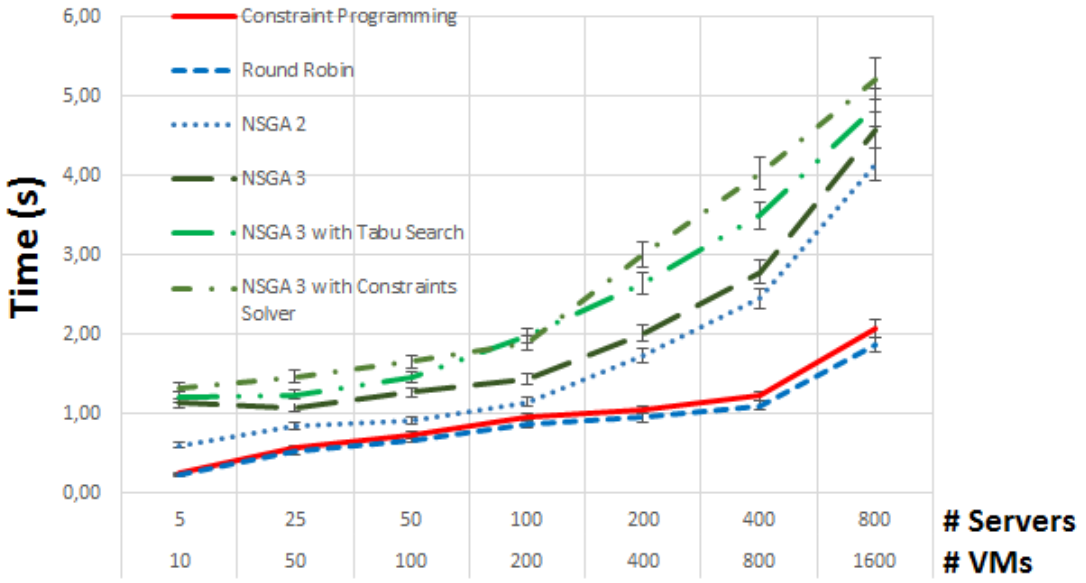

FIGURE 4.37: Comparison of means of Resolution Time with random scenarios

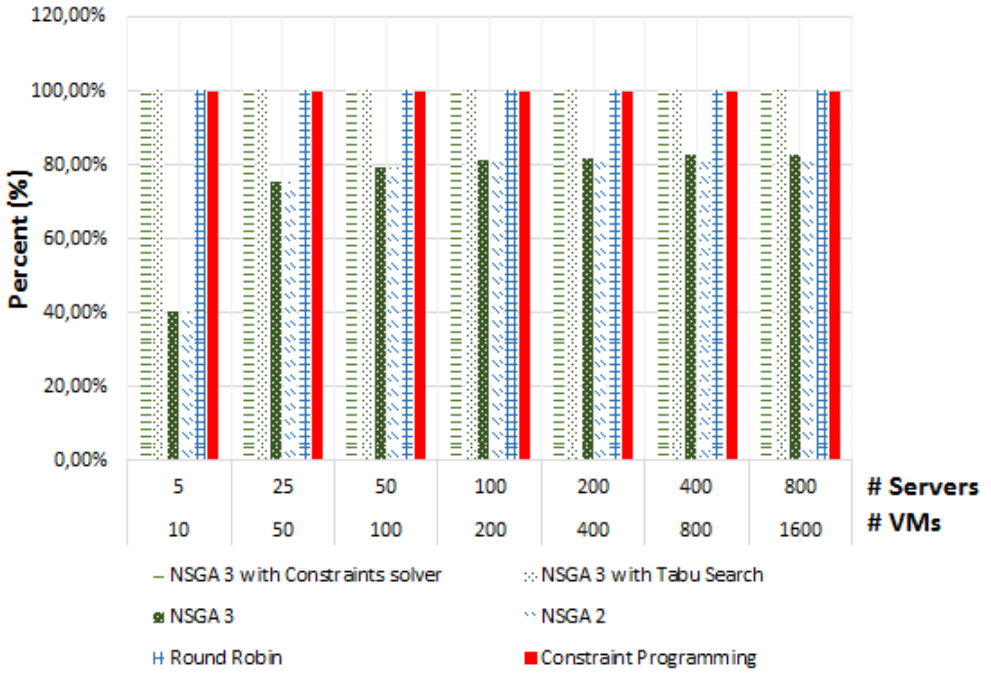

FIGURE 4.38: Servers Used with random Scenarios

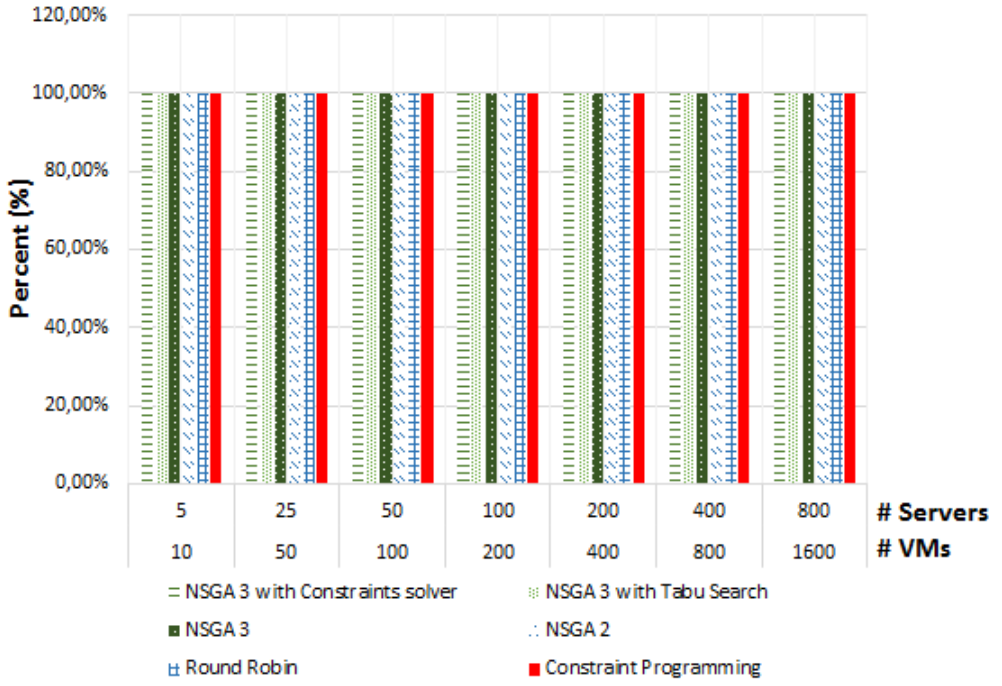

FIGURE 4.39: Consumer Resources Hosted with random scenarios

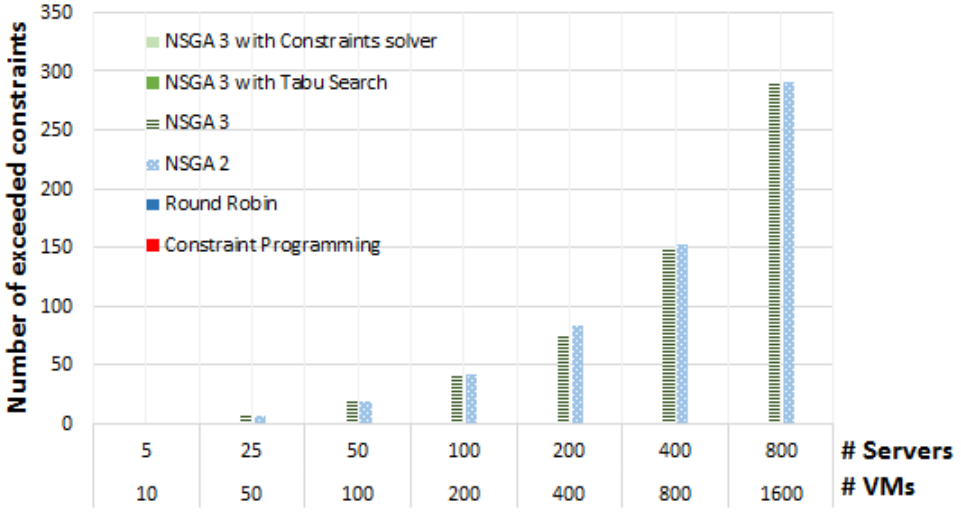

FIGURE 4.40: Comparison of means of Exceeding constraints with random scenarios

All these simulations have been performed hundreds of times over several randomly generated scenarios in order to validate and confirm our results. We can conclude from these performance evaluation results that for random scenarios (representing more faithfully real cloud platforms, Round Robin and Constraints Programming do not offer absolute cost effectiveness. The NSGA II and NSGA III algorithms are only based on stochastic processes and consequently cannot comply with constraints. The NSGA III enhanced with a Tabu search or a Constraint programming model can provide customized solutions in line with the requirements and interests of the consumers/users and the providers and are most adequate to address possibly conflicting goals of stakeholders.

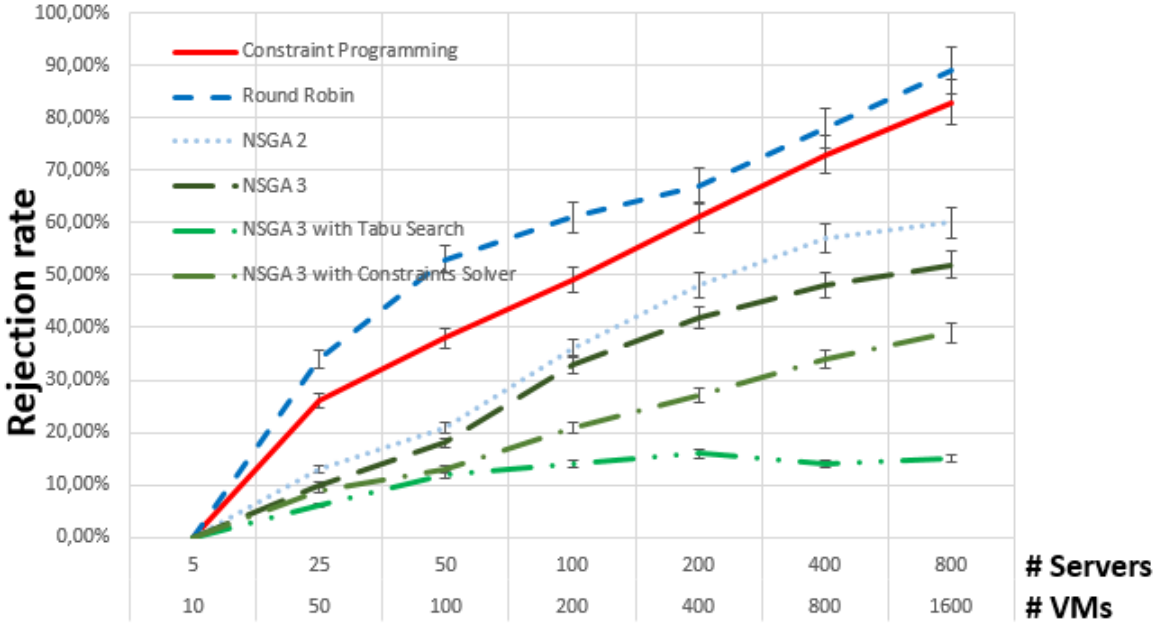

FIGURE 4.41: Rejection rate comparison with increasing problem size

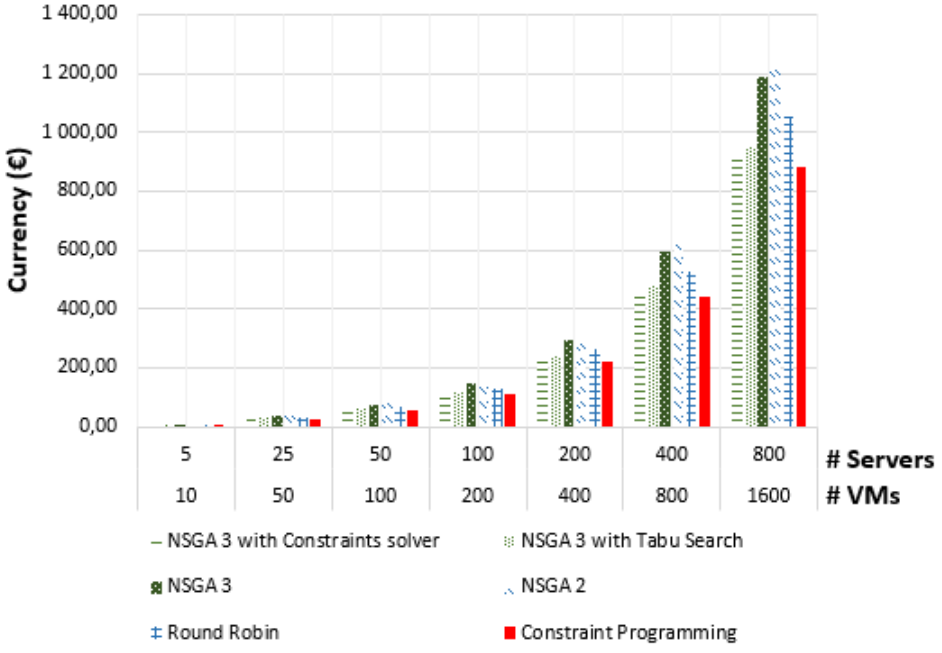

FIGURE 4.42: Comparison of means of costs generated by algorithms

4.3.5.1 Conclusion

To conclude, in the case of a linear scenario with provider and customer homogeneous resources, both Round Robin and constraint programming algorithms provide optimal solutions. However, the resolution time is exponential. Our modified NSGA II algorithm is outperformed by these more optimal solutions, depending on problem size.

In the case of random scenario with provider and customer heterogeneous resources, the Constraint Programming algorithm provides optimal solutions. However, it remains relatively slow for a dynamic and flexible system for practical cloud computing platforms. The Round Robin algorithm is not efficient in heterogeneous environments because resolution time is very high and generates a lot of overall costs compared to the optimal.

Our modified NSGA III algorithm with the Tabu search, one of the key contributions of this thesis, provides the best possible tradeoff of all criteria and can cater for both the user and providers interests.

In the tables below we summarize the advantages and disadvantages of existing algorithms :

TABLE 4.4: Performance Comparison of allocation algorithms

	Round Robin	Constraint Programming	NSGA-II	NSGA-III with Tabu Search	NSGA-III with Constraint Programming
Compliance with constraints	✗	✓	✗	✓	✓
Resource Scalability	✗	✗	✓	✓	✗
Integrity of customer requests	✗	✗	✗	✓	✓
Control over infrastructure	✗	✓	✗	✓	✓

Chapter 5

Conclusions and Perspectives

Abstract. This chapter summarizes the contributions of the thesis on the cloud resource allocation problem.We describe a number of foreseen perspectives beyond the thesis results and contributions. within a cloud computing platform.

We summarize the research work and contributions of our thesis on resource provisioning over IaaS cloud. This chapter outlines some perspectives for future investigations.

5.1 Conclusion et discussions

This thesis addresses cloud resource (services) allocation in relation to the IaaS layer of cloud services. We can summarize the thesis contributions as:

- We have reviewed the existing methods allowing to solve the resource allocation problem within a cloud computing platform. We have focused mainly on combinatorial optimization permutations theory and algorithms performing this type of action. We have identified different metrics to compare these allocation algorithms. The indicators are divided into three groups: technical indicators, business indicators and pricing indicators. We also presented resource allocation algorithms used by the industry (See Chapter 2).
- We have identified the key points of the users and providers business models to consider their affinity constraints and anti affinity between their resources (See Chapter 3).
- We proposed a representation model of a cloud platform equivalent to an extension to the model suggested by Open Cloud Computing Interfaces (OCCI). Our extension is more general description of the properties of compute and network resources. We have defined affinity and anti-affinity constraints on these resources in order to provide higher availability of the infrastructure resources without degradation (See Chapter 3).
- A genetic algorithm has been modified to speed up convergence to good solutions. We have improved variation operators (initialization and crossover) through initialization of the individuals respecting the capacity constraints (See Chapter 4).
- We have a feedback loop to check the resource allocation at regular time intervals, resulting in minimized reconfiguration costs.

5.2 Future Research Directions

Beyond the main contributions of this thesis, we plan to explore other research areas and improvements of our algorithm.

- Providing an optimal reconfiguration plan is a natural extension of this work and shall be a key question for future research. Such as how to migrate from the current resource allocation on the cloud computing platform to the new allocation provided with our algorithm.

- We have made use of a genetic algorithm to find an approximate solution taking into account the interests of each player. The disadvantage of genetic algorithms is that they are slow to converge. As a matter of fact, it would not be irrelevant to experiment more with different optimization algorithms. The advantage of genetic algorithms is they can be parallelized to explore the search space using multiple servers.

- We have chosen to use, in our work, vector and matrix representation for the resources and allocation solution. This method of information representation is low-level. A more generic representation at higher levels using Johnson graphs is a track to be taken into account to reduce complexity and improve execution time.

Appendix A

Thesis Publications

T. Ecarot, D. Zeghlache and C. brandily, "Consumer and provider based efficient cloud resource allocation" 2016 IEEE 8th International Conference on Cloud Computing Technology and Science (CloudCom), Luxembourg

Appendix B

HAWKS Allocation Tool

Abstract. This presents the way our resource allocation tool operates within a cloud computing platform. We are to stress on how the model of the platform can be best described and how an appropriate solution can be obtained through different algorithms.

The tool we have developed to allocate resources in a cloud computing platform supports a set of user requests and infrastructure provider representing the substrate parameters. We named this tool "HAWKS", standing for Heterogeneous Algorithmic Wide Knowledge Scheduler. HAWKS provides user requests allocation solution optimizing pecuniary costs and downtime costs. The implementation of the model and algorithms was done via the Java 8 language.

HAWKS provides two features. Feature One is an issue-pattern generator whereas Feature Two comes up with a model-based solver.

- **Pattern generator** is a process generating models describing provider infrastructure and user requests. We can generate problems with random parameters or through a linear distribution.
- **Optimization process** finds a resource allocation solution in a cloud computing platform. It is possible to use several resolution algorithms like Round Robin, Branch and Bound or genetic algorithms.
- **Showing results** is a process of recording results such as the fill rate of the hypervisor or the global cost of the cloud computing platform.

These two processes are detailed in the following sections.

B.1 Pattern generator

First program is a Jar file. When executed without defined parameters, help is displayed. The following command displays "help" while opening a user prompt for input:

```
$ java -jar GenAllocProb.jar
Generate file with problem
+------- HELP -------+
GenAllocProb.jar -file pathToFutureProblem
-file pathToFutureProblem
```

When user provides the name of the future model file, the program displays the main menu. From the main menu, you can save resources describing the provider's infrastructure, user requests and the business model of resources. Action Number Four can generate a completely random pattern without saving resources as before; Or it is possible to generate a model with a linear distribution with the previously saved resources.

The following command displays the main menu of pattern generation program:

```
$ java -jar GenAllocProb.jar -file test.hawks
Generate file with problem

Enter one of the following commands:
1 - Add Consumer Resource Model
2 - Add Provider Resource Model
3 - Add Consumer Resource Pricing
4 - Generate File Model
5 - Quit program
```

If we want to generate a pattern with a linear distribution we can add resources. We need to generate provider and consumer resources and define a business model. Consumer resource is described by an identifier (-1 if not allocated) of a server on which it is hosted and these attributes (CPU, RAM, Disk).

```
Please write consumer resource model like that: ID_SERVER;CPU;RAM;DISK
-1;2;4;100
```

The provider resources may be described in the same way. We must provide a value for each attribute by adding operating cost and usage cost for each hosted virtual machine.

```
Please write provider resource model like that: CPU;RAM;DISK;COPEX;USAGE_COST
8;16;500;1.230;0.89
```

We can detail the information business as profit for each virtual machine along with the downtime cost.

```
Please write provider resource pricing: XX.XX PRICING;DOWNTIME_COST
2.00;5.00
```

Next step is the generation of the pattern. It is saved as a file. The desired number of resources is required. User is prompted to specify path to the save folder. It is possible to enable a verbose mode with boolean variable. Verbose mode provides information on each stage of the optimization process or on each covered solution. Then, we choose the way the model will be generated. Choosing to generate a random pattern, for instance, would go as follows:

```
File Model Generation process...
Please enter the number of consumer resource:
10
Please enter the number of provider resource:
5
Please enter Results path of model:
/home/tecarot/Simu/Exp1/
Please enter verbose mode for Results: [true, false]
false
Please enter Characterization of the distribution [linear, random] :
random
```

A random distribution will generate a random pattern. Random generation using bounded variables. A linear distribution uses the saved resources to generate pattern. Following the previous generation from a random distribution, we get this file:

```
/home/tecarot/Simu/Exp1/;false
10;5
-1;9;266;57;10.35693061846847;1032.6556094089087
0;8;62;567;8.715393866438916;1532.2655873307108
2;27;387;445;16.292297215644588;201.19710935007242
4;1;424;969;19.782526074887713;969.6481617442731
1;6;53;255;17.096271340779133;531.1526237477988
3;13;334;483;9.033664716676899;237.77703403099054
1;30;237;815;5.203908188536724;178.91106419116164
4;18;260;615;1.4332063872638519;1617.192734013044
1;27;108;684;5.710043817763131;291.19628303436355
3;31;329;392;6.74485492060951;312.3982454746189
7;1135;7756;364.4387510581734;50.23627236686403
61;1299;4320;906.0071108984656;66.39641166357119
53;1801;2935;188.3761968979007;7.653800776244503
5;1397;935;753.3408099331476;98.74051278810396
59;1236;9889;213.1924165446822;16.593338724146555
```

With this file containing the pattern of the allocation problem, we can go to optimization process of cloud computing resources.

B.2 Optimization process

To solve the problem of allocating resources within a cloud computing platform, we have implemented a Java 8-based solver. Our solver can be used with different algorithms such as Round Robin (rr), Branch and Bound (bab) or Genetic Algorithm (ga).

```
$ java -jar AllocAlgo.jar -algo rr -file test.hawks
```

```
$ java -jar AllocAlgo.jar -algo bab -file test.hawks
```

```
$ java -jar AllocAlgo.jar -algo ga -file test.hawks
```

B.3 Showing results

While running allocation optimization algorithms, ten indicators have been collected. The indicators are as follows:

- **Resolution Time** The time for the algorithm to obtain a feasible solution.

- **Global Cost** The costs of the Cloud platform.
- **Downtime** The service degradation cost on user resources.
- **Percent Consumer Hosted** Percentage of client resources hosted.
- **Percent Provider Used** Percentage of provider resources used.
- **Fill Rate Average** The rate of average filling of the provider resources.
- **Fill Rate Median** The rate of median filling of the provider resources.
- **Fill Variance** The variance of the filling rate of provider resources.
- **Fill Rate Standard Deviation** The standard deviation of the filling rate of provider resources.
- **TurnOver** The sales turnover generated by the platform.

Sample results of Round Robin algorithm:

```
#Time   Costs   Downtime        PercentConsumerHosted   PercentProviderUsed
    FillRatesAverage        FillRatesMedian FillRatesVariance       FillRatesSD
        TurnOver
0.0;414.552470388795; 0.0; 1.0; 0.8; 0.45290448053611876; 0.526025874336121;
    0.05665000925118011; 0.23801262414246038 ; 122.35257381554007;
```

Appendix C

Résumé en Français

Abstract. Désormais, le Cloud Computing est couramment employé dans le secteur de l'industrie et des services. Ses utilisations sont nombreuses. Nous pouvons citer, en particulier, les services de stockage ou de sauvegarde en ligne qui offrent la possibilité d'avoir accès à ces données, en tout temps et en tout lieu, grâce à une interface distante. Ces facilités pour les utilisateurs existent, car les plate-formes de Cloud Computing sous-jacentes sont soumises à une optimisation de bout en bout en termes de gestion et de flexibilité.

Les utilisateurs de ces services sont de plus en plus exigeants au niveau de la disponibilité et de la sécurité de leurs données sur les plate-formes. Pour répondre à ces attentes, les fournisseurs de services se tournent vers une automatisation dans le placement des ressources au sein de leur plate-forme. Cette automatisation permettra une plus grande qualité de service grâce à une meilleure flexibilité de leurs infrastructures informatiques. Nous présentons dans ce mémoire, une nouvelle méthode de placement permettant de prendre en charge les intérêts du fournisseur, mais aussi ceux du consommateur de services. Ainsi, notre algorithme fournit des solutions de placement en fonction des coûts que la plate-forme va engendrer et des contraintes utilisateurs d'affinité et d'anti-affinité.

C.1 Introduction

La recherche opérationnelle dans le domaine de l'allocation de ressource informatique est ancienne et vaste. Les premières questions à ce sujet se sont posées sur l'allocation des bytes sur une plate-forme physique au sein d'un disque dur. Depuis, et suivant les évolutions du monde de l'informatique, les problèmes de placement de ressources ont migré vers des ressources représentant des tâches de calculs sur des grilles de serveurs. Ces méthodes d'allocation de tâches fonctionnent efficacement, mais elles ne peuvent pas intégrer les relations complexes qui peuvent exister entre les consommateurs et les fournisseurs dans un environnement métier. Un nouveau changement de paradigme a alors été opéré entre les plate-formes mainframe et les grilles de serveurs distribués pour tenter d'obtenir plus de flexibilité tout en gérant la complexité des contraintes métiers. Ce nouveau paradigme a pris le nom commercial de : Cloud Computing.

Le Cloud Computing est un nouveau paradigme, dans lequel diverses ressources telles que les infrastructures ou les composants logiciels sont fournis en tant que services, et qui connaît une grande popularité. Dans une infrastructure Cloud de type public, les fournisseurs louent des services ou de la capacité suivant des abonnements ou via un principe de paiement à l'utilisation à des clients qui consomment ces services. Les consommateurs de ces services interagissent avec la plate-forme Cloud grâce à une interface utilisateur ou une interface de programmation (API : Application Programming Interface) gérées par le fournisseur. Le Cloud Computing a eu de multiples définitions en fonction de l'évolution du paradigme et des personnes le décrivant comme les chercheurs ou l'industrie. Ainsi, la définition la plus couramment utilisée est celle du National Institute of Standards and Technology (Institut national des normes et des technologies, NIST). Ainsi, Peter Mell et Tim Grance définissent le Cloud Computing comme suit: "*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. This Cloud model is composed of five essential characteristics, three service models, and four deployment models.*" [2]. Cette définition peut trouver son équivalent en français ainsi : "*Le Cloud Computing est l'accès via un réseau de télécommunications, à la demande et en libre-service, à des ressources informatiques partagées configurables.*".

Les cinq caractéristiques [3] d'une plate-forme de Cloud Computing sont le provisionnement à la demande des ressources et des services où la Qualité de Services est garantie aux utilisateurs. Les ressources ou les services au sein d'une plate-forme peuvent être configurés automatiquement, orchestrés et consolidés pour présenter un point d'accès

unique rendu aux utilisateurs de la plate-forme. Une plate-forme de Cloud Computing doit être souple pour s'adapter aux diverses exigences d'un nombre potentiellement important d'utilisateurs.

Il existe plusieurs modèles de déploiement [4] tels que le Cloud privé qui fonctionne uniquement pour une seule organisation. Un nuage est appelé "Public Cloud" lorsque les services sont rendus sur un réseau qui est ouvert au public. Une fédération de Cloud partage l'infrastructure parmi plusieurs organisations d'une communauté spécifique avec des préoccupations communes (la sécurité, la conformité ou la juridiction). Enfin, un Cloud hybride est une composition de deux ou plus, d'infrastructures distinctes de Cloud (privé, communautaire ou public).

Les trois modèles de services [5] sont Software as a Service, SaaS, qui offre aux consommateurs des applications fonctionnant sur une infrastructure de Cloud. Le modèle de service suivant, Platform as a Service, PaaS, permet aux consommateurs de déployer grâce à l'infrastructure Cloud du fournisseur des applications créées en utilisant des langages de programmation, des bibliothèques, des services et des outils pris en charge par le fournisseur. Le dernier modèle est l'infrastructure en tant que service, IaaS. Il offre des ressources supplémentaires, comme une bibliothèque de machines virtuelles avec des images disques, brutes (bloc) et du stockage de fichiers, les pare-feu, les équilibreurs de charge, les adresses IP, les réseaux locaux virtuels (VLANs) tout ceci dans un paquetage vendu aux clients [6].

Les fournisseurs de plate-formes Cloud de type IaaS fournissent des ressources à la demande grâce à leurs bassins d'hyperviseur installés dans leurs centres de données. Mais les clients et les fournisseurs n'ont pas les mêmes objectifs sur la plate-forme. Pour la connectivité à ces plate-formes de Cloud, les clients peuvent utiliser Internet ou des réseaux porteurs (réseaux privés virtuels dédiés).

Dans ce mémoire et notre thèse, mise en œuvre et tests de recherche seront menés au sein d'une plate-forme de Cloud Computing de type IaaS privé. Cependant, le point crucial dans le paradigme Cloud Computing se trouve dans la ressource mise en commun avec d'autres acteurs ; et cette élasticité rapide n'est pas d'évidence. Malgré l'argument voulant qu'une plate-forme de Cloud Computing fonctionne automatiquement et sans exiger d'interaction humaine, la multiplicité de clients et des contraintes de la part des fournisseurs l'a rendue de plus en plus difficile à gouverner et il est devenu malaisé de contrôler l'infrastructure entière ainsi que la totalité des exigences, en temps réel, venant des clients. Les nouveaux défis du paradigme Cloud vont s'orienter vers la sécurité, la disponibilité et la gestion des ressources qui doivent être soigneusement considérées pour répondre à des exigences d'instanciation et d'hébergement ainsi qu'au

besoin d'évolutivité et de temps réel prenant en considération les intérêts de tous les utilisateurs.

Au sein d'une plate-forme de Cloud, les divers acteurs peuvent agir l'un sur l'autre à différents niveaux et sur différentes couches. Désormais, il y a deux acteurs majeurs pouvant interagir entre eux. Le client, qui consomme des ressources et des services, et le fournisseur, qui partage ou loue son infrastructure tout en fournissant l'entretien de celle-ci. Typiquement, un utilisateur placera des services ou des ressources sur demande sur différentes couches telles que le IaaS, PaaS ou SaaS.

En termes simples, dans une infrastructure de Cloud Computing, il y a deux acteurs génériques (Voir C.1). En premier lieu, il y a le fournisseur de services. En second lieu, le client de ces services. Malheureusement, les enjeux stratégiques pour chaque acteur sont différents. En outre, les consommateurs veulent des services de plus en plus complexes tout en payant moins. D'autre part, les fournisseurs doivent relever le défi de l'informatique écologique, tout en réduisant les frais d'exploitation et assurant une infrastructure plus gouvernable.

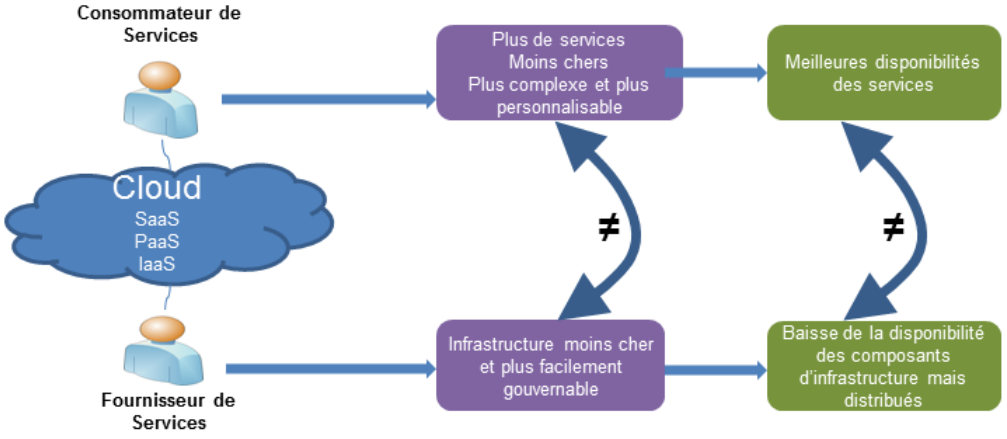

FIGURE C.1: Organisation du mémoire

Les besoins des clients qui consomment ces services étant en perpétuelle évolution, les consommateurs du Cloud Computing doivent fournir des efforts pour couvrir les coûts liés à la mise en conformité avec leurs nouveaux modèles d'affaires. À ce coût de mise en conformité s'ajoute le coût à l'indisponibilité potentielle de l'infrastructure distante. C'est pourquoi certains consommateurs de services Cloud se tournent vers une infrastructure privée ou hybride. Le terme "privé" renvoie au fait que la plate-forme de Cloud Computing n'est pas partagée et non à un éventuel avantage en termes de sécurité. En réalité, les modèles du fournisseur sont simplifiés et sont, bien entendu, insuffisants pour représenter la variété de relations entre le consommateur et le fournisseur de services dans différents contextes techniques et commerciaux.

C'est pourquoi les clients exigeants se tournent vers des solutions d'infrastructure de type Cloud privé où il est possible d'opérer une automatisation des processus d'évolution des services. L'automatisation est un processus essentiel au sein d'une plate-forme de Cloud Computing, car elle permet une livraison plus rapide des services aux clients, tout en évitant le risque d'erreurs humaines. Cette automatisation peut s'effectuer sur différents processus comme la vérification de la conformité du service rendu avec les SLAs (Service Level Agreement), le déploiement des correctifs, la montée en version d'un service ou l'allocation des ressources (provisionnement, déprovisionnement des ressources).

Aujourd'hui, afin de contrôler les demandes des clients et l'infrastructure dans son ensemble, une plate-forme de Cloud Computing utilise un module appelé "orchestrateur" ou "ordonnanceur" de ressources. Le module d'orchestration des ressources est l'élément central d'une plate-forme de Cloud Computing. En effet, il est conçu pour contrôler l'allocation des ressources et les services sur les couches de l'infrastructure du fournisseur. Par exemple, au sein de la couche Infrastructure as a Service (IaaS), un emplacement est assigné par l'orchestrateur sur l'hyperviseur pour chaque machine virtuelle.

Ce module d'ordonnancement ou d'orchestration aborde le problème de placement au sein des plate-formes de Cloud Computing. Ces ordonnanceurs sont conçus pour automatiser le placement des ressources ou des tâches. Si une erreur se produit sur la plate-forme de Cloud Computing, ils peuvent placer les ressources touchées par la panne sur un autre emplacement de l'infrastructure du fournisseur sans interaction humaine. Parfois, il peut être préférable d'effectuer des maintenances quotidiennes pendant que l'ordonnanceur apporte des améliorations dans le placement des ressources afin de préserver un groupe de ressource commune (Service identique ou machine virtuelle composant un même projet). Mais comment l'ordonnanceur peut-il choisir le centre de données et l'hyperviseur le plus approprié ? Et comment réagit-il en cas d'échec ?

Initialement, dans des environnements Cloud ou de grille de serveurs, les méthodes d'allocation de ressources consistaient à faire de la consolidation et de la répartition de charge ce qui est un héritage des méthodes d'allocation de bytes sur des disques physique. Ces dernières années, nous constatons l'introduction des workflows dans les méthodes de placement. Cependant, il est difficile d'appliquer ces principes sur des plate-formes de Cloud Computing du fait de l'hétérogénéité de celle-ci. Concernant les contraintes business et technique au sein d'une plate-forme de Cloud Computing, il y a eu des avancées et des améliorations dans l'implémentation d'algorithmes de résolution des problèmes d'allocation permettant leur intégration. En effet, des travaux ont été effectués afin d'intégrer les "workflows" entre les consommateurs et les fournisseurs de services. Un de ces travaux utilise un algorithme de résolution du problème du flot

de coûts minimum pour résoudre le problème d'allocation des ressources au sein d'une plate-forme de Cloud Computing.

Comme souvent en Cloud Computing, des travaux ont été repris dans le domaine du calcul haute performance. Ces travaux ont été réalisés pour réussir à placer de manière statique des tâches de calculs représentés sous forme de graphes sur des grilles de multiprocesseurs [7]. En se basant sur les travaux précédents, des auteurs ont fourni des méthodes exactes et heuristiques [8] pour allouer des réseaux virtuels à l'aide de sous-graphes au sein d'une architecture de prestation de services.

L'adoption grandissante des solutions de Cloud pour externaliser les couches informatiques "non-métier" pour les clients amène de nouveaux usages et de nouvelles opportunités sur les infrastructures porteuses. Dans l'optique d'intégrer ses nouvelles opportunités, les fournisseurs de services doivent mettre à disposition des solutions de plus en plus complexes afin de fournir un haut niveau de qualité de service tout en tentant de réduire les coûts, du fait de l'accroissement du nombre d'acteurs sur le marché de l'infrastructure en nuage qui induit une concurrence accrue. La gestion de ces nouveaux usages et de ces nouvelles opportunités est une problématique majeure pour les fournisseurs de services. En effet, les clients recherchent la plus haute disponibilité possible, ce qui est antinomique avec la forte concurrence sur le marché du Cloud Computing qui impose aux fournisseurs de services de réduire leurs coûts. Dès lors que critères et objectifs du problème sont connus, le problème peut être modélisé.

Nous définissons par la suite, les problèmes et les défis pour améliorer les ordonnanceurs de ressources sur les plate-formes de Cloud Computing ainsi que les objectifs que nous visons.

C.1.1 Problèmes et objectifs de recherche

Conformément à ce qui a été défini dans l'élaboration du sujet, ce mémoire de thèse se concentre sur l'allocation de ressources sur une plate-forme de type Infrastructure as a Service (IaaS) et décrit le développement d'un moteur d'optimisation permettant une allocation intelligente et dynamique des ressources clientes en mettant l'accent sur les principaux défis suivants :

- **Comment mesurer les performances d'un algorithme de placement des ressources au sein d'une plate-forme de Cloud Computing ?** Pour atteindre cet objectif, il faudra dans un premier temps dresser un état de l'art des

solutions décrivant l'état d'une infrastructure Cloud et ensuite identifier les indicateurs pertinents manquant capables de décrire l'état du système. Ces indicateurs permettront de définir un modèle généralisant les infrastructures d'un Cloud.

- **Comment représenter les ressources du fournisseur et celles demandées par les utilisateurs ?** Nous avons travaillé en adoptant une approche analytique où, dans un premier temps, il s'agit de résoudre le problème théorique en réalisant et en standardisant un modèle comprenant les différents composants d'une infrastructure en nuage de divers type (NaaS, IaaS ou PaaS) en intégrant les contraintes d'infrastructure comme la topologie, la politique et l'évolution de celle-ci. Par exemple, il s'agit de réduire le temps de latence entre les machines virtuelles ou bien le taux d'utilisation des liens d'interconnexion réseau entre les différentes baies et plaques et de pouvoir suivant l'évolution de l'infrastructure, replacer les ressources clientes tout en en optimisant la disponibilité.

- **Comment obtenir une solution de placement viable en un temps convenable ?** il s'agit de mettre en pratique notre modèle théorique et de réaliser un moteur d'optimisation capable d'implémenter les composants d'une infrastructure en nuage, les différentes contraintes qui y sont liées et de recevoir les demandes des utilisateurs et les informations des senseurs de l'infrastructure. Ce moteur d'optimisation devra être capable de communiquer avec divers types d'infrastructure en nuage et de répondre en un temps convenable (< 1 minute) pour un problème de taille moyenne (2000 machines virtuelles et 500 hyperviseurs).

Les verrous scientifiques que nous traitons sont principalement :

- Proposer une définition de modèle généralisant les infrastructures porteuses d'un Cloud.

- Identifier des indicateurs pertinents pour décrire l'état des infrastructures porteuses et définir le processus d'agrégation de l'information permettant d'alimenter le processus de décision.

- Résolution d'un problème d'optimisation NP-difficile permettant une allocation dynamique, efficace et intelligente des ressources clientes au sein d'un Cloud.

C.1.2 Contributions

Nous résumons pour plus de commodité nos contributions et nos réalisations décrites dans ce mémoire en respectant les défis identifiés à l'origine et nos objectifs de recherche définis.

La première contribution est un nouveau modèle de représentation des ressources se voulant générique et que nous appliquons à une plate-forme de Cloud Computing de type Infrastructure as a Service. Notre représentation des données est une extension de l'interface proposée par l'Open Cloud Computing Interface (OCCI) qui permet de décrire des ressources de type "infrastructure". Nous avons principalement travaillé sur la description des propriétés composant une ressource. Ainsi, nos ressources représentent des services d'infrastructure, mais pourraient aussi représenter des services plus complexes dans le domaine du Platform as a Service (PaaS) ou du Software as a Service (SaaS). Nous nous sommes attachés à la gestion des coûts sur une plate-forme et à la modélisation de l'interruption de service en corrélation avec la charge de travail des ressources du fournisseur hébergeant les demandes des utilisateurs.

La seconde contribution est l'identification et l'expression des contraintes d'affinité et d'anti-affinité entre les ressources du fournisseur et les ressources demandées par les utilisateurs. Ces contraintes d'affinité et d'anti-affinité permettent d'obtenir plus de flexibilité dans la gestion des coûts globaux de l'infrastructure et la qualité de service rendu aux utilisateurs. De plus, elles permettent de s'assurer que certaines ressources sensibles des utilisateurs soient allouées sur des ressources du fournisseur sécurisées. Ces contraintes s'expriment sous la forme de contraintes de placement des ressources virtuelles sur l'infrastructure du fournisseur.

La dernière contribution réside dans le développement d'un algorithme génétique capable de répondre et de converger en un temps convenable pour le domaine métier du placement de ressource au sein d'une plate-forme de Cloud Computing de type Infrastructure as a Service (< 30 secondes pour des problèmes de taille moyenne, 2000 machines virtuelles et 500 hyperviseurs). Il est aussi complexe de trouver le bon paramétrage d'ensemble des opérateurs, car cela dépend du problème traité. Il est aussi complexe de trouver le bon paramétrage d'ensemble des opérateurs, car il dépend du problème traité. Nous avons choisi d'utiliser un algorithme génétique, car celui-ci permet de trouver de multiples optima potentiellement proches de la solution idéale en un temps relativement court. De plus, ils peuvent être plus facilement parallélisables ce qui, dans le domaine du placement sur une plate-forme de Cloud Computing, peut devenir intéressant même si nous n'en traiterons pas dans ce mémoire. Notre algorithme génétique a été modifié afin d'améliorer le temps de convergence de la population vers une solution viable. Nous avons amélioré le processus d'initialisation de la population afin que chaque nouvel individu respecte la contrainte de capacité, ce qui a permis de faire converger la population plus rapidement vers une solution viable.

C.1.3 Organisation du mémoire

Les principaux chapitres de ce mémoire se structurent tels que décrits en Figure C.2.

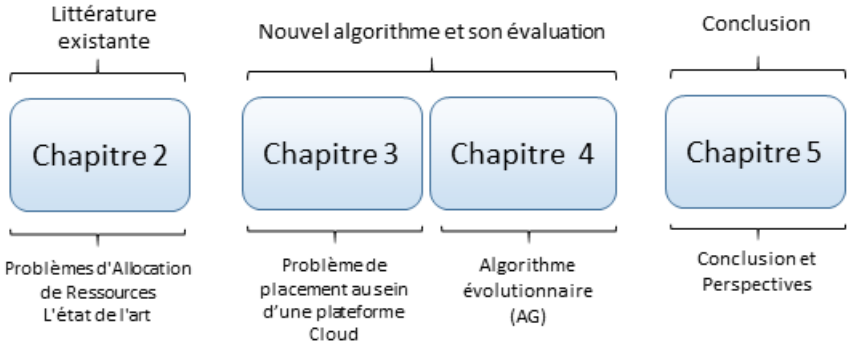

FIGURE C.2: Organisation du mémoire

Le chapitre 2 décrit les représentations et les méthodes existantes répertoriées dans la littérature du domaine de recherche sur le placement de ressource. Cet état de l'art est divisé en trois sections présentant les problèmes d'allocation de ressources multi-dimensions, les métriques de performance d'une plate-forme de Cloud Computing et une série d'algorithmes généralement utilisé pour résoudre ces problèmes d'allocation. La première section donne une description complète théorique de ce que sont les problèmes d'allocation multi-dimensions. Nous détaillons la notion de permutation en combinatoire qui permet de résoudre ces problèmes. La seconde section est une vue générale des différentes métriques permettant de mesurer la performance d'une plate-forme de Cloud Computing en vue de les comparer. La dernière section liste les algorithmes de placement de ressources dans un environnement Cloud de type Infrastructure as a Service.

Le chapitre 3 décrit un nouvel algorithme utilisant un modèle générique pour le placement des ressources Cloud. Le problème de placement des ressources Infrastructre as a Service est modélisé comme un problème de bin packing en multi-dimensions. Nous donnons une définition du problème sous la forme d'un problème uni-objectif et multi-objectifs avec leurs contraintes. Ce chapitre est résumé en français dans la section C.2.

Le Chapitre 4 expose en détails notre contribution à la résolution d'un problème de placement de ressource au sein d'une plate-forme de Cloud Computing grâce à un algorithme génétique. Nous décrivons le processus de décomposition des demandes exprimées sous la forme de graphe vers une représentation matricielle plus efficiente. Nous détaillons l'ensemble des opérateurs que nous avons modifié à notre algorithme génétique avec l'ensemble des paramètres que nous avons utilisés. Ce chapitre est résumé en français dans la section C.3. Afin d'améliorer le temps de convergence, nous définissons nos propres opérateurs d'initialisation et de croisement. Par la suite, nous effectuons une évaluation et une comparaison de notre algorithme avec les métriques que nous avons

définies dans la section 2.2 et avec un autre algorithme utilisé dans l'industrie (section 2.4).

Dans le dernier Chapitre 5 nous résumons nos contributions dans le domaine de l'allocation de ressource au sein d'une plate-forme de Cloud Computing puis nous décrivons les futurs axes de recherche sur le sujet et les perspectives relatives à ceux-ci. Notre conclusion se trouve résumée en français dans la section C.4.

C.2 Modélisation des problèmes de placement

Notre objectif est de fournir une solution d'allocation des ressources au sein d'une plate-forme de Cloud Computing en intégrant les intérêts des différents acteurs. Pour atteindre cet objectif, nous avons besoin de créer un nouveau modèle matriciel de représentation des ressources. Ce modèle devra prendre en compte l'ensemble des besoins des consommateurs de service et le substrat décrivant l'infrastructure du fournisseur. Nous fournirons une formulation mathématique du problème en intégrant les relations entre les ressources tout en respectant les intérêts de l'ensemble des acteurs d'une plate-forme de Cloud Computing sans failles ni comportement paradoxal.

Nous proposons un modèle suffisament générique décrivant des ressources de l'ensemble des couches d'une plate-forme de Cloud Computing comme Infrastructure as a Service (IaaS), Platform as a Service (PaaS) ou Software as a Service (SaaS). Mais nous nous concentrons principalement sur le placement de ressources au sein d'un IaaS. Nous étudierons la performance du placement au sein d'un IaaS qui permet d'allouer des ressources de calcul (machine virtuelle), de stockage ou de services réseaux sur des hyperviseurs. En outre, l'accent se porte sur la satisfaction des objectifs des consommateurs et du fournisseur tout en assurant la qualité de service requise pour les clients.

L'approche que nous adoptons pour trouver une solution de placement des ressources au sein d'un IaaS en fonction des objectifs des consommateurs et du fournisseur et des relations entre les ressources est une approche fondée sur une modélisation matricielle. Nous proposons d'utiliser des matrices pour décrire les demandes de service et les offres de service sous une forme compatible avec l'utilisation d'algorithmes évolutionnaires plus appropriés pour des problèmes avec des objectifs multiples en tenant compte de multiples contraintes.

La Figure C.3 décrit les processus internes qui composent généralement le module allouant les ressources au sein d'une plate-forme de Cloud Computing.

- **Entrées du système** représentent les requêtes des utilisateurs qui devront être allouées sur l'infrastructure du fournisseur ainsi que les données du système de monitoring du substrat collecté sur la plate-forme.
- **Analyseur de données** est un composant qui analyse et transforme les requêtes des utilisateurs composées de ressources et de relations en matrices. Les matrices sont utilisées par le modèle afin de représenter les demandes et les ressources disponibles au sein de l'infrastructure du fournisseur ;

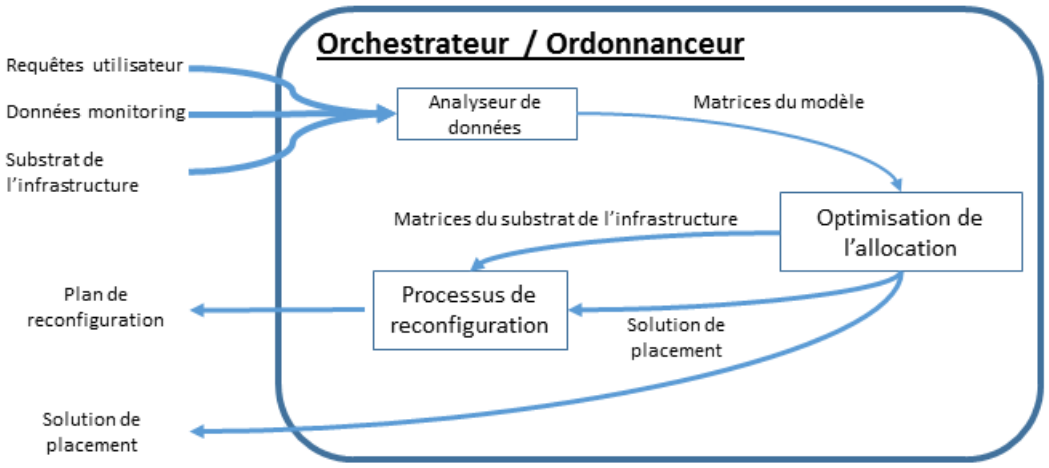

FIGURE C.3: Architecture du module d'allocation de ressource au sein d'une plate-forme Cloud

- **Matrices du modèle** sont composées de vecteurs et de matrices de nombre réel qui décrivent les requêtes utilisateurs et les ressources de l'infrastructure avec leurs relations et les contraintes associées ;
- **Processus d'optimisation de l'allocation** permet de résoudre le problème d'allocation des ressources au sein d'une plate-forme de Cloud Computing en utilisant des algorithmes exacts ou approchés grâce à des algorithmes heuristiques.
- **Solution de placement** est une solution de placement possible respectant les contraintes des clients et du fournisseur au sein d'une plate-forme de Cloud Computing mais potentiellement pas la plus optimale.
- **Plan de Reconfiguration** regroupe les actions nécessaires à mettre en oeuvre sur une plate-forme de Cloud Computing. Il comprend les actions d'instanciation et de provisionnement des ressources utilisateurs sur le substrat du fournisseur.

Pour ce mémoire, nous travaillons uniquement avec les ressources du consommateur que nous décrivons au travers de matrices représentant des requêtes utilisateurs et avec le substrat du fournisseur. Nous utilisons des matrices, car elles peuvent être facilement employées par des algorithmes évolutionnaires.

Nous avons choisi de fonder notre modèle sur le cas d'usage où un datacenter utilise une architecture de type Spine/Leaf comme décrit en Figure C.4.

Notre modèle prend en compte les contraintes des utilisateurs et du fournisseur avec pour objectif de trouver une solution optimale ou approchée minimisant les coûts,

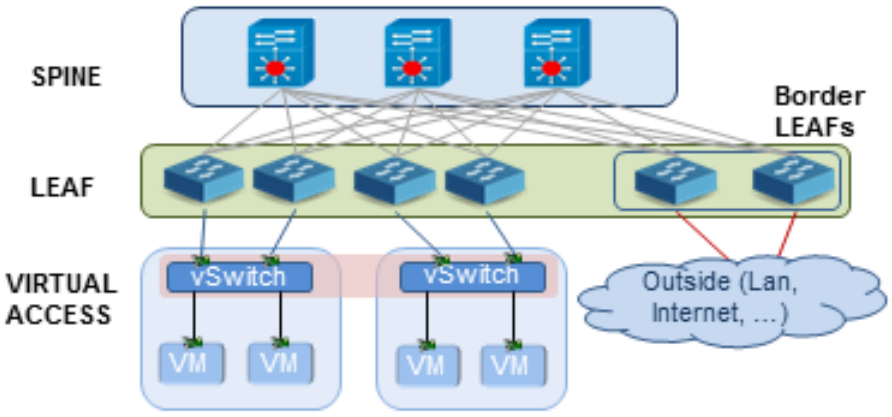

FIGURE C.4: Architecture d'un datacenter en Spine / Leaf

l'indisponibilité et la nécessité de migrer des ressources au sein de la plate-forme. Sans perte de généralité, nous affectons des poids égaux à ces objectifs qui peuvent être configurés différemment par le responsable de la plate-forme. Les objectifs à minimiser sont les suivants :

- **Coût d'exploitation et d'utilisation** détermine les coûts globaux d'une infrastructure en intégrant les coûts d'utilisation (la consommation de l'infrastructure);
- **Coût d'indisponibilité** représente les coûts dus à une indisponibilité des ressources du consommateur ;
- **Coût de migration** représente les coûts de migration des ressources des consommateurs de service définis par le plan de reconfiguration.

Nous avons identifié cinq relations et contraintes majeures permettant de décrire les besoins en termes d'affinité et d'anti-affinité entre les ressources exprimé par les consommateurs et le fournisseur :

- **Colocalisation au sein d'un même datacenter** : les utilisateurs ou les applications imposent une colocation de leurs ressources virtuelles au sein d'un même datacenter ;
- **Colocalisation au sein d'un même serveur** : cette contrainte impose une colocation des ressources virtuelles d'un utilisateur au sein d'un même hôte (serveur) ;
- **Séparation totale** : les ressources des consommateurs ont besoin d'être séparées à la fois sur des datacenters différent, mais aussi sur des serveurs différents ;

- **Séparation sur des datacenters différents** : c'est une séparation des ressources consommateurs sur différents datacenters ;
- **Séparation sur des serveurs différents** : les ressources consommateurs peuvent être hébergées dans le même datacenter mais elles ont besoin d'être séparées sur différents hôtes.

Il existe bien évidemment une contrainte permettant de vérifier la capacité des hôtes par rapport aux ressources des consommateurs.

- **Contrainte de capacité des ressources** C'est la contrainte utilisée afin de s'assurer que le total des ressources alloués ne dépasse pas la capacité de la ressource du fournisseur les hébergeants.

La Figure C.5 montre la composition d'u ne plateforme de Cloud Computing de type IaaS. Une plate-forme IaaS est constituée d'une couche de ressource matérielle et une autre des ressources virtuelles. La couche de matériel se compose de serveurs et de switchs reliés entre eux en utilisant l'architecture Spine/Leaf qui est utilisée dans la plupart des datacenters modernes. La couche virtuelle est constituée de machines virtuelles, des switchs virtuels et de réseaux sous la forme de VLAN. Ces ressources virtuelles sont attribuées sur des hyperviseurs de la couche sous-jacente, dans le cas présent l'infrastructure du fournisseur. Vous pouvez définir des relations entre les ressources comme des contraintes d'affinité ou d'anti-affinité lors de l'allocation.

Cette nouvelle modélisation des relations business permet de mettre en œuvre des services tout en gardant une vue d'ensemble du système. Cela conduira à faciliter l'automatisation de bout en bout sur l'infrastructure du fournisseur car il sera plus simple de gérer les requêtes complexes. Il sera plus simple de mettre en œuvre les services des utilisateurs sans défaillance et sans comportement paradoxal.

La modélisation, la représentation ou la description des requêtes utilisateurs et de l'infrastructure du fournisseur avec des ressources au travers de matrices est suffisamment générique pour être réutilisée avec d'autres types de services. Le modèle est obtenu en définissant les ressources des utilisateurs et des fournisseurs comme spécifié dans le paragraphe suivant.

La Figure C.6 décrit les attributs des ressources composant un datacenter. Nous détaillons les attributs des serveurs, des liaisons réseaux et des switchs. Ainsi, un serveur est modélisé au travers de la puissance et du nombre de CPU, de la capacité de la RAM, du disque et de la bande passante maximum de la carte réseau. Nous associons au serveur un coût d'exploitation. Les liaisons réseaux sont décrites avec une bande passante et

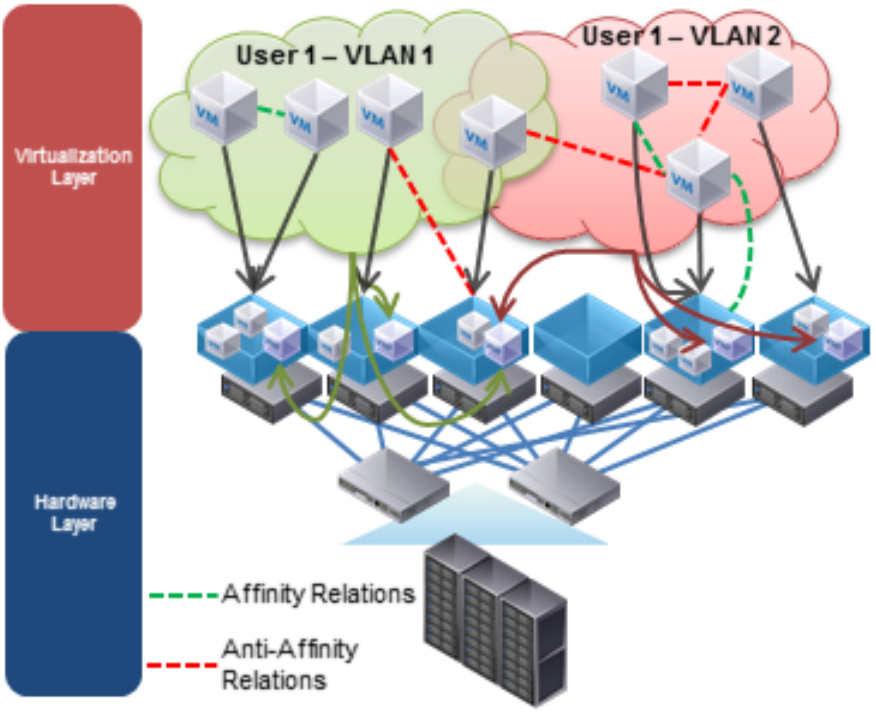

FIGURE C.5: Allocation des ressources virtuelles au sein d'une plateforme de Cloud Computing

un débit. Les switchs disposent d'attributs comme la capacité de la RAM, de la bande passante et du taux de renvoi.

La Figure C.7 montre les attributs des ressources composant une requête d'un utilisateur. Une requête est une demande d'un utilisateur contenant des machines virtuelles, des switchs virtuels et des relations entre ces ressources. Une machine virtuelle disposent des mêmes attributs qu'un serveur. Les relations entre les ressources représentent des contraintes d'affinité ou d'anti-affinité ou une description d'un réseu de type VLAN. Un switch virtuel demande les mêmes attributs qu'un switch physique.

Nos travaux se focalisent principalement sur l'allocation des machines virtuelles sur des serveurs afin de présenter une comparaison des performances de différents algorithmes de placement (recherche tabou, programmation par contraintes, algorithmes évolutionnaires)

Le nombre de datacenters du fournisseur est noté g, le nombre de serveurs défini est m, le nombre total de ressources demandées par les utilisateurs est n et le nombre d'attributs des ressources est représenté par h. Dans notre modèle, le nombre d'attributs des ressources fournisseur et celui des ressources des utilisateurs sont identiques. La correspondance entre les ressources s'effectue entre les mêmes attributs en respectant les demandes des utilisateurs.

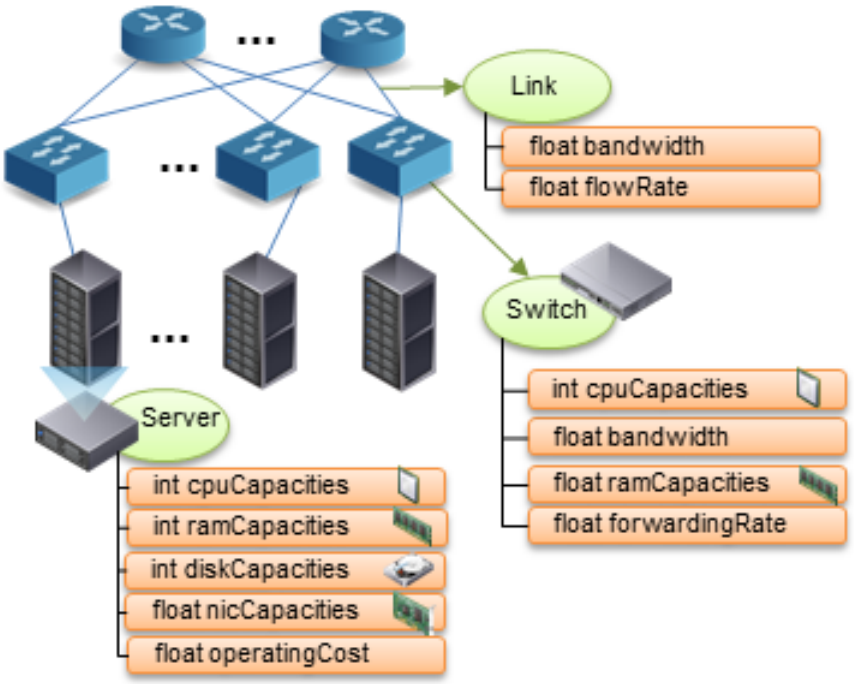

FIGURE C.6: Attributs des Ressources d'Infrastructure

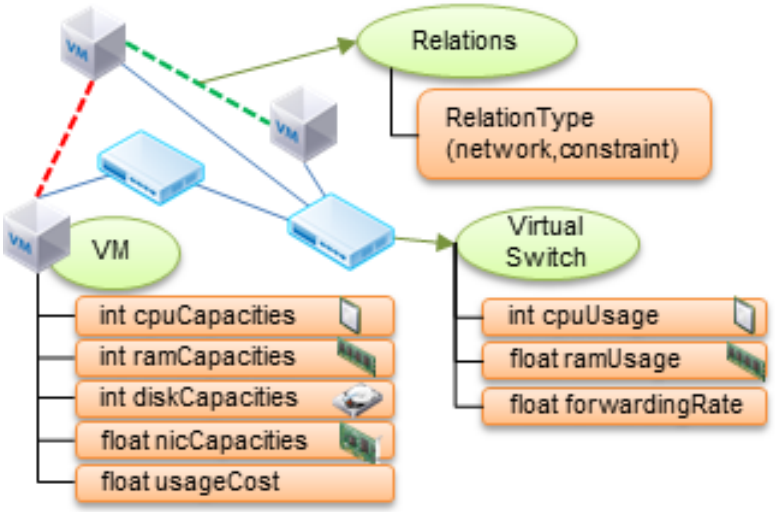

FIGURE C.7: Attributs des Ressources Virtuelles

Les capacités d'une ressource (capacité maximum disponible par serveur ou liaison) sont représentées avec deux matrices. La première matrice [Eq. C.1] représente la capacité de chaque attribut d'une ressource du fournisseur.

$$\begin{array}{cc} P_{jl} & 1 \leq j \leq m, 1 \leq l \leq h \\ P_{jl} \in \mathbb{R}^{+} & m \times h \end{array} \tag{C.1}$$

La seconde matrice [Eq.C.2] représente la capacité de chaque attribut d'une ressource de l'utilisateur.

$$\begin{array}{cc} C_{kl} & 1 \leq k \leq n, 1 \leq l \leq h' \\ C_{kl} \in \mathbb{R}^{+} & n \times h' \end{array} \tag{C.2}$$

Le nombre d'attributs pour les ressources du fournisseur et des utilisateurs sont les mêmes pour les deux types de ressources ($h = h'$).

De plus, nous définissons un facteur sur les attributs qui correspond au ratio entre les capacités des ressources matérielles et les capacités des ressources virtuelles. Ces facteurs sont décrits en utilisant une matrice [Eq.C.3] de taille $m \times h$.

$$\begin{array}{cc} F_{jl} & 1 \leq j \leq m, 1 \leq l \leq h \\ F_{jl} \in \mathbb{R}^{+} & m \times h \end{array} \tag{C.3}$$

Nous notons X_{ijk}, une variable booléenne qui est vraie quand une ressource consommateur k est hébergée pour le serveur du fournisseur j dans un datacenter i. Nous définissons une contrainte qui s'assure que la capacité d'une ressource du fournisseur [Eq.C.10]. Cette contrainte inclut la consommation des ressources virtuelles sur les ressources matérielles du fournisseur en incluant le facteur sur chaque attribut défini précédemment.

$$\sum_{k=1}^{n} C_{kl} \times X_{ijk} \leq P_{jl} F_{jl}, \forall l = 1...h, \forall i = 1...g \tag{C.10}$$

Afin de représenter les coûts d'exploitation des ressources du fournisseur, nous introduisons un vecteur [Eq.C.4] dont les éléments sont des coûts qui représentent les coûts directs associés à l'utilisation d'un serveur : la puissance, l'espace au sol, le stockage et les opérations informatiques courantes de gestion.

$$E_j \quad 1 \leq j \leq m, \quad E_j \in \mathbb{R}^{+} \tag{C.4}$$

Au-delà des coûts d'exploitation, chaque ressource du fournisseur dispose d'un coût d'utilisation qui sera appliqué pour chaque ressource du consommateur hébergée. Nous

utilisons un vecteur [Eq. C.5] pour représenter ces coûts d'utilisation.

$$U_j \quad 1 \leq j \leq m, \quad U_j \in \mathbb{R}^+ \tag{C.5}$$

Chaque attribut sur chaque ressource du fournisseur a une charge maximum avant que la qualité de service se détériore ou soit dégradée pour les ressources virtuelles des clients. Nous notons L_{jl}^M la charge maximale avant dégradation et L_{jl} la charge courante du serveur j pour l'attribut l. Ces serveurs ont une qualité de service courante et maximum définis respectivement par Q_{jl} and Q_{jl}^M. Le niveau de service garanti est noté C_k^Q et s'il n'est pas respecté le fournisseur paie une pénalité d'indisponibilité définis comme un coût C_k^U. Le prestataire de services doit garantir qu'il n'y aura pas d'interruption de service. Nous définissons ces charges et la qualité de services à l'aide de matrices [Eq.C.6]:

$$\begin{array}{ll} L_{jl}^M & 0 \leq L_{jl}^M < 1 \\ L_{jl} & 0 \leq L_{jl} < 1 \\ Q_{jl}^M & 0 \leq Q_{jl}^M < 1 \\ Q_{jl} & 0 \leq Q_{jl} < 1 \end{array} \tag{C.6}$$

Toutes les variables et les constantes du modèle utilisé sont listées dans le tableau C.1.

Le modèle est amélioré avec toutes les relations nécessaires décrivant des liens entre les ressources et des contraintes exprimées grâce à des inégalités et des égalités.

La contrainte de **Colocalisation au sein d'un même datacenter** est défini par [Eq.C.11] et s'assure que les ressources consommateurs sont hébergées au sein d'un même datacenter.

$$\sum_{i=1}^{g} \prod_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{C.11}$$

La contrainte de **Colocalisation au sein d'un même serveur** vérifie si la somme des produits des ressources consommateurs allouées est strictement égale à un [Eq.C.12]. Elle garantit la colocalisation au sein d'un même serveur :

$$\sum_{i=1}^{g} \sum_{j=1}^{m} \prod_{k=1}^{n} X_{ijk} = 1 \tag{C.12}$$

La contrainte de **séparation totale** est exprimée grâce à la somme des ressources consommateurs allouées sur les datacenters et les serveurs. Cette contrainte doit être égale à un comme indiqué par [Eq. C.13] :

TABLE C.1: Variables du modèle d'allocation de ressource Cloud

	Substrat du fournisseur et les ressources demandées
G	Ensemble de datacenter disponibles $G = \{1, ..., g\}$
M	Ensemble de serveur disponibles $M = \{1, ..., m\}$
N	Ensemble des ressources demandées $N = \{1, ..., n\}$
H	Ensemble des attributs disponibles pour chaque ressource $H = \{1, ..., h\}$
	Matrices des capacités
P_{jl}	Capacités de la ressource du fournisseur j pour l'attribut l
C_{kl}	Capacités de la ressource consommateur k pour l'attribut l
F_{jl}	Facteur de capacité pour l'attribut l sur la ressource du fournisseur j
	Modèle d'allocation
X_{ijk}	Variable booléenne indiquant que la ressource du consommateur k est assignée à la ressource du fournisseur j dans le datacenter i
	Vecteurs et matrices pour la qualité de service
L_{jl}	Charge de l'attribut l sur la ressource du fournisseur j
L_{jl}^{M}	Charge maximum d'un attribut l sur la ressource du fournisseur j
Q_{jl}^{M}	Qualité de service maximum d'un attribut l sur la ressource du fournisseur j
Q_{jl}	Qualité de service d'un attribut l sur la ressource du fournisseur j
C_{k}^{Q}	Qualité de service garantie par le fournisseur pour une ressource consommateur k
	Vecteurs des coûts
E_j	Coût d'exploitation de la ressource du fournisseur j
U_j	Coût d'utilisation pour chaque ressource consommateur hébergée sur la ressource du fournisseur j
C_{k}^{U}	Coût d'indisponibilité pour la ressource consommateur k
M_k	Coût de migration de la ressource consommateur k

$$\sum_{k=1}^{n} X_{ijk} = 1, \forall i = 1...g, \forall j = 1...m \tag{C.13}$$

La contrainte de **séparation sur différents datacenters** est sensiblement identique à la précédente, sauf que la somme s'effectue sur les centres de données et les ressources demandées et elle est formulée ainsi [Eq. C.15] :

$$\sum_{i=1}^{g} \sum_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{C.15}$$

La contrainte de **séparation sur différents serveurs** implique une somme sur les centres de données, les serveurs et les ressources demandées et elle est exprimée par [Eq. C.14].

$$\sum_{i=1}^{g} \sum_{j=1}^{m} \sum_{k=1}^{n} X_{ijk} = 1 \tag{C.14}$$

Toutes ces relations et les contraintes sont mises en œuvre dans notre algorithme qui sera utilisé pour fournir des solutions au problème de l'allocation des ressources au sein d'une plate-forme de Cloud Computing abordées dans ce travail.

Nous résolvons le problème d'allocation des ressources en fonction des coûts d'exploitation et d'utilisation, les relations d'affinité / anti-affinité entre les ressources et la qualité de service requise sur les ressources allouées sur la plate-forme de Cloud Computing . Notre approche prend en compte de multiples contraintes, des relations et des objectifs, mais les multiples objectifs sont agrégés au sein d'une fonction d'objectif global tel que décrit par la suite.

Notre problème d'optimisation est similaire au problème de "bin packing" multi-dimension et au problème de sac à dos qui ont été montrés comme étant NP-Difficile, notre problème est effectivement NP-Difficile. Le problème de bin packing multi-dimension est un problème d'allocation de vecteur avec d dimension et il a été prouvé comme étant NP-Difficile dans [105].

Comme notre objectif est de minimiser les coûts, les interruptions de service et la taille du plan de reconfiguration, nous introduisons plusieurs fonctions objectives comme les coûts globaux générés par la plate-forme de Cloud Computing [Eq. C.7], la qualité de service des ressources [Eq. C.8] et la taille approximative du plan de reconfiguration [Eq.C.9]. Notre problème d'allocation de ressource au sein d'une plate-forme de Cloud Computing peut être résumé en regroupant toutes les fonctions objectives avec toutes les contraintes et les conditions grâce à l'ensemble des équations suivantes :

$$min\ Z = min[\sum_{j=1}^{m} E_j \times X_{ijk} + \sum_{k=1}^{n} U_j \times X_{ijk}, {}^{\forall i=1...g} \tag{C.7}$$

$$+ \sum_{k=1}^{n} C_k^U (\lfloor \frac{Q_{jl}}{C_k^Q} \rfloor) \times X_{ijk}, {}^{\forall i=1...g}_{\forall j=1...m}{}_{\forall l=1...h} \tag{C.8}$$

$$+ \sum_{k=1}^{n} (X_{ijk}^t - X_{ijk}^{t+1}) M_k, {}^{\forall i=1...g}_{\forall j=1...m}] \tag{C.9}$$

Subject To en fonction des demandes utilisateurs:

$$\sum_{k=1}^{n} C_{kl} \times X_{ijk} \leq P_{jl} F_{jl}, \forall l = 1...h, \forall i = 1...g \tag{C.10}$$

$$\sum_{i=1}^{g} \prod_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{C.11}$$

$$\sum_{i=1}^{g}\sum_{j=1}^{m}\prod_{k=1}^{n} X_{ijk} = 1 \tag{C.12}$$

$$\sum_{k=1}^{n} X_{ijk} = 1, \forall i = 1...g, \forall j = 1...m \tag{C.13}$$

$$\sum_{i=1}^{g}\sum_{j=1}^{m}\sum_{k=1}^{n} X_{ijk} = 1 \tag{C.14}$$

$$\sum_{i=1}^{g}\sum_{k=1}^{n} X_{ijk} = 1, \forall j = 1...m \tag{C.15}$$

Les fonctions objectifs et les paramètres sont tous convertis à un coût monétaire équivalent afin qu'ils puissent être regroupés en une fonction "objectif global" Nous détaillons toutes les fonctions objectives. Nous rassemblons toutes les fonctions objectives en convertissant ces différentes valeurs en un coût pécunier. La première fonction est le calcul du coût global d'une plateforme donnée par [Eq.C.7]. Cette fonction prend en compte les coûts d'exploitation et d'utilisation des ressources. Pour chaque ressource du fournisseur, E est le coût d'exploitation d'un hyperviseur j. Pour chaque ressource du consommateur, la variable U représente le coût de consommation d'une ressource k.

$$\sum_{j=1}^{m} E_j \times X_{ijk} + \sum_{k=1}^{n} U_j \times X_{ijk}, {}^{\forall i=1...g} \tag{C.7}$$

Le second objectif [Eq.C.8] représente les coûts d'une interruption de service ou la pénalité en cas de baisse du niveau de service demandé par l'utilisateur sur les ressources du fournisseur hébergeant les services. Sur la base des évaluations de la performance, les auteurs [110] [111] prouvent, par une démarche empirique, que la qualité du service des ressources consommateur diminue de façon exponentielle avec l'augmentation de la charge de travail des ressources du fournisseur. Le coût total d'indisponibilité est la somme de chaque coût d'indisponibilité C_k^U de la ressource consommateur k quand le niveau de service C_k^Q garanti n'est pas respecté.

$$\sum_{k=1}^{n} C_k^U(\lfloor\frac{Q_{jl}}{C_k^Q}\rfloor) \times X_{ijk}, {}^{\substack{\forall i=1...g \\ \forall j=1...m \\ \forall l=1...h}} \tag{C.8}$$

Nous avons besoin de calculer la qualité de service [Eq. C.16] pour chaque attribut l pour chaque ressource fournisseur j. Nous notons cette variable Q_{jl}. La qualité de service est en fonction de la charge sur les hyperviseurs et elle se calcule grâce à une fonction affine par morceaux où chaque ressource du fournisseur a une charge maximum avant détérioration L_{jl}^{M}. Cela se traduit par [Eq.C.16] :

$$Q_{jl} = \begin{cases} Q_{jl}^{M} & if\ L_{jl} \leq L_{jl}^{M} \\ Q_{jl}^{M} e^{\frac{L_{jl}^{M} - L_{jl}}{1 - L_{jl}}} & if\ L_{jl} > L_{jl}^{M} \end{cases} \tag{C.16}$$

La charge de travail sur la ressource du fournisseur j pour un attribut spécifique l est calculée [Eq.C.17] comme la somme des valeurs des attributs des ressources hébergés sur les serveurs divisée par la capacité des ressources du fournisseur.

$$L_{jl} = \frac{\sum_{k=1}^{n} C_{kl} \times X_{ijk}}{P_{jl}} \begin{matrix} \forall i=1...g \\ \forall j=1...m \\ \forall l=1...h \end{matrix} \tag{C.17}$$

Le dernier objectif est la somme des coûts entraînés par l'exécution du plan de reconfiguration [Eq.C.9]. La taille du plan de reconfiguration est une estimation fondée sur l'allocation courante X_{ijk}^{t} en utilisant la nouvelle solution de placement X_{ijk}^{t+1} fournie par le processus d'optimisation.

$$\sum_{k=1}^{n} (X_{ijk}^{t} - X_{ijk}^{t+1}) M_k, \begin{matrix} \forall i=1...g \\ \forall j=1...m \end{matrix} \tag{C.9}$$

Bien entendu, nous pouvons résoudre le problème comme un programme linéaire en utilisant le modèle proposé pour réduire au minimum les coûts grâce aux équations C.7 à C.9 en prenant en compte les contraintes spécifiées dans les équations C.10 à C.15.

C.3 Algorithme de placement des ressources IaaS

L'utilisation d'une représentation matricielle pour décrire les coûts d'exploitation et d'utilisation ainsi que les contraintes de performance, nous a conduit naturellement vers l'utilisation d'algorithmes évolutionnaires qui sont généralement mieux adaptés à la résolution de problèmes d'optimisation avec de multiples objectifs et contraintes. Nous présentons, ainsi, un modèle de décomposition des requêtes utilisateurs vers le modèle matriciel et une description des besoins de chaque acteur de la plate-forme.

Dans un premier temps, nous savons que résoudre le problème d'allocation des ressources sur une plate-forme de Cloud Computing grâce à un solveur de résolution de contrainte comme le solveur CPLEX, entraîne un temps de résolution exponentielle, ce qui implique que les solutions ne passent pas à l'échelle en fonction de la taille du problème. De plus, ces méthodes ne sont pas adaptées pour les problèmes multi-objectifs et multi-critères. Afin de fournir un algorithme permettant de passer une mise à l'échelle et de fournir des solutions dans des délais raisonnables, nous avons recours à l'algorithme NSGA II [108] qui est un algorithme génétique permettant de résoudre notre problème d'allocation des ressources au sein d'une plate-forme de Cloud Computing que nous avons adressé dans ce mémoire. NSGA II est un algorithme génétique populaire fondé sur la non-domination de Pareto pour l'optimisation multi-objectifs et il constitue donc un choix approprié pour le problème que nous considérons ici.

La complexité en temps de l'algorithme NSGA II est $O(MN^2)$ où M est le nombre d'objectifs et N est la taille de la population. Afin d'accélérer le temps d'exécution de l'algorithme et réduire l'impact de cette complexité, nous modifions l'algorithme comme décrit dans la figure Fig.C.8 et nous développons une méthode spécifique d'initialisation (Algorithme 4) fondé sur les observations des résultats du tirage au sort des gènes composant les solutions en fonction du taux de succès permettant de trouver des solutions viables et acceptables, c'est-à-dire celles qui répondent favorablement aux contraintes de capacité. Nous utilisons une probabilité dérivée de ces observations pour améliorer les futurs tirages et permettre d'accélérer l'exploration vers de bonnes solutions. En substance, nous exploitons les observations passées pour affiner les futures explorations.

La Figure C.8 montre la structure générale de notre algorithme NSGA II modifié. L'initialisation de la population se fonde sur un tirage au sort aléatoire. L'opérateur de sélection est un opérateur de sélection pour tournoi binaire utilisant la domination de Pareto. Le principe de la sélection par tournoi augmente les chances pour les individus de piètre qualité de participer à l'amélioration de la population. L'opérateur de croisement utilisé est un opérateur de croisement binaire simulé du nom de Simulated binary crossover [1] (SBX). L'opérateur de croisement SBX reproduit les mécanismes du croisement standard à un point utilisé lorsque les variables objets sont représentés sous la forme de chaînes binaires. Nous utilisons l'opérateur de mutation polynomiale (PM). Il est fondé sur l'échange de bit au sein d'un gène. Semblable à SBX, PM favorise les descendants les plus proches des parents.

Le choix de la population initiale d'individus conditionne fortement la rapidité de l'algorithme. Un individu est constitué d'un chromosome et est une solution de placement des ressources. Un chromosome correspond à une variable booléenne X_{jk} indiquant que la ressource demandée k est assigné à la ressource du fournisseur j. Le chromosome est

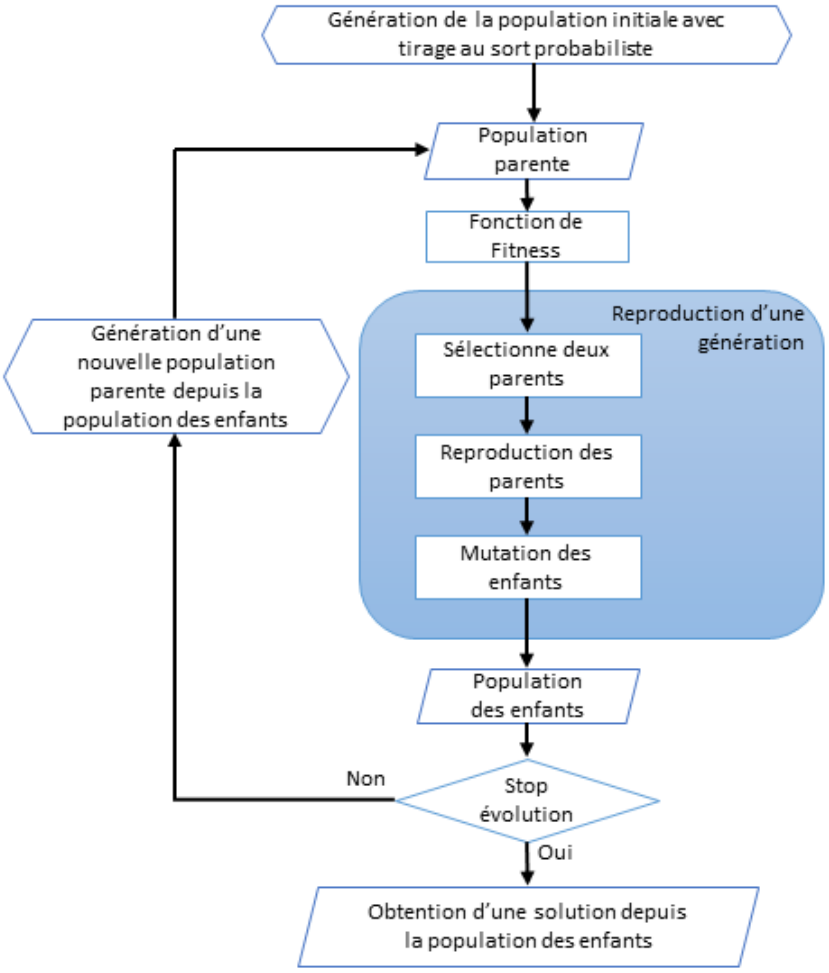

FIGURE C.8: Principe de notre algorithme génétique

composé de gènes où chaque gène étant un nombre réel représentant l'identifiant de la ressource de fournisseur j.

Nous voulons un algorithme qui converge rapidement vers une solution approchée. A cette fin, nous vérifions que la contrainte de capacité est respectée à l'initialisation comme nous le décrivons dans l'algorithme 6. La taille de la population est fixe et constitue une entrée de l'algorithme. Pour initialiser la population, nous procédons pour chaque individu à un tirage au sort (Algorithme 4) sur leur gène, en choisissant au hasard la ressource qui sera l'hôte de chaque gène. Tant que la solution obtenue n'est pas valable, ou ne respecte pas les contraintes et les relations exprimées dans les demandes initiales, nous répétons le processus. Le tirage au sort se poursuit jusqu'à ce que des solutions valables soient trouvées.

Le tirage n'est pas un simple tirage aléatoire simple, car il est guidé par la capacité des ressources de fournisseur d'accueillir les demandes des consommateurs. Nous introduisons un vecteur R_j qui enregistre le nombre de tests échoués sur la ressource du

Algorithm 4 Initialize population

Require: Pop, S, R_j
  for each $p \in Pop$ **do**
    initialize R_j
    for each $s \in S$ **do**
      repeat
        $s = randomDraw(p, R_j)$
      until isPossible(p)
    end for
  end for

fournisseur j pour capturer et estimer cette capacité d'accueil et l'utiliser pour conduire les futures explorations de l'espace de recherche.

Nous notons Pop la population, p un individu et S un ensemble de variables représentant les solutions. La variable s est juste une solution d'allocation. La variable P_{jl} représente la matrice des capacités des ressources fournisseur et C_{kl} est la matrice des capacités des ressources clientes.

Le tirage n'est pas un simple tirage aléatoire simple, car il est guidé par la capacité des ressources de fournisseur d'accueillir les demandes des consommateurs. DC_j qui spécifie le datacenter où le serveur j se trouve. Les variables booléennes $NODEused_j$ et $DCused_i$ indiquent si un serveur ou un datacenter est utilisé. Ces variables sont utilisées afin d'implémenter les contraintes et les relations entre les ressources.

Les variables utilisées dans la définition de notre algorithme sont listées ci-dessous pour que le lecteur puisse y faire référence plus rapidement : C.2:

TABLE C.2: Variables de l'algorithme

	Variables de l'algorithme
Pop	Ensemble des individus $Pop = \{1, ..., p\}$
S	Chromosome: Ensemble de gène $N = \{1, ..., s\}$
DC_j	spécifie le datacenter du serveur j
$NODEused_j$	Variable booléenne si un serveur est utilisé
$DCused_i$	Variable booléenne si un datacenter est utilisé

L'algorithme 5 fournit en sortie une ressource de fournisseur viable (qui peut accueillir la demande et respecter les contraintes et les relations) à utiliser pour le processus d'initialisation dans l'algorithme 4.

Algorithm 5 Tirage au sort aléatoire

Require: R_j
Ensure: $p \in Pop$
  List greatLocation
  for each $j \in P_{jl}$ **do**
    if i == Collections.min(R_j) **then**
      $greatLocation.add(j)$
    end if
  end for
  $p = Random.select(greatLocation)$
  return p

Nous conservons et stockons dans une liste les meilleures ressources fournisseur, ainsi que celles qui font partie des valeurs minimales de R_j. Les valeurs de R_j sont mises à jour durant le processus de vérification des contraintes (Algorithme 6). Pour finir, nous tirons aléatoirement au sort une ressource à partir de la liste fournie de meilleures ressources fournisseur et nous la transmettons au processus d'initialisation.

L'inconvénient avec les algorithmes évolutionnaires c'est qu'ils gèrent difficilement les contraintes strictes. Techniquement, il existe différentes méthodes possibles pour que cette question soit contournée:

- en excluant les individus qui ne sont pas en conformité avec les contraintes;
- en fixant les individus défectueux à travers le processus de réparation;
- en préservant les bons individus par le biais d'opérateurs spéciaux;
- en modifiant l'espace de recherche et en guidant l'algorithme (via des hyperplans).

Cet article se concentrera sur les deux premières méthodes. La méthode 1, qui exclut les individus qui ne respectent pas les contraintes, se révèle inefficace car trop d'individus finissent exclus. La méthode 2, que nous utilisons dans nos travaux, est conçue pour corriger les individus défectueux en appliquant un algorithme de recherche tabou sur eux. Le processus de réparation à travers une recherche tabou améliorée d'un algorithme génétique (NSGA-II) est décrite dans la Figure C.9.

Le processus de réparation grâce à la recherche tabou est rendu possible grâce à la recherche d'un serveur pouvant héberger les machines virtuelles (Figure C.10) qui sont hébergés sur des serveurs surchargés. Le processus de réparation est lancé lorsque les

```
1: procedure REPAIR(I)                          ▷ I is an individual
2:     serversError ← exceedingDetection(I)  ▷ we are
   scanning for servers where constraints are exceeded
3:     for  i ≠ numberOfVM()  do
4:         server ← getServerOfVM(I, i)
5:         if server ∈ serversError then
6:             I(i) ← findNeighbour(I, i)
7:         end if
8:     end for
9: end procedure
```

FIGURE C.9: Recherche tabou pour la réparation des individus de NSGA

individus non valides sont évalués. La méthode de correction a pour but de les rendre conformes aux contraintes. Chaque gène défectueux trouvé dans un individu sera alors traité et modifié en conséquence.

```
1: procedure FINDNEIGHBOR(I, i)     ▷ i is an ID of a VM
2:     for  j ≠ numberOfServer(I)  do
3:         if isValidAllocation(i, j) then
4:             return(j)              ▷ We return a valid server
5:         end if
6:     end for
7: end procedure
```

FIGURE C.10: Trouve le plus proche voisin respectant les contraintes

Algorithm 6 Checks whether the solution is valid

Require: p, P_{jl}, R_j
Ensure: $p \in Pop$
// Capacity constraint [Eq. C.10] below
for each $q \in Q_p$ **do**
capacityProvider = getVariable(q)
for each $j \in P_{jl}$ **do**
capacityProvider = capacityProvider - capacityConsumer
end for
if capacityProvider < 0 **then**
increaseAmount(R_j); return failureConstraint
end if
end for
// All relationship constraints below
switch RelationshipConstraints **do**
case ColocalizationSameDC **:** // [Eq. C.11]
for each $q \in Q_p$ **do**
if DC[q] ! = anyDC **then**
return failureConstraint
end if
end for
end case
case ColocalizationSameNode **:** // [Eq. C.12]
for each $q \in Q_p$ **do**
if q ! = anyNode **then**
return failureConstraint
end if
end for
end case
case TotalSeparation **:** // [Eq. C.13]
for each $q \in Q_p$ **do**
if DCused[DC[q]] == true OR
NODEused[q] == true **then**
return failureConstraint
else
NODEused[q] = true
end if
end for
end case
case SeparationOnDifferentNode **:** // [Eq. C.14]
for each $q \in Q_p$ **do**
if NODEused[q] == true **then**
return failureConstraint
else
NODEused[q] = true
DCused[DC[q]] = true
end if
end for
end case
case SeparationOnDifferentDC **:** // [Eq. C.15]
for each $q \in Q_p$ **do**
if DCused[DC[q]] == true **then**
return failureConstraint
else
DCused[DC[q]] = true
end if
end for
end case
end switch

C.3.1 Evaluation des algorithmes

Nous évaluons les performances de plusieurs algorithmes (Round Robin, Branch and Bound, programmation par contraintes) afin de révéler les limites et les capacités de notre algorithme génétique. Nous évaluons la performance du temps de résolution, le coût d'indisponibilité, l'utilisation globale des ressources et des coûts d'exploitation. Ces simulations sont effectuées sur un ordinateur Intel NUC avec Intel Celeron 847 (1.1 GHz - 2 Mo Cache) et 4Go DDR3 RAM.

Nous comparons les algorithmes grâce à un scénario linéaire et à un scénario aléatoire. Le scénario linéaire consiste à allouer des machines virtuelles sur des hyperviseurs qui peuvent en héberger au maximum deux. Le scénario aléatoire consiste à allouer des machines virtuelles sur des hyperviseurs où chaque ressource a des capacités définient aléatoirement.

C.3.1.1 Avec un scénario linéaire

Dans le cadre du scénario linéaire, nous effectuons une comparaison sur le temps de résolution des algorithmes (Fig. C.11). Nous constatons que sur des scénarios linéaires les algorithmes de Round Robin et de programmation par contraintes sont plus performants de, respectivement, 99.80% et de 80.80% sur des petits problèmes par rapport à notre algorithme NSGA-2 modifié. Quand les problèmes augmentent en taille suivant le nombre de ressources, notre algorithme est plus performant que les autres algorithmes.

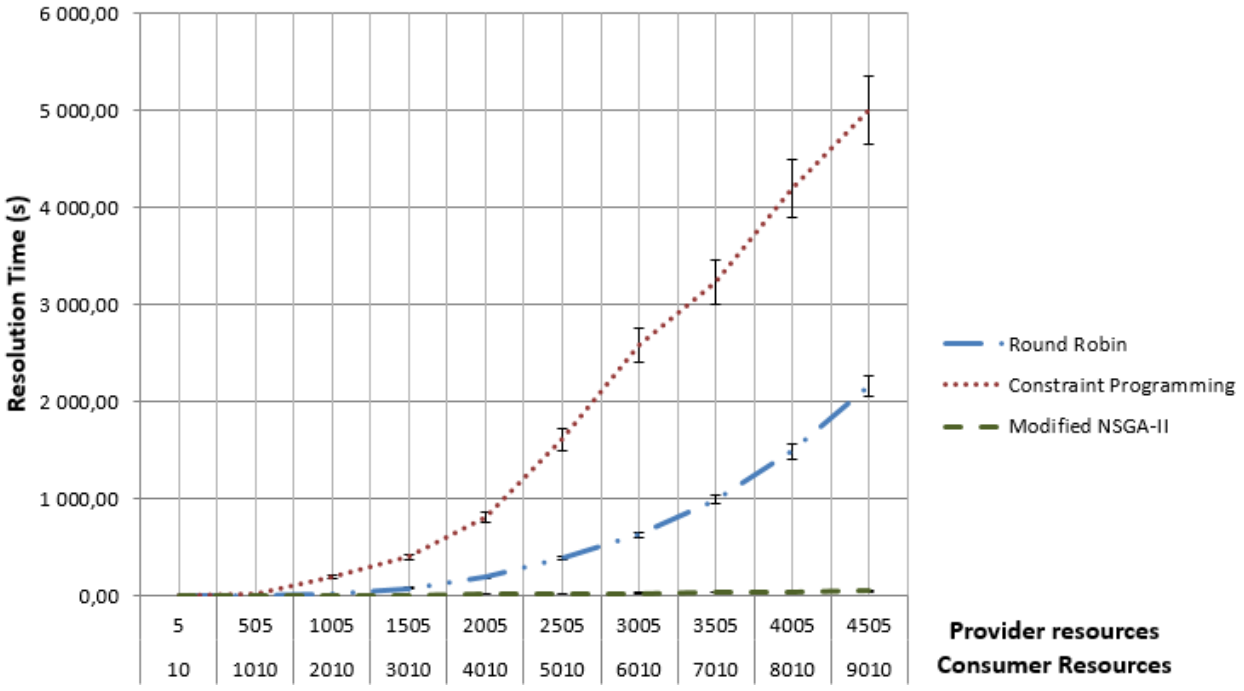

FIGURE C.11: Comparaison du temps de résolution avec des scénarios linéaire

Ensuite, nous comparons les coûts moyens générés par le placement adressé par les algorithmes (Fig. C.12). Nous constatons que l'algorithme de Round Robin est tout aussi efficace que l'algorithme de programmation par contraintes qui fournit l'optimal sur des problèmes linéaires. Notre algorithme génétique s'éloigne des coûts optimaux au fur et à mesure de l'augmentation de la taille du problème.

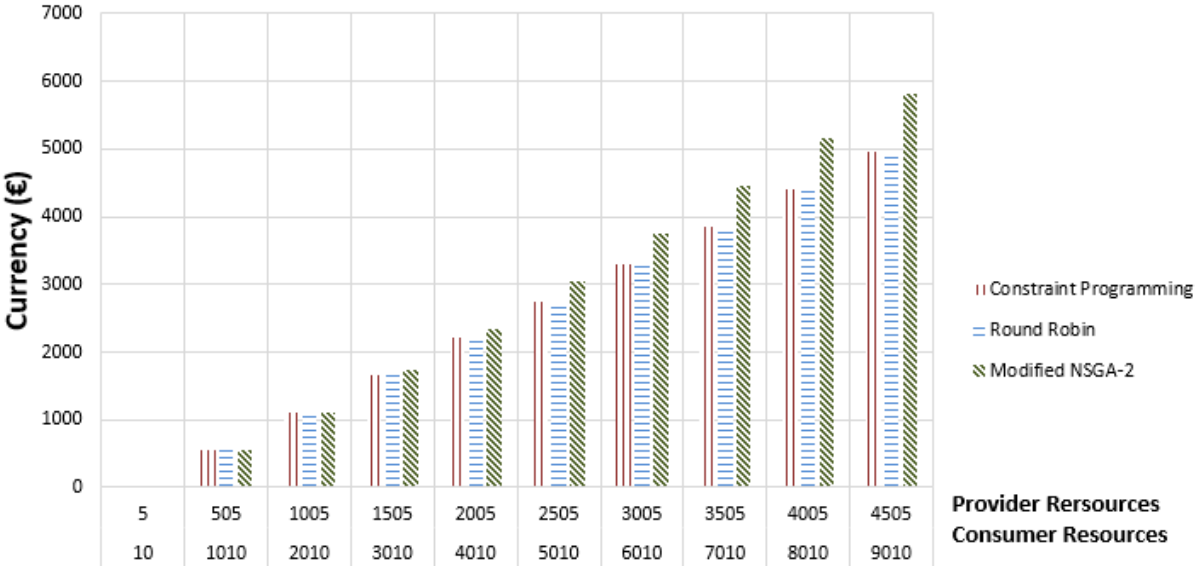

FIGURE C.12: Comparaison des coûts globaux avec des scénarios linéaire

Après nous comparons le pourcentage de requêtes des consommateurs pouvant être accueillies par l'infrastructure du fournisseur suivant l'algorithme de placement utilisé (Fig. C.13). Le pourcentage moyen de ressources du consommateur hébergées par le fournisseur diminue en fonction de la taille du problème en utilisant notre algorithme NSGA-2 modifié. Cela est dû au fait que l'algorithme génétique ne trouve pas toujours les solutions optimales.

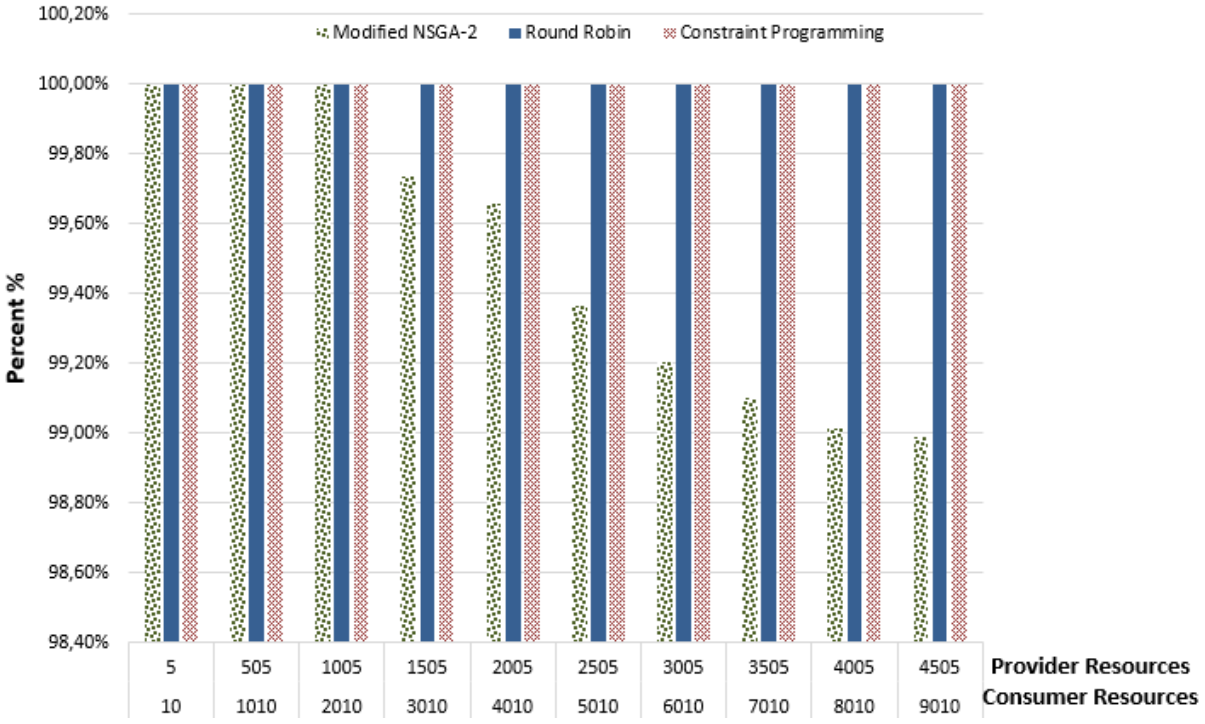

FIGURE C.13: Comparaison du pourcentage de ressources du consommateur hébergées avec des scénarios linéaires

Pour terminer, nous montrons le nombre de ressources du fournisseur utilisées pour héberger les demandes des consommateurs (Fig. C.14). Il apparaît que le nombre de ressources du fournisseur utilisées est proportionnel aux ressources des consommateurs hébergées dans le cadre d'un scénario linéaire. L'algorithme de Round Robin et de programmation par contraintes fournissent les solutions optimales. Notre algorithme NSGA-2 modifié perd en efficacité et s'éloigne de l'optimal en fonction de la taille du problème.

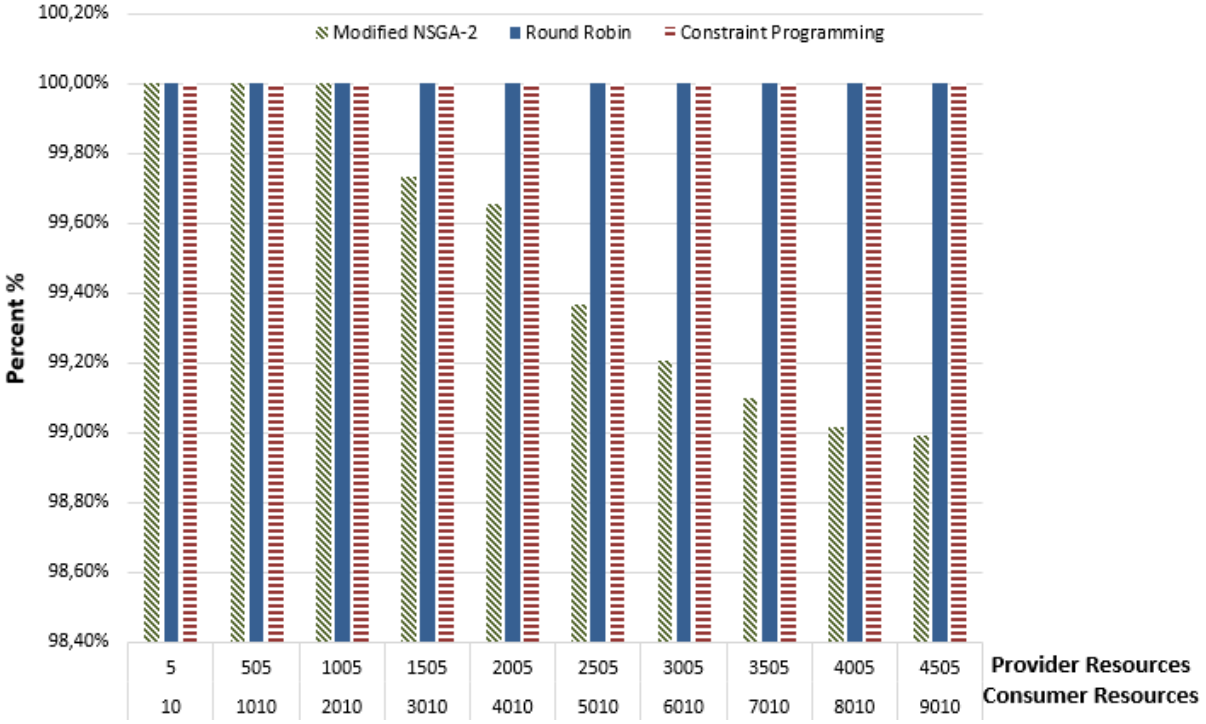

FIGURE C.14: Comparaison du pourcentage de ressources du fournisseur utilisées avec des scénarios linéaires

Pour conclure, dans le cas de scénarios linéaires avec des ressources fournisseur et consommateur homogènes, les algorithmes de Round Robin et de programmation par contraintes fournissent des solutions optimales. Mais le temps de résolution est exponentiel. Notre algorithme NSGA-2 modifié s'éloigne des solutions optimales en fonction de la taille du problème.

C.3.1.2 Avec un scénario aléatoire

Dans le cadre d'un scénario aléatoire, les ressources du fournisseur et les requêtes des consommateurs sont définies suivant des ressources disposant de capacités aléatoires. Nous commençons par comparer le temps de résolution pour obtenir une solution puis nous mettons en corrélation les coûts moyens générés par une plate-forme de Cloud Computing. Pour finir, nous comparons le pourcentage moyen de ressources des consommateurs hébergées et le pourcentage moyen des ressources du fournisseur utilisées.

Nous effectuons une comparaison du temps de résolution des algorithmes (Fig. C.15) sur des scénarios aléatoires. Nous constatons que l'algorithme de Round Robin est devenu le moins efficace en termes de temps de résolution. L'algorithme de programmation par contraintes et l'algorithme de round robin ont également une évolution exponentielle.

L'algorithme modifié NSGA-2 a, quant à lui, une croissance quasi-polynomiale au niveau du temps de résolution.

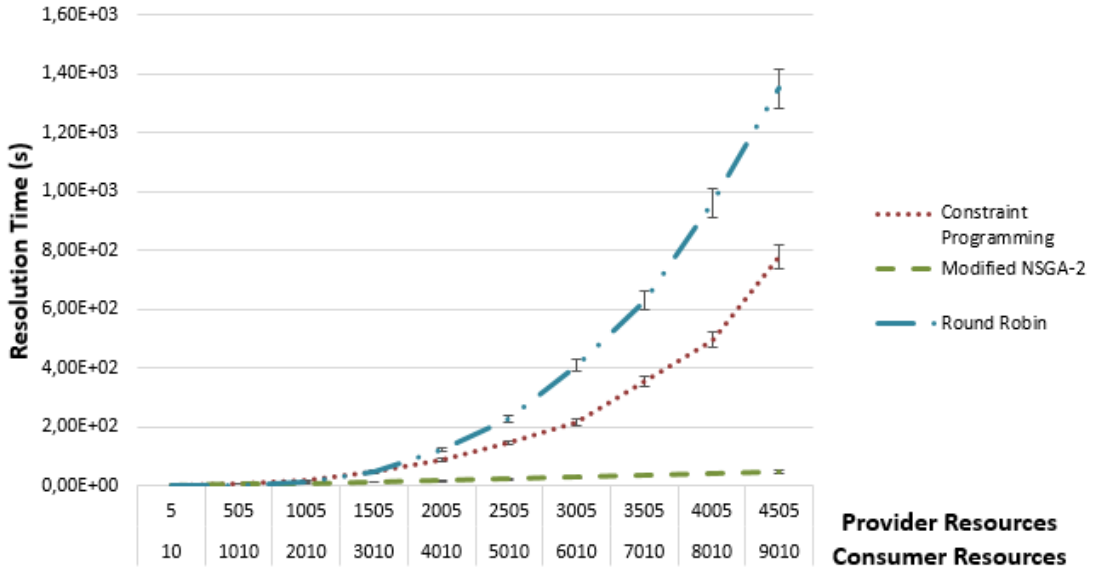

FIGURE C.15: Comparaison du temps de résolution avec des scénarios aléatoires

Ensuite, nous comparons les coûts moyens générés par le placement adressé par les algorithmes (Fig. C.16). Nous mettons en évidence que l'algorithme de Round Robin n'est pas efficace, car il génère le plus de coûts globaux par rapport aux solutions optimales données par l'algorithme de programmation par contraintes.

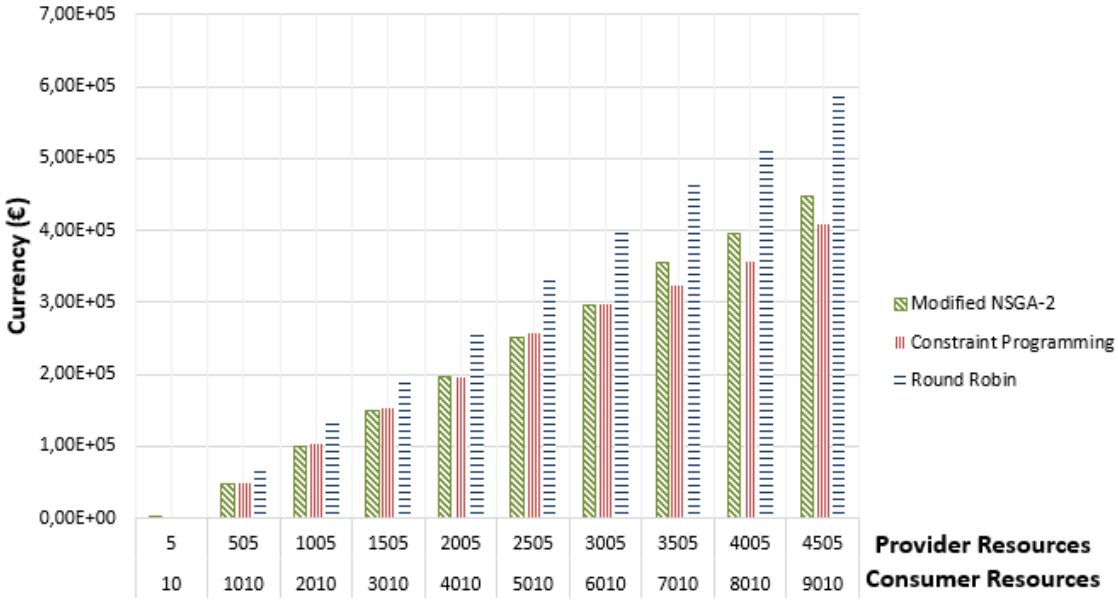

FIGURE C.16: Comparaison des coûts globaux avec des scénarios aléatoires

Notre algorithme NSGA-2 modifié, dans le cadre de ces scénarios, génère plus de coûts globaux moyens par rapport à l'optimal. Cela est dû au placement initial décidé aléatoirement, car l'algorithme génétique évite de déplacer trop de ressources consommateurs.

Ensuite nous comparons le pourcentage de requêtes des consommateurs pouvant être accueillies par l'infrastructure du fournisseur suivant l'algorithme de placement utilisé (Fig. C.17). Comme dans le cadre des scénarios linéaires, l'algorithme NSGA-2 modifié ne trouve pas en permanence la solution optimale et la moyenne de placement des ressources consommateur chute jusqu'à 98.324%. Les autres algorithmes parviennent à placer en permanence les ressources.

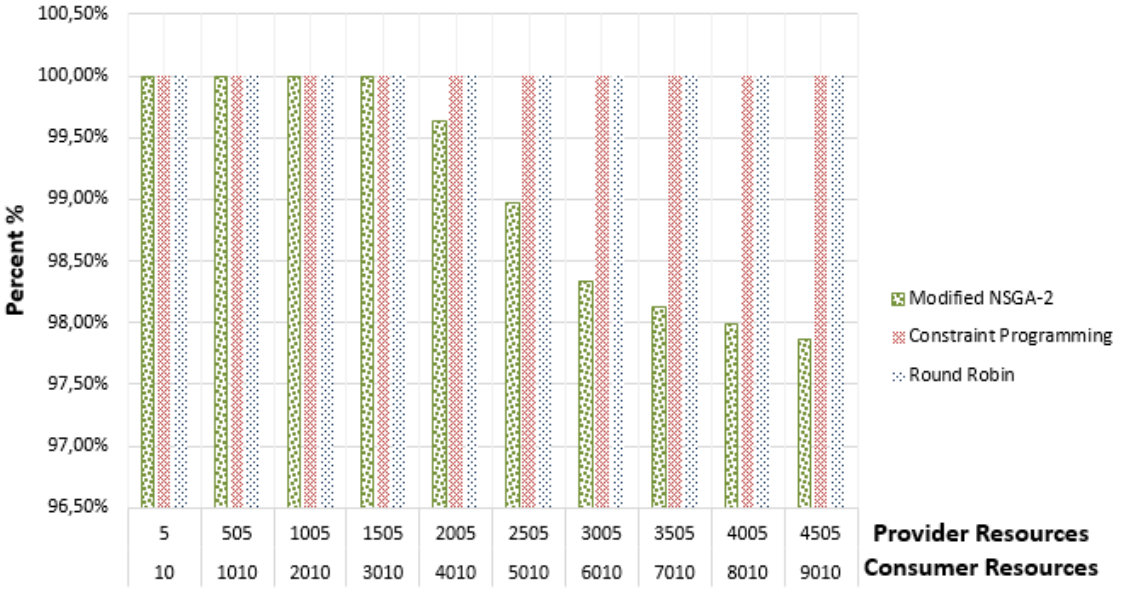

FIGURE C.17: Comparaison du pourcentage de ressources du consommateur hébergées avec des scénarios aléatoires

Pour terminer, nous montrons le nombre de ressources du fournisseur utilisées pour héberger les demandes des consommateurs (Fig. C.18). Nous constatons que l'algorithme de Round Robin n'arrive pas à consolider le placement des ressources clientes sur l'ensemble de l'infrastructure. L'algorithme de programmation par contraintes tente d'utiliser le maximum d'hyperviseur du fournisseur pour allouer les ressources des consommateurs. Notre algorithme NSGA-2 modifié tente d'utiliser au mieux les hyperviseurs du fournisseur.

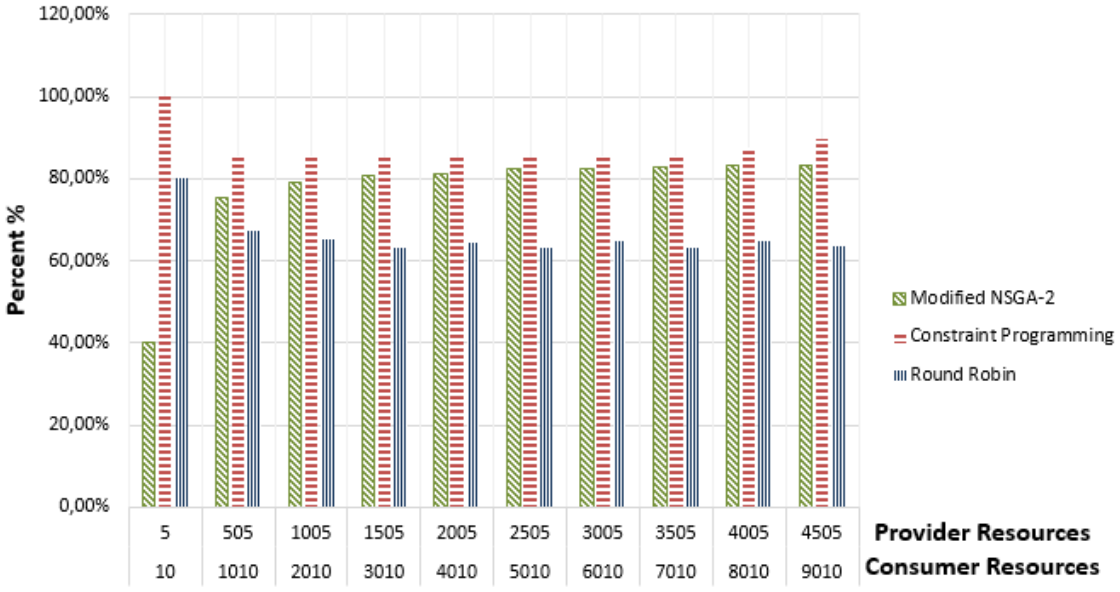

FIGURE C.18: Comparaison du pourcentage de ressources du fournisseur utilisées avec des scénarios aléatoires

En complément, nous comparons le temps moyen de résolution de l'ensemble des algorithmes. Nous effectuons des évaluations sur des scénarios jusqu'à 800 serveurs et 1600 machines virtuelles car il s'agit de la taille critique la plus courante au sein des plateformes de Cloud Computing comme OpenStack, OpenNebula, etc. En effet, les fournisseurs préfèrent installer plusieurs plateformes plutôt que de gérer une seule installation de grande échelle.

Les paramètres utilisés dans NSGA-II et NSGA-III sont listés dans le Tableau C.3.

Parameter	Value
populationSize	100
Number of evaluations	10000
sbx.rate	0.70
sbx.distributionIndex	15.00
pm.rate	0.20
pm.distributionIndex	15.00

TABLE C.3: Paramétrage de NSGA II et III

La Figure C.19 la moyenne du temps de résolution de l'ensemble des algorithmes. L'algorithme de programmation par contraintes semble être plus efficace dans le contexte de scénarios aléatoires avec des ressources hétérogènes, comparé à l'algorithme de Round Robin qui tente de placer les ressources en parcourant l'ensemble des ressources du fournisseur. Le parcours de l'ensemble des ressources du fournisseur par l'algorithme de Round Robin est très coûteux en temps de résolution, ce qui fait que cette méthode est inadaptée pour les scénarios aléatoires. Les algorithmes évolutionnaires consomment

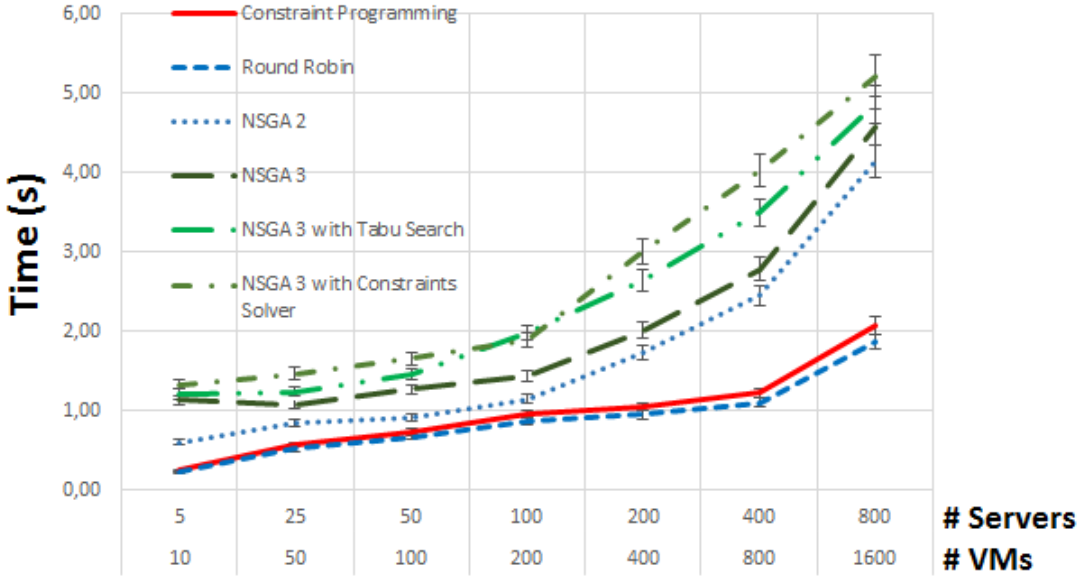

FIGURE C.19: Comparaison de la moyenne du temps de Résolution avec des scenarios aléatoire

plus de temps que les autres car ils doivent faire évoluer leurs populations mais l'écart n'est que de 18.75%, ce qui est négligeable si nous obtenons une solution proche des attentes du décideur.

La Figure C.20 souligne que les ressources du fournisseur ne sont pas toujours pleinement utilisées. Nous pouvons voir que les algorithmes NSGA-II et III ne fournissent pas une consolidation de l'infrastructure et ils placent des machines virtuelles sur des grappes de serveurs. Mais cela nuit à la performance par rapport aux utilisateurs. Les autres algorithmes réussissent à faire de la consolidation au sein de l'infrastructure.

La Figure C.21 indique le pourcentage des ressources des clients hébergés sur l'infrastructure du fournisseur. Nos scénarios aléatoires montrent que toutes les demandes des utilisateurs sont satisfaites.

La figure C.22 présente le nombre de dépassement de contraintes durant l'execution des algorithmes. Nous constatons que les algorithmes NSGA-II et NSGA-III basés essentiellement sur des processus stochastiques n'arrivent pas à trouver de solutions viables. L'algorithme NSGA-III que nous avons amélioré d'une recherche tabou et d'un algorithme de résolution de contraintes ne génère aucune erreur.

L'algorithme de programmation par contraintes repose sur des contraintes pour accélérer le temps d'exécution (pour éliminer rapidement des solutions irréalisables) mais au détriment du taux de rejet (ou taux d'acceptation). Ceci peut être vu en analysant les résultats de la figure C.23 où les algorithmes de Round Robin et de programmation par contraintes rejettent beaucoup plus de demandes que les algorithmes évolutionnaires.

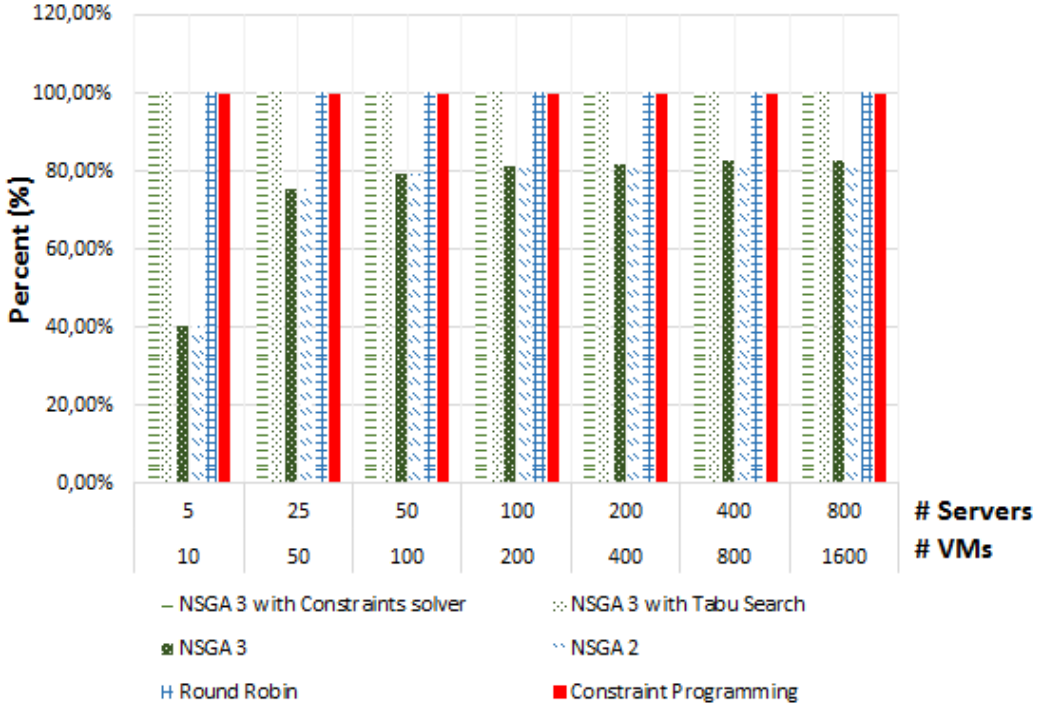

FIGURE C.20: Taux d'tilisation des serveurs avec des Scenarios aléatoires

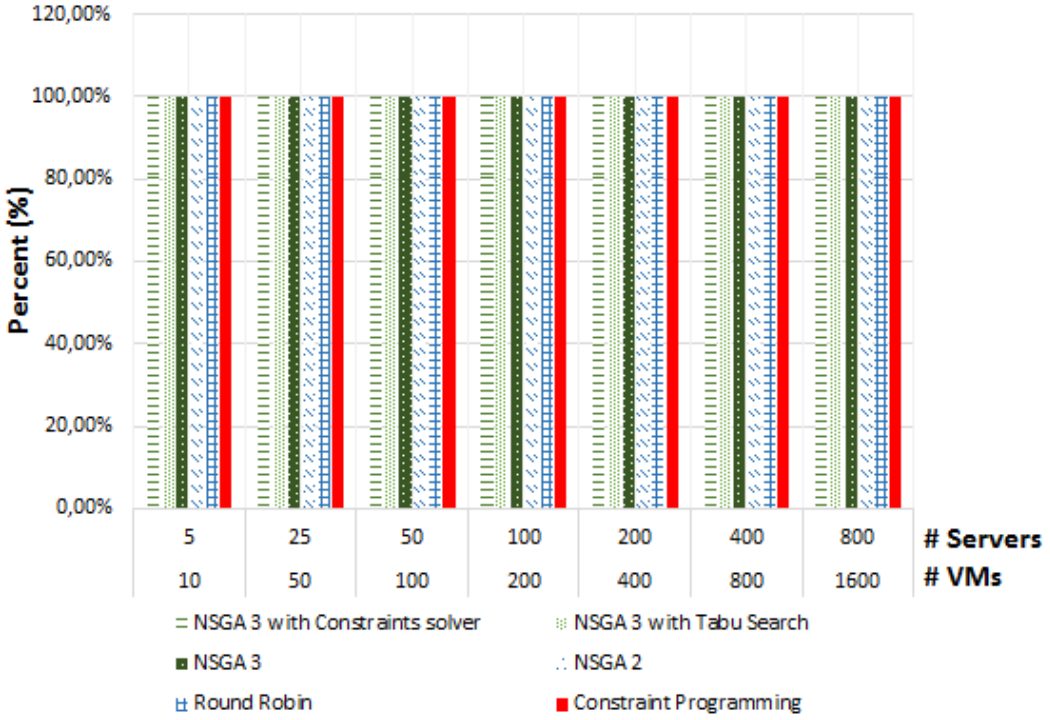

FIGURE C.21: Pourcentage de ressources clientes hébergées avec les scénarios aléatoires

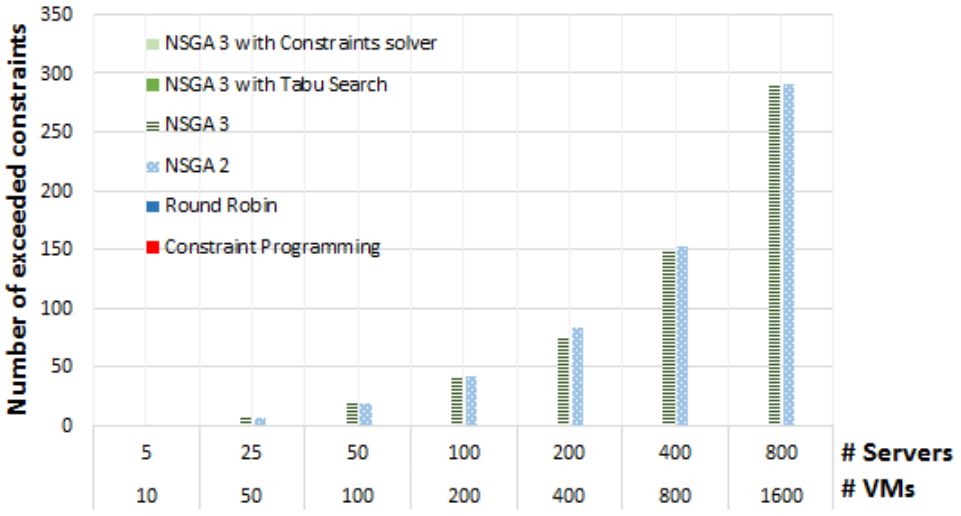

FIGURE C.22: Comparaison de la moyenne des contraintes non respectées

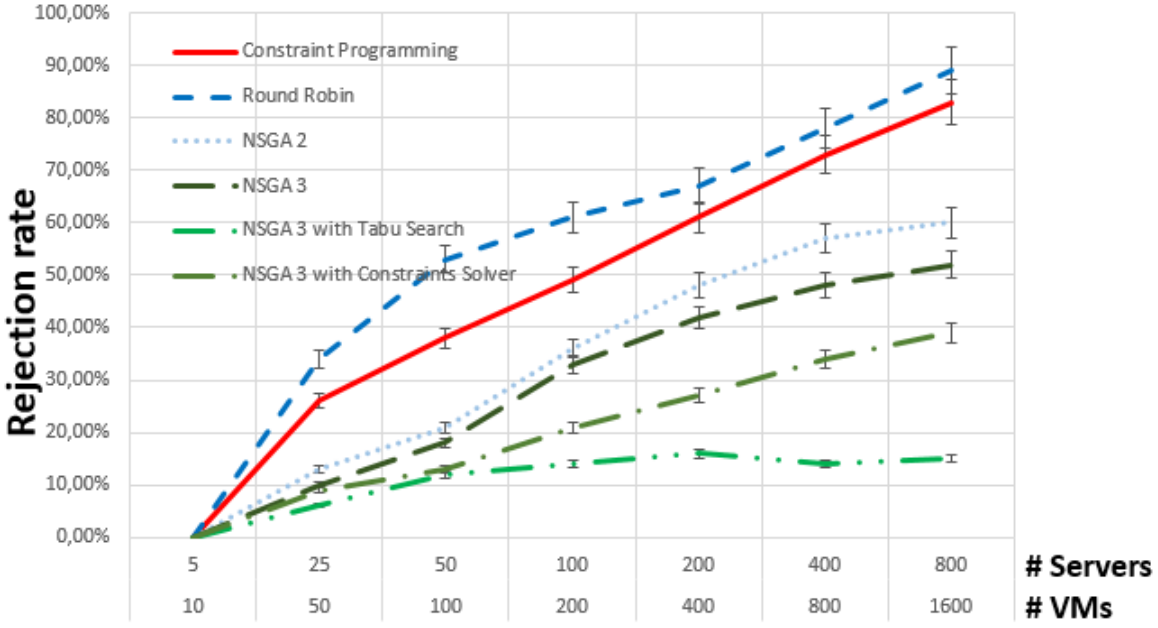

FIGURE C.23: Comparaison du taux de rejet en fonction de la taille des problèmes à traiter

L'algorithme NSGA-III amélioré avec une recherche tabou que nous proposons surpasse tous les autres algorithmes en termes de taux d'acceptation et génère ainsi les plus grands revenus pour les fournisseurs tout en répondant aux intérêts des deux acteurs, les consommateurs et les fournisseurs. Les algorithmes non modifiés NGSA et l'algorithme NSGA-III amélioré avec la méthode de programmation par contraintes améliore le taux de rejet, mais n'ont pas la force de réparer les gènes et les individus, car ils sont basés exclusivement sur des méthodes stochastiques.

La figure suivante C.24 représente une vue de la moyenne des coûts générés par notre

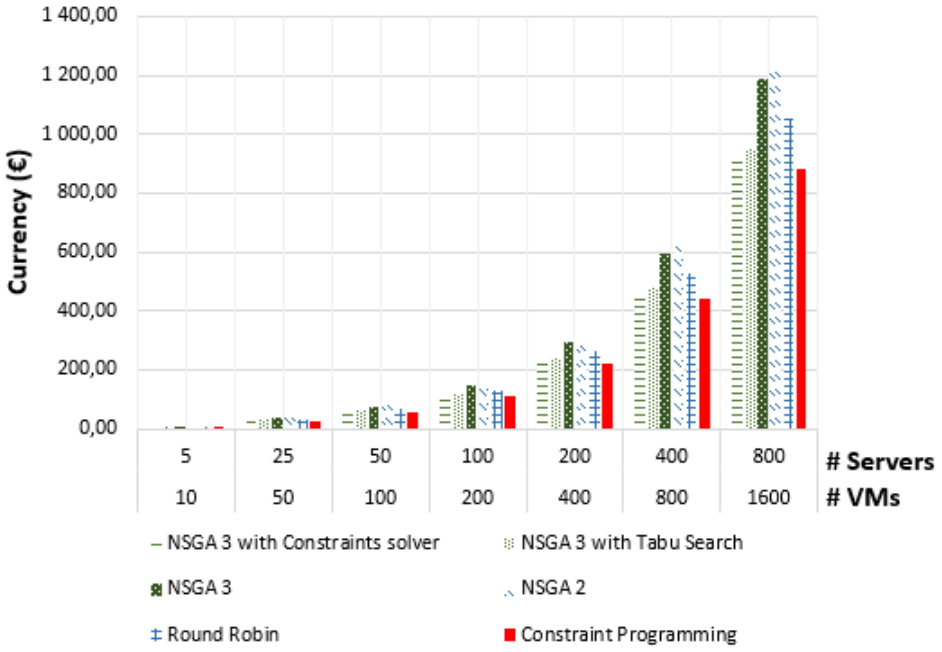

FIGURE C.24: Comparaison des moyennes de coûts générés par algorithmes

scénario aléatoire. Il apparaît que l'algorithme de Round Robin fait abstraction des coûts et génère ainsi une hausse de ceux-ci. L'algorithme de programmation par contraintes fournit une solution qui respecte les contraintes mais il est difficile d'y inclure les préférences du décideur. Les algorithmes basiques NSGA-II et NSGA-III ne respectent pas les contraintes et génèrent un maximum de coûts pour le fournisseur. NSGA-III avec une recherche tabou ou une résolution de programmation par contraintes engendre moins de coûts relatifs à l'exploitation des serveurs et l'utilisation/indisponibilité des machines virtuelles. Cela est dû au fait que l'algorithme génétique dirige, grâce à l'objectif pécunier, l'exploration des solutions dans un espace de recherche restreint. Pour conclure sur l'aspect pécunier, le processus de réparation n'impacte que faiblement les coûts.

Pour conclure, dans le cas de scénarios aléatoires avec des ressources fournisseur et consommateur hétérogènes, l'algorithme de programmation par contraintes fournit des solutions optimales, mais il est relativement lent pour un système dynamique et flexible comme l'est une plate-forme de Cloud Computing. L'algorithme de Round Robin n'est pas du tout efficace dans un environnement hétérogène, car le temps de résolution est très élevé et il génère beaucoup de coûts globaux par rapport à l'optimal. L'algorithme NSGA-2 modifié a un temps de résolution extrêmement rapide, mais génère des coûts supplémentaires par rapport à l'optimal.

C.4 Conclusion

Ce chapitre résume les travaux de recherche et les contributions de la thèse sur le placement de ressource au sein d'une plate-forme de Cloud Computing. Nous mettons aussi en évidence les points importants qui serviront de base pour de futurs axes de recherche.

C.4.1 Conclusion et discussions

Ce mémoire de thèse aborde les enjeux de recherche dans le domaine du placement de ressource (Services) au sein d'une plate-forme de Cloud Computing de type Infrastructure as a Service (IaaS).

Nous mettons en évidence dans les prochains paragraphes nos principales contributions de recherche effectuées durant ces années de travaux :

- Nous avons passé en revue les méthodes existantes permettant de résoudre les problèmes d'allocation au sein d'une plate-forme de Cloud Computing. Nous nous sommes intéressés principalement à la théorie des permutations en optimisation combinatoire et aux algorithmes effectuant ce type d'action. Nous avons identifié les différentes métriques permettant de comparer ces algorithmes de placement entre eux. Ces indicateurs sont divisés en trois groupes : les indicateurs techniques, les indicateurs business et les indicateurs de tarification. Nous terminons en présentant les algorithmes de placement qui sont actuellement utilisés par les industriels. (Voir Chapitre 2)

- Nous avons identifié les points-clés du modèle d'affaires des utilisateurs et du fournisseurs permettant de prendre en compte leurs contraintes d'affinité et d'anti-affinité entre leurs ressources (Chapitre 3 ou en français C.2).

- Nous proposons un modèle de représentation d'une plate-forme de Cloud comme une extension au modèle proposé par l'Open Cloud Computing Interface (OCCI). Notre extension apporte une description plus généraliste de ce que sont les propriétés d'une ressource de calcul ou de réseaux. Nous n'établissons pas de distinctions entre ces deux types de ressource dans la définition de leur capacité. Nous définissons des contraintes d'affinité et d'anti-affinité sur ces ressources afin d'obtenir, grâce à elles, une meilleure disponibilité des ressources sans dégradation (Chapitre 3 ou en français C.2).

- Un algorithme génétique modifié afin de permettre une convergence plus rapide. Nous avons amélioré les opérateurs de variation (initialisation et croisement) en

permettant une initialisation des individus qui respectent les contraintes de capacité (Chapitre 4 ou en français C.3).

- Nous introduisons une boucle de rétroaction afin de vérifier le placement des ressources à des temps réguliers en minimisant les coûts de déplacements.

C.4.2 Perspectives et axes futurs de recherche

Au-delà des principales contributions de cette thèse, nous prévoyons d'explorer un certain nombre d'axes de recherche et d'améliorations.

- L'axe de recherche important pour la suite de nos travaux repose sur la proposition d'un plan de reconfiguration optimale entre le placement actuel des ressources sur la plate-forme de Cloud Computing et le nouveau placement fourni par notre algorithme. C'est un problème aussi complexe que de trouver le placement des ressources.

- Nous utilisons un algorithme génétique pour trouver une solution approchée prenant en compte les intérêts de chaque acteur. L'inconvénient des algorithmes génétiques est leur lenteur de convergence. Il conviendrait de réaliser des expérimentations avec d'autres algorithmes d'optimisation. L'avantage des algorithmes génétiques, est qu'ils sont plus facilement parallélisables en parcourant l'espace des solutions sur différents serveurs.

- Nous avons choisi d'utiliser dans nos travaux une représentation vectorielle et matricielle des ressources et de l'allocation. Cette méthode de représentation des informations est bas-niveau. Une représentation plus générique et plus haut niveau sous la forme de graphe de Johnson est une piste à prendre en compte afin d'améliorer la complexité en temps d'un algorithme de placement des ressources.

- Nous pourrions définir et modéliser les besoins business grâce à une prise en compte des capacités réseaux dans l'application des contraintes d'affaires des utilisateurs et du fournisseur de service au niveau du placement des ressources.

Bibliography

[1] Kalyanmoy Deb and Ram B. Agrawal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115–148, 1995. URL `citeseer.ist.psu.edu/deb95simulated.html`.

[2] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing, 2009.

[3] Martin Schindewolf, Barna Bihari, John Gyllenhaal, Martin Schulz, Amy Wang, and Wolfgang Karl. What scientific applications can benefit from hardware transactional memory? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 90:1–90:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press. ISBN 978-1-4673-0804-5. URL `http://dl.acm.org/citation.cfm?id=2388996.2389118`.

[4] A.M. Bento and A. Aggarwal. *Cloud Computing Service and Deployment Models: Layers and Management: Layers and Management*, volume ISBN 9781466621886. Business Science Reference, 2013.

[5] Vasilios Andrikopoulos, Santiago Gomez Saez, Dimka Karastoyanova, and Andreas Weiss. Collaborative, dynamic and complex systems - modeling, provision and execution. In *CLOSER*, 2014.

[6] A. Amies; H. Sluiman; Q.G. Tong; and G. Liu. *Developing and Hosting Applications on the Cloud*. ISBN 9780133066852. IBM Press. Pearson Education, 2012. URL http://books.google.fr/books?id=4gwIYbtTH5MC.

[7] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, December 1999. ISSN 0360-0300. doi: 10.1145/344588.344618. URL `http://doi.acm.org/10.1145/344588.344618`.

[8] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *SODA*, volume 99, pages 185–194. Citeseer, 1999.

[9] Alexander Schrijver. On the history of combinatorial optimization (till 1960). In G.L. Nemhauser K. Aardal and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1 – 68. Elsevier, 2005. doi: http://dx.doi.org/10.1016/S0927-0507(05)12001-5. URL http://www.sciencedirect.com/science/article/pii/S0927050705120015.

[10] M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer, Heidelberg, 2010.

[11] G. Rinaldi and L.A. Wolsey, editors. *Integer Programming and Combinatorial Optimization - IPCO III*. Centro Ettore Majorana, Erice, 1993.

[12] M. Jünger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors. *Combinatorial Optimization and Integer Programming*, volume 124 of *Mathematical Programming*. 2010.

[13] Piercar Nicola. *Mainstream Mathematical Economics in the 20th Century*, chapter Linear Programming and Extensions, pages 133–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-662-04238-0. doi: 10.1007/978-3-662-04238-0_13. URL http://dx.doi.org/10.1007/978-3-662-04238-0_13.

[14] K. G. Murty. *Linear Programming*. John Wiley & Sons, 1983.

[15] L. Kantorovich. On the calculation of production inputs. *Problems in Economics*, 3(1):3–10, 1960. doi: 10.2753/PET1061-199103013. URL http://www.tandfonline.com/doi/abs/10.2753/PET1061-199103013.

[16] Tjalling C. Koopmans. A Note About Kantorovich's Paper, "Mathematical Methods of Organizing and Planning Production". *Management Science*, 6(4):363–365, July 1960. URL https://ideas.repec.org/a/inm/ormnsc/v6y1960i4p363-365.html.

[17] Tjalling C. Koopmans. On the Evaluation of Kantorovich's Work of 1939. *Management Science*, 8(3):264–265, April 1962. URL https://ideas.repec.org/a/inm/ormnsc/v8y1962i3p264-265.html.

[18] George b. dantzig: Operations research icon. *Operations Research*, 53(6):892–898, 2005. doi: 10.1287/opre.1050.0250. URL http://dx.doi.org/10.1287/opre.1050.0250.

[19] George B. Dantzig and Mukund N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 0-387-94833-3.

[20] David S. Johnson. Vector bin packing. In *Encyclopedia of Algorithms*. 2015. URL http://dx.doi.org/10.1007/978-3-642-27848-8_495-1.

[21] Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006. doi: 10.1287/moor.1050.0168. URL http://dx.doi.org/10.1287/moor.1050.0168.

[22] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, May 2007. doi: 10.1109/INM.2007.374776.

[23] Mauro Maria Baldi and Maurizio Bruglieri. On the generalized bin packing problem. *International Transactions in Operational Research*, pages n/a–n/a, 2016. ISSN 1475-3995. doi: 10.1111/itor.12258. URL http://dx.doi.org/10.1111/itor.12258.

[24] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *Computers, IEEE Transactions on*, C-35(5):389–393, May 1986. ISSN 0018-9340. doi: 10.1109/TC.1986.1676781.

[25] Merve Şeker and Nilay Noyan. Stochastic optimization models for the airport gate assignment problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(2):438 – 459, 2012. ISSN 1366-5545. doi: http://dx.doi.org/10.1016/j.tre.2011.10.008. URL http://www.sciencedirect.com/science/article/pii/S1366554511001335.

[26] Conway; Richard;, Maxwell; William L;, and Miller; Louis W. *Theory of scheduling / [by] Richard W. Conway, William L. Maxwell [and] Louis W. Miller*. Addison-Wesley Pub. Co Reading, Mass, 1967.

[27] David M Ryan and Brian A Foster. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280, 1981.

[28] Arnaud Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

[29] Aravind Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29(2):648–670, October 1999. ISSN 0097-5397. doi: 10.1137/S0097539796314240. URL http://dx.doi.org/10.1137/S0097539796314240.

[30] Robert S Garfinkel and George L Nemhauser. *Integer programming*, volume 4. Wiley New York, 1972.

[31] S. Kullaaok. An introduction to combinatorial analysis (john riordan). *SIAM Review*, 2(1):54–54, 1960. doi: 10.1137/1002018.

[32] Richard M. Karp, Michael Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 289–298, New York, NY, USA, 1984. ACM. ISBN 0-89791-133-4. doi: 10.1145/800057.808693. URL `http://doi.acm.org/10.1145/800057.808693`.

[33] O. O. Iemets and T. O. Parfionova. Transportation problems on permutations: Properties of estimates in the branch and bound method. *Cybernetics and Sys. Anal.*, 46(6):953–959, November 2010. ISSN 1060-0396. doi: 10.1007/s10559-010-9276-0. URL `http://dx.doi.org/10.1007/s10559-010-9276-0`.

[34] B. Naderi and Rubén Ruiz. The distributed permutation flowshop scheduling problem. *Comput. Oper. Res.*, 37(4):754–768, April 2010. ISSN 0305-0548. doi: 10.1016/j.cor.2009.06.019. URL `http://dx.doi.org/10.1016/j.cor.2009.06.019`.

[35] Robert Sedgewick. Permutation generation methods. *ACM Comput. Surv.*, 9 (2):137–164, June 1977. ISSN 0360-0300. doi: 10.1145/356689.356692. URL `http://doi.acm.org/10.1145/356689.356692`.

[36] P. Bratley. Algorithm 306: Permutations with repetitions. *Commun. ACM*, 10(7): 450–451, July 1967. ISSN 0001-0782. doi: 10.1145/363427.363473. URL `http://doi.acm.org/10.1145/363427.363473`.

[37] Gideon Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. ACM*, 20(3):500–513, July 1973. ISSN 0004-5411. doi: 10.1145/321765.321781. URL `http://doi.acm.org/10.1145/321765.321781`.

[38] Phillip J. Chase. Algorithm 382: Combinations of m out of n objects [g6]. *Commun. ACM*, 13(6):368–, June 1970. ISSN 0001-0782. doi: 10.1145/362384.362502. URL `http://doi.acm.org/10.1145/362384.362502`.

[39] Fermín Galán, Americo Sampaio, Luis Rodero-Merino, Irit Loy, Victor Gil, and Luis M. Vaquero. Service specification in cloud environments based on extensions to open standards. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, COMSWARE '09, pages

19:1–19:12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-353-2. doi: 10.1145/1621890.1621915. URL http://doi.acm.org/10.1145/1621890.1621915.

[40] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '09, pages 1:1–1:6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-847-6. doi: 10.1145/1657120.1657121. URL http://doi.acm.org/10.1145/1657120.1657121.

[41] E. Feller, C. Rohr, D. Margery, and C. Morin. Energy management in iaas clouds: A holistic approach. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 204–212, June 2012. doi: 10.1109/CLOUD.2012.50.

[42] Dimitrios Kourtesis, Jose María Alvarez-Rodríguez, and Iraklis Paraskakis. Semantic-based qos management in cloud systems: Current status and future challenges. *Future Generation Computer Systems*, 32:307 – 323, 2014. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2013.10.015. URL http://www.sciencedirect.com/science/article/pii/S0167739X1300232X. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.

[43] Varunya Attasena, Nouria Harbi, and Jérôme Darmont. fvss: A new secure and cost-efficient scheme for cloud data warehouses. In *Proceedings of the 17th International Workshop on Data Warehousing and OLAP*, DOLAP '14, pages 81–90, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-0999-8. doi: 10.1145/2666158.2666173. URL http://doi.acm.org/10.1145/2666158.2666173.

[44] Giuseppe Cicotti, Luigi Coppolino, Rosario Cristaldi, Salvatore D'Antonio, and Luigi Romano. *Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 – September 2, 2011, Revised Selected Papers, Part I*, chapter QoS Monitoring in a Cloud Services Environment: The SRT-15 Approach, pages 15–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-29737-3. doi: 10.1007/978-3-642-29737-3_3. URL http://dx.doi.org/10.1007/978-3-642-29737-3_3.

[45] Jelena Zdravkovic, Janis Stirna, Martin Henkel, and Jānis Grabis. *Advanced Information Systems Engineering: 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, chapter Modeling Business Capabilities and Context Dependent Delivery by Cloud Services, pages 369–383.

Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38709-8. doi: 10.1007/978-3-642-38709-8_24. URL http://dx.doi.org/10.1007/978-3-642-38709-8_24.

[46] M. Di Sano. Business intelligence as a service: A new approach to manage business processes in the cloud. In *WETICE Conference (WETICE), 2014 IEEE 23rd International*, pages 155–160, June 2014. doi: 10.1109/WETICE.2014.95.

[47] NIST Cloud Computing Reference Architecture and Taxonomy Working Group. Cloud computing service metrics description. Technical report, Gaithersburg, MD, United States, 2015.

[48] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1): 45–52, January 2011. ISSN 0146-4833. doi: 10.1145/1925861.1925869. URL http://doi.acm.org/10.1145/1925861.1925869.

[49] Ann-Kristin Achleitner, Reiner Braun, and Florian Tappeiner. Structure and determinants of financial covenants in leveraged buyouts - evidence from an economy with strong creditor rights. CEFS Working Paper Series 2009-15, Technische Universität München (TUM), Center for Entrepreneurial and Financial Studies (CEFS), 2009. URL http://EconPapers.repec.org/RePEc:zbw:cefswp:200915.

[50] A. G. Doig A. H. Land. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[51] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, February 1991. ISSN 0036-1445. doi: 10.1137/1033004. URL http://dx.doi.org/10.1137/1033004.

[52] F.M. Solomon M. Desrochers. A column generation approach to the urban transit crew scheduling problem. *Transp. Sci.*, 23:1–13, 1989.

[53] E.L. Hahne. Round-robin scheduling for max-min fairness in data networks. *Selected Areas in Communications, IEEE Journal on*, 9(7):1024–1039, Sep 1991. ISSN 0733-8716. doi: 10.1109/49.103550.

[54] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *Networking, IEEE/ACM Transactions on*, 4(3):375–385, Jun 1996. ISSN 1063-6692. doi: 10.1109/90.502236.

[55] B. Alam, M.N. Doja, and R. Biswas. Finding time quantum of round robin cpu scheduling algorithm using fuzzy logic. In *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*, pages 795–798, Dec 2008. doi: 10.1109/ICCEE.2008.89.

[56] J.A. Perez-Espinoza, V.J. Sosa-Sosa, and J.L. Gonzalez. Distribution and load balancing strategies in private cloud monitoring. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2015 12th International Conference on*, pages 1–6, Oct 2015. doi: 10.1109/ICEEE.2015.7357993.

[57] A. Kaur and S. Kinger. Temperature aware resource scheduling in green clouds. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pages 1919–1923, Aug 2013. doi: 10.1109/ICACCI.2013.6637475.

[58] A.V. Karthick, E. Ramaraj, and R. Kannan. An efficient tri queue job scheduling using dynamic quantum time for cloud environment. In *Green Computing, Communication and Conservation of Energy (ICGCE), 2013 International Conference on*, pages 871–876, Dec 2013. doi: 10.1109/ICGCE.2013.6823557.

[59] A. Alnowiser, E. Aldhahri, A. Alahmadi, and M.M. Zhu. Enhanced weighted round robin (ewrr) with dvfs technology in cloud energy-aware. In *Computational Science and Computational Intelligence (CSCI), 2014 International Conference on*, volume 1, pages 320–326, March 2014. doi: 10.1109/CSCI.2014.62.

[60] Rajesh Raman, Miron Livny, and Marv Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138. ISSN 1573-7543. doi: 10.1023/A:1019022624119. URL `http://dx.doi.org/10.1023/A:1019022624119`.

[61] R. Raman, M. Livny, and M. Solomon. Matchmaking: distributed resource management for high throughput computing. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146, Jul 1998. doi: 10.1109/HPDC.1998.709966.

[62] R. Raman, M. Livny, and M. Solomon. Resource management through multilateral matchmaking. In *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*, pages 290–291, 2000. doi: 10.1109/HPDC.2000.868662.

[63] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521825830.

[64] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. ISBN 0444527265.

[65] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Inc., New York, NY, USA, 1989. ISBN 0-471-92146-7.

[66] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. ISSN 0036-8075. doi: 10.1126/science.220.4598.671. URL http://science.sciencemag.org/content/220/4598/671.

[67] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, Sep 1990. ISSN 0018-9219. doi: 10.1109/5.58326.

[68] J. Zurada. *Introduction to Artificial Neural Systems*. West Publishing Co., St. Paul, MN, USA, 1992. ISBN 0-314-93391-3.

[69] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evol. Comput.*, 8(2):125–147, June 2000. ISSN 1063-6560. doi: 10.1162/106365600568158. URL http://dx.doi.org/10.1162/106365600568158.

[70] State of the art for genetic algorithms and beyond in water resources planning and management. *Journal of Water Resources Planning and Management*, 136 (4):412–432, 2010. doi: 10.1061/(ASCE)WR.1943-5452.0000053. URL http://dx.doi.org/10.1061/(ASCE)WR.1943-5452.0000053.

[71] Hosein Azarbonyad and Reza Babazadeh. A genetic algorithm for solving quadratic assignment problem(qap). *CoRR*, abs/1405.5050, 2014. URL http://arxiv.org/abs/1405.5050.

[72] H. Tamaki, E. Nishino, and S. Abe. A genetic algorithm approach to multi-objective scheduling problems with earliness and tardiness penalties. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, page 52 Vol. 1, 1999. doi: 10.1109/CEC.1999.781906.

[73] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984. ISSN 0163-5808. doi: 10.1145/971697.602266. URL http://doi.acm.org/10.1145/971697.602266.

[74] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL http://doi.acm.org/10.1145/361002.361007.

[75] H.J Bremermann. The evolution of intelligence. the nervous system as a model of its environment. Technical report 1, Dept. Mathematics, Univ. Washington, Seattle, jul 1958. contract no. 477(17).

[76] John H. Holland. Hierarchical descriptions, universal spaces and adaptive systems. Technical report, DTIC Document, 1968.

[77] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0262082136.

[78] Kumara Sastry, David Goldberg, and Graham Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter Genetic Algorithms, pages 97–125. Springer US, Boston, MA, 2005. ISBN 978-0-387-28356-2. doi: 10.1007/0-387-28356-0_4. URL http://dx.doi.org/10.1007/0-387-28356-0_4.

[79] J. Alcaraz and C. Maroto. A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, 102(1):83–109. ISSN 1572-9338. doi: 10.1023/A:1010949931021. URL http://dx.doi.org/10.1023/A:1010949931021.

[80] Optimization of resource allocation and leveling using genetic algorithms. *Journal of Construction Engineering and Management*, 125(3):167–175, 1999. doi: 10.1061/(ASCE)0733-9364(1999)125:3(167). URL http://dx.doi.org/10.1061/(ASCE)0733-9364(1999)125:3(167).

[81] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128, Jan 1986. ISSN 0018-9472. doi: 10.1109/TSMC.1986.289288.

[82] SHYUE-JIAN WU and PEI-TSE CHOW. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engineering Optimization*, 24(2):137–159, 1995. doi: 10.1080/03052159508941187. URL http://dx.doi.org/10.1080/03052159508941187.

[83] P.T. Endo, A.V. de Almeida Palhares, N.N. Pereira, G.E. Goncalves, D. Sadok, J. Kelner, B. Melander, and J.-E. Mangs. Resource allocation for distributed cloud: concepts and research challenges. *Network, IEEE*, 25(4):42–46, July 2011. ISSN 0890-8044. doi: 10.1109/MNET.2011.5958007.

[84] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. *Algorithms and Architectures for Parallel Processing: 10th International Conference, ICA3PP*

2010, Busan, Korea, May 21-23, 2010. Proceedings. Part I, chapter InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, pages 13–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13119-6. doi: 10.1007/978-3-642-13119-6_2. URL http://dx.doi.org/10.1007/978-3-642-13119-6_2.

[85] Mohammad Mehedi Hassan, M. Shamim Hossain, A. M. Jehad Sarkar, and Eui-Nam Huh. Cooperative game-based distributed resource allocation in horizontal dynamic cloud federation platform. *Information Systems Frontiers*, 16(4):523–542, 2012. ISSN 1572-9419. doi: 10.1007/s10796-012-9357-x. URL http://dx.doi.org/10.1007/s10796-012-9357-x.

[86] M. Alicherry and T.V. Lakshman. Network aware resource allocation in distributed clouds. In *INFOCOM, 2012 Proceedings IEEE*, pages 963–971, March 2012. doi: 10.1109/INFCOM.2012.6195847.

[87] Jeffrey Shneidman and David C. Parkes. *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers*, chapter Rationality and Self-Interest in Peer to Peer Networks, pages 139–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-45172-3. doi: 10.1007/978-3-540-45172-3_13. URL http://dx.doi.org/10.1007/978-3-540-45172-3_13.

[88] Openstack community. Nova documentation, jan 2016. URL http://docs.openstack.org/developer/nova/.

[89] Openstack community. Nova architecture documentation, jan 2016. URL http://docs.openstack.org/developer/nova/rpc.html.

[90] Openstack community. Filter scheduler, jan 2016. URL http://docs.openstack.org/developer/nova/filter_scheduler.html.

[91] Amazon Elastic Compute Cloud Documentation. Regions and availability zones, jan 2016. URL http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[92] VP Amazon Web Services (AWS), James Hamilton and Distinguished Engineer. Aws innovation at scale keynote, nov 2014. URL https://www.youtube.com/watch?v=JIQETrFC_SQ.

[93] AWS re:Invent 2015. It strategy & migration session, oct 2015. URL https://reinvent.awsevents.com/.

[94] VMware company. Vmware: The trusted industry leader, feb 2016. URL `http://www.vmware.com/uk/why-choose-vmware/trusted-industry-leader/`.

[95] Gulati; Holler; Ji; Shanmuganathan; Waldspurger and In. Vmware distributed resource management: Design, implementation, and lessons learned. *VMware Technical Journal*, 2012.

[96] R. ; Montero Moreno-Vozmediano, R. S. and I. M. Llorente. *IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures Computer, IEEE Computer Society.* 45, 2012.

[97] Al-Fares, M. ; Loukissas, A. and Commodity Data Center Network Architecture SIGCOMM Comput Vahdat, A. A Scalable. Commun. *Rev., ACM*, 38:63–74, 2008.

[98] Alizadeh, M. and T. Edsall. *On the Data Path Performance of Leaf-Spine Datacenter Fabrics 2013 IEEE 21st Annual Symposium on High-Performance Interconnects.* 2013.

[99] Greenberg, A. ; Lahiri, P. ; Maltz, D. A. ; Patel, P. and S. Sengupta. *Towards a Next Generation Data Center Architecture: Scalability and Commoditization Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow, ACM.* 2008.

[100] Duffield, N. G. ; Goyal, P. ; Greenberg, A. ; Mishra, P. ; Ramakrishnan, K. K. and J. E. van der Merive. A flexible model for resource management in virtual private networks SIGCOMM comput. *Commun. Rev., ACM, 1999*, 29:95–108, 1999.

[101] Sivakumar, R. ; Das, B. and V. Bharghavan. *Spine Routing in Ad Hoc Networks ACM/Baltzer Cluster Computing Journal (special issue on Mobile Computing.* 1, 1998.

[102] B. & Rasmussen Rimler. *N. Mobile data center Google Patents.* Google, 2006.

[103] D. & Stewart Hart. *N. Mobile data center Google Patents.* Google, 2009.

[104] W. & Aigner Whitted. *G. Modular data center Google Patents.* Google, 2007.

[105] D.S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 79–89, Oct 1985. doi: 10.1109/SFCS.1985.63.

[106] Antoniou A and Iosup A. *Performance evaluation of cloud infrastructure using complex workloads.* 2012.

[107] Suganya Sridharan. *A Performance Comparison of Hypervisors for Cloud Computing*. University of North Florida, 2012.

[108] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6 (2):182–197, Apr 2002. ISSN 1089-778X. doi: 10.1109/4235.996017.

[109] Kalyanmoy Deb and Samir Agrawal. *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Portorož, Slovenia, 1999*, chapter A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms, pages 235–243. Springer Vienna, Vienna, 1999. ISBN 978-3-7091-6384-9. doi: 10.1007/978-3-7091-6384-9_40. URL `http://dx.doi.org/10.1007/978-3-7091-6384-9_40`.

[110] A. Antoniou. Performance evaluation of cloud infrastructure using complex workloads. Master's thesis, Electrical Engineering, Mathematics and Computer Science, mar 2012.

[111] Sanjay P. Ahuja and Suganya Sridharan. Performance evaluation of hypervisors for cloud computing. *Int. J. Cloud Appl. Comput.*, 2(3):26–67, July 2012. ISSN 2156-1834. doi: 10.4018/ijcac.2012070102. URL `http://dx.doi.org/10.4018/ijcac.2012070102`.