POLITECNICO
MILANO 1863

Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Optimization models and heuristics for Virtual Machine placement and migration

Relatore:
Prof. Edoardo Amaldi — Politecnico di Milano
Correlatore:
Prof. Danilo Ardagna — Politecnico di Milano

Tesi Magistrale di:
Marco Romani
Matricola 852361

Anno Accademico 2016-2017

Alla mia famiglia.

Ringraziamenti

Un sentito ringraziamento al Professor *Edoardo Amaldi* e al Professor *Danilo Ardagna* per il costante supporto e disponibilità mostrati nell'arco degli ultimi sei mesi. Mai avrei pensato di uscire dal Poli, ormai vuoto, alle 20.15 o di collegarmi via skype con il mio relatore pur di rispettare l'incontro settimanale. Grazie davvero.

Un grazie alla mia famiglia. A mia mamma, instancabile supporto e punto riferimento nei momenti di bisogno. A mio papà, che non ha mai smesso di darmi consigli anche quando non sapeva nulla dell'argomento.

Un grazie ad *Andrea* e *Roberto* per la compagnia intellettuale, ma anche un grazie a *Emanuele* per quella un po' meno intellettuale.

Un grazie ad *Alessandro* e ad *Angelo*, le persone vanno e vengono ma loro sono sempre lì.

Infine un grazie a *Rosa*, che non si è ancora arresa.

Contents

List of Figures

List of Tables

Abstract

Virtualization is a key component for the deployment of Internet services and applications nowadays. The main players in this context are cloud data center providers that offer the infrastructure of their data centers to customers who need their applications to be hosted by a third party. A cloud provider is interested in finding the most efficient way of allocating its resources to customers in order to be able to maximize its profit. This concern consists in finding the optimal placement of Virtual Machines (VMs) on physical servers. Such placement may need to change over time due the system's evolution, thus requiring a constant effort during the entire lifetime of a VM and of the data center itself.

We define two problem variants for VM placement and VM migration, with particular focus on traffic-awareness. The VM placement problem consists in, given a data center and a set of new VMs, assigning each VM to exactly one server, while satisfying all resource capacities. Instead, the VM migration problem consists in periodically updating the placement of a set of already existing VMs without violating any resource capacity, including the bandwidth capacities of the network links. Traffic-aware optimization constitutes a currently popular trend in the field due to the widespread presence of highly intercommunicating multi-tier applications that seriously threatens to be a bottleneck for the scalability of modern data center networks.

We provide two Mixed Integer Linear Programming (MILP) formulations to solve the two problems. Then, since both the problems are NP-hard, to tackle larger instances we develop two Greedy Adaptive Search Procedures (GRASP). In particular, the heuristic devised for the VM placement problem consists in a greedy randomized procedure followed by a local search procedure. Instead, the heuristic devised for the VM migration problem includes, beside its own greedy randomized and local search procedures, an adaptation of the Path Relinking technique.

In order to determine the quality of the solutions found by the two heuristics, they are compared to the solutions obtained by solving the MILP formulations with the state of the art solver CPLEX on instances with up to 128 servers for the VM placement and 54 servers for the VM migration , leading to promising relative gaps from the optimal solution of approximately 0% and 8% respectively. The two heuristics are also tested on instances with up to 10000 servers, on which we show the incremental benefits brought by the proposed techniques and refinements. Finally, as previously done in the literature, we provide a comparison with a First Fit heuristic. Instances are generated with data derived from real sources, benchmarks and VM vendors.

Sommario

Al giorno d'oggi la virtualizzazione è una componente chiave per la distribuzione di servizi Internet e applicazioni. I principali attori in questo contesto sono i provider di cloud data center che offrono l'infrastruttura dei loro data center a clienti che necessitano di una parte terza per ospitar la loro applicazione. Un cloud provider ha interesse nel trovare il modo più efficiente di allocare le sue risorse ai clienti in modo da massimizzare il suo profitto. Questa preoccupazione consiste nel trovare il collocamento ottimale di Virtual Machine (VM) su server fisici. Col passare del tempo, tale collocamento potrebbe richedere di essere cambiato a causa dell'evoluzione del sistema, rendendo così necessario uno sforzo costante per tutto il tempo di vita di una VM e dello stesso data center.

Definiamo due varianti di problemi relativi al collocamento di VM e alla migrazione di VM, con particolare attenzione alla gestione del traffico. Il problema del collocamento di VM consiste nel, dato un data center e un insieme di nuove VM, assegnare ogni VM esattamente ad un server assicurandosi di soddisfare tutti i vincoli sulle risorse. Invece, il problema della migrazione di VM consiste nell'aggiornare periodicamente il collocamento di un insieme di VM già esistenti senza violare alcuno vincolo sulle risorse, tra cui i vincoli di banda sui link della rete. L'ottimizzazione attenta al traffico costituisce un trend recentemente popolare in questo campo grazie alla presenza diffusa di applicazioni multi-tier dall'intenso traffico interno che minacciano di essere un collo di bottiglia per la scalabilità delle reti dei moderni data center.

Forniamo due formulazioni di Programmazione Lineare Misto-Intera (MILP) per risolvere i due problemi. In seguito, poichè entrambi problemi sono NP-difficili, per affrontare istanze di dimensione maggiore sviluppiamo due algoritmi di tipo Greedy Adaptive Search Procedure (GRASP). Nello specifico, l'euristica sviluppata per il problema di collocamento consiste in una procedura greedy randomizzata seguita da una procedura di ricerca locale. Invece, l'euristica sviluppata per il problema di migrazione include, oltre alla sua procedura greedy randomizzata e alla sua ricerca locale, una procedura aggiuntiva che implementa la tecnica del Path Relinking.

Per determinare la qualità delle soluzioni trovate dalle due euristiche, queste sono confrontate con le soluzioni ottenute risolvendo le formulazioni MILP con il risolutore CPLEX su istanze con al massimo 128 server per il problema di collocamento e 54 server per il problema di migrazione, portando a promettenti gap relativi nei confronti della soluzione ottima di circa 0% e 8% rispettivamente. Le due euristiche sono anche testate su instanze fino a 10000 server, sulle quali mostriamo i benefici incrementali delle tecniche e dei miglioramenti proposti. Infine, come fatto in precedenza nella letteratura, forniamo un confronto con una euristica First Fit. Le istanze sono generate con dati ottenuti da fonti reali, benchmark e fornitori di VM.

CHAPTER 1
Introduction

1.1 Motivations

Virtualization is the key to understand the widespread appearence of Internet services and cloud computing in today's world. The ability to separate the logical aspect of an application from the bare metal of the substrate machine on which it is running opens a whole new set of possibilities with respect to the past. The market of Internet services is not restricted to few entities that have the resources to host them. Instead, service providers and owners of generic applications may rent physical resources from cloud providers in a business model called *Infrastructure as a Service* (Iaas). All goes in favor of the end users that have access to a large variety of different services among which they can choose what better suits their needs.

This business model promotes the growth of data centers, i.e., facilities that contain large collections of server farms and exploit economy of scale. Modern data centers make intense use of virtualization techniques to host customer's applications in a transparent way. The customer selects a set of Virtual Machines (VMs), usually the ones that best match their application's requirements, from the cloud provider's catalog. Then, it is up to the cloud provider to decide how to allocate them inside the data center to achieve high utilization of resources while at the same time guaranteeing the Quality of Service (QoS) agreements with the customers. The strength of this business model relies on the fact that a data center acts as a virtually unlimited pool of resources that are given to applications depending on their needs. Then, when a resource is not needed anymore, it is released and goes back to the resource pool, ready for being reassigned.

Amortized cost	Component	Sub-components
~45%	Servers	CPU, memory, storage systems
~25%	Infrastructure	Power distribution and cooling
~15%	Power draw	Electrical utility costs
~15%	Network	Links, transit, equipment

Table 1.1: *Costs in a data center [1].*

To be able to maintain this kind of infrastructure, cloud providers have to face significant costs. Depending of their size, data centers can accomodate up to dozens, or even hundreds, of thousands of servers. This reflects in an electricity consumption comparable to a small town. According to [1], a mega data center's annual power consumption can cost up to 10 million dollars on average. This number is probably outdated today. According to Raritan Inc. [5], one of the leading providers in power managements solutions for data centers, among the top five challenges for data center managements we can find:

- Improving Utilization of Capacity (Power, Cooling, Space),
- Managing Energy Usage & Costs.

The energy is mainly consumed by servers and network switches. The power consumption of a server depends on his workload and can range from approximately 50W when idle to 300W-400W at full load. A lot of effort has been put in the past to lower the power consumption of an idle machine. The power consumption of a network switch depends on the volume of traffic that it is able to route. For example, a 24-port switch connected to Gigabit links can require up to 15W, and often a large data center is equipped with larger switches connected to 10-Gigabit links. Additionally, there are significant periodic costs devoted to servers' and network's maintainance. ICT technology evolves constantly and deteriorates quickly, requiring periodical upgrading or fixing. Therefore, while turning on a server brings an energy cost, the cloud provider would prefer to have it running and generating the profit that it was paid for. Table 1.1, taken from [1], provides an overview of the aggregated costs of a data center.

In order to overcome these significant costs, the task of finding the most profitable resource allocation plays a key role. A commonly used metric to describe this idea is the data center Power Usage Efficiency (PUE) which is defined as:

$$PUE = (TotalFacilityPower)/(ITEquipmentPower)$$

According to [1], a state of the art facility will typically attain a PUE of approximately 1.7, which is far below the average of the world's facilities but more than the best. Inefficient enterprise facilities will have a PUE between 2.0 and 3.0. This means that, just through a more careful choice of resource allocations, power consumption could be reduced by 50%, maybe even more, leading to significant profits for the cloud providers and to infrastructures more sustainable from an environmental perspective.

The efficiency/inefficiency described by the PUE index can be explained by the phenomenon of fragmentation. A poor resource allocation causes servers capacities to be inefficiently used, i.e, a fraction of the virtual pool of resources cannot be used because it is distributed among hosts in fragments too small to accommodate anything. Fragmentation can also cause applications to be unnecessarily split among servers. Ideally the network should be such that any server has the same distance from all the others, but in reality network topologies cannot guarantee such property. Therefore, splitting an application organized in multiple tiers can cause the application traffic to travel through more links and switches than necessary, causing unnecessary links' usage, more switches' power consumption and ultimately more delays for the end-user. Inefficiency in resource utilization also translates in more power consumed by the cooling system, whose goal is to reduce the machines' temperature to avoid system failures. This quantity is quite relevant. Accordingly to [1], out of each watt delivered, approximately 59% goes to the IT equipment, 8% goes to power distribution loss, and 33% goes to cooling. Improving resource allocation would allow for lower cooling costs as well as more applications served at the same power costs.

Seagate Technology PLC, one of the companies leader in data storage, provides another list containing the top 10 challenges and priorities for data center operators [6], including:

- Increasing energy costs,
- High energy consumption,
- Network clogging and connectivity issues,
- System scalability.

The emphasis is put not only on energy-related costs, but also on the difficulty in realizing larger and larger data centers able to keep up with big-data trends. In particular, cloud providers are interested in accepting as much requests as possible. Given that nowadays resources as computing power and memory are available in generous quantities, the most critical type of resource that can constitute a bottleneck for the scalability of modern data centers is the bandwidth

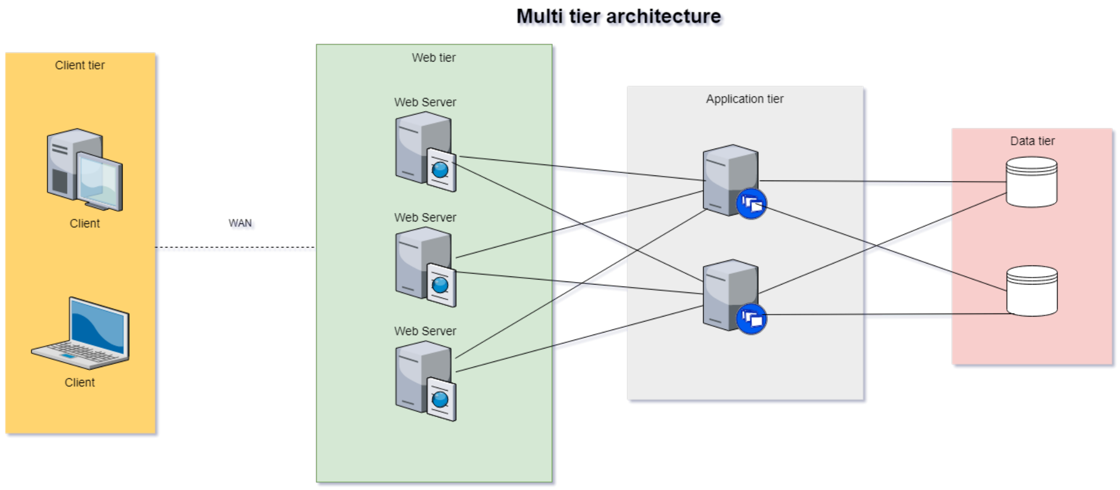


Figure 1.1: Three-tier web application scheme.

utilization. In fact, in today's wide spectrum of applications, the most common ones are three tiers web applications (see Figure 1.1) that require intensive communication from and towards the Internet as well as among themselves. While data centers employ robust network topologies to partially overcome the issue, they are not enough and a careful optimization planning is needed to avoid the saturation of network links.

This focus of optimizing the network aspect goes under the name of traffic-aware optimization. Such scenario, and the optimization problem related to it, have been known and studied for quite some time now. However, technologies involved in this context are always changing and evolving and the techniques used to solve the problem can be refined to account for new factors and priorities that arise in the real world.

1.2 Thesis organization

This thesis is organized as follows.

In Chapter 2 we briefly discuss some relevant technological aspects, such as VM consolidation via live-migration, differences between container and VM technology and most frequently used network topologies. Then, we summarize a selection of previous works about the optimization problems of VM placement and VM consolidation via live-migration. The VM placement problem consists in determining the optimal allocation, in terms of traffic management, of a set of new VMs, while the VM consolidation problem consists in periodically updating, through the tool of VM migration, the allocation of some VMs already running in order to minimize the energy costs of the data center.

In Chapters 3 and 4 respectively we present two variants of the two problems previously introduced: the short-term VM Placement Problem (VMPP) and the long-term VM Migration Problem (VMMP). The two are tackled independently, but are meant to coexist. For each of them, we provide a Mixed Integer Linear Programming (MILP) formulation.

In Chapter 5 we describe two Greedy Randomized Adaptive Search Procedures (GRASP) that we developed to find approximate solutions of the two problems. The VMPP's heuristic is composed by a greedy randomized procedure, that builds an initial solution, and by a local search procedure that starts from such solution and looks for improvements until it reaches a local minimum. The VMPP's heuristic is composed by a greedy randomized procedure, a local search procedure and an additional Path Relinking procedure that builds new solutions starting from a pool of solutions obtained by the local search.

In Chapter 6 we describe how we derived the instances used in our tests from real-world data, benchmarks and VM vendors.

In Chapter 7 we report the computational results of the comparison between the quality of the solutions found by our two heuristics and the optimal solutions obtained by solving the MILP formulations with the state of the art solver CPLEX for instances with up to 128 (VMPP) and 54 servers (VMMP). Larger instances with up to 10000 servers are also tested.

In Chapter 8 we draw some conclusions and mention some possible future works.

The Appendix contains some details that were omitted or shortened for readability throughout the whole thesis, as well as a recap of the more refined versions of the MILP formulations and some UML class diagrams explaining how the code of our heuristics is structured and organized, so that anyone willing to adopt it or extend it is offered all the tools to do it.

CHAPTER 2

State of the Art

In this chapter we analyze the state of the art related to resource management in cloud data centers. Section 2.1 is devoted to some relevant technology solutions and tools that are available to cloud providers and that must be considered when tackling any problem related to this field. Section 2.2 is devoted to the two main optimization problems we are interested in: VM placement and VM migration. We summarize a subset of relevant works in the literature from which we take inspiration.

2.1 Technology review

In this section we summarize some of the most successful technical solutions adopted in modern data centers. These solutions are fundamental to understand some of the key aspects and decisions that we are going to make when modeling the problems later on.

Section 2.1.1 introduces the VM technology, which constitutes the core of the whole thesis. Section 2.1.2 is devoted to the description of how VM migration is implemented, which are the tools that make it effective, which are the goals that drive the choice of the specific techniques that can be used. Section 2.1.3 discuss the benefits of containers technology that is recently raising in popularity as an alternative tool to implement virtualization. We go through a direct comparison with the classical VM technology, pointing out advantages and drawbacks of both. Finally, Section 2.1.4 contains an overview on the three most commonly used network topologies in modern data centers.

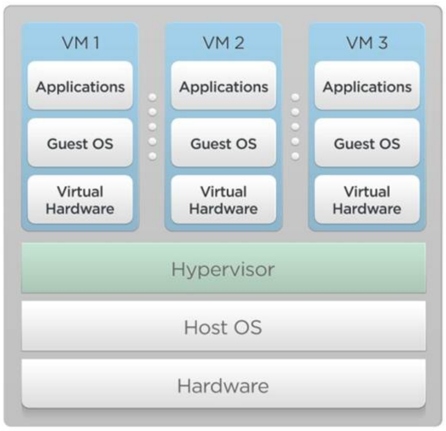


Figure 2.1: *Virtual Machine technology.*

2.1.1 Virtualization and Virtual Machines

Virtualization technologies allow to separate applications from the systems on which they are running treating them as two distinct independent entities, with evident benefits in terms of application management, portability and deployment. Historically, the most successful virtualization technology is the Virtual Machine (VM) technology. A VM is a software environment that, from the point of view of an application, acts exactly as a physical machine. It features its own Operating System that runs on top of the OS of the physical machine, including a virtual storage, a virtual processor and a virtual memory space. This is possible thanks to a hypervisor, i.e., a software that operates on a level between the physical machine and the VM, as shown in Figure2.1. A hypervisor, e.g., Xen, manages and coordinates all the different VMs running on top of it, making sure that each one of them is allocated the proper amount of physical resources at any given time. Indeed, a VM is supposed to be transparent to other VMs running on the same system, i.e., it should be isolated from anything outside it and its execution should not be influenced by the behavior of other VMs.

2.1.2 VM Live-Migration

Thanks to virtualization, cloud providers have the possibility not only of choosing the most convenient allocation of VMs, but also of changing VMs deployments whenever a more convenient configuration becomes available. This eventuality is intrinsic in the nature of an online ever-changing system and needs to be considered to achieve good results in the profit maximization. While offline migration, i.e., turning off a VM on a host and booting an identical new one on another host, is a possibility, the most appealing option is to be able to migrate a VM's memory and execution state through the network towards the new selected host while the VM is running. This way, the end-user does not lose its session at any point of the process. Two main different techniques exist in the context of VM Live-Migration: post-copy migration and pre-copy migration [7]. Each of them is designed to strive towards a particular goal, with the two goals being in contrast with each other.

Post-copy migration chooses to immediately switch the execution of the application to the new selected machine. The new hosts will not have yet the correct memory pages and will need to fetch them from the original host when required. This implies that there will be a period right after migration in which the end-user will perceive a significant downgrade of application performance until the VM has recovered enough memory pages to work correctly. The advantage of this approach is that memory pages are fetched only once and the overall migration process have minimum predictable duration, even if the user experiences full perception of this duration. Since, apart from the migration traffic, the network has already to sustain the normal traffic loads of VMs that do not migrate, this advantage can be relevant.

On the other hand, pre-copy migration sends the memory pages to new host first, and then switches the execution context of the application from the original machine to it. This way, the end-user perceives minimal performance downgrade, almost none. However, since memory pages are sent when the original VM is still running, many of them can become incorrect shortly after, requiring a longer process of synchronization where modified pages are continuously sent until a certain level of stability is reached. Only at that point the execution context is switched. This implies more pages sent than the actual size of a VM state and more time spent overall. Also, pre-copy migration allows the possibility to safely recover from a failure during migration, whereas post-copy migration does not.

If robustness and user experience are mandatory and the system offers enough network capacity, pre-copy migration is the correct option despite of its limits. In all other cases post-copy migration is preferable due to predictability, short duration and low resource usage. In particular tests run in [7] show how pre-copy migration consistently takes up 600% of the time used by post-copy migration, if not more.

2.1.3 Containers vs VMs

In recent years, containers have grown in popularity as a way to virtualize customers' applications in a cloud environment, the reason being mainly their

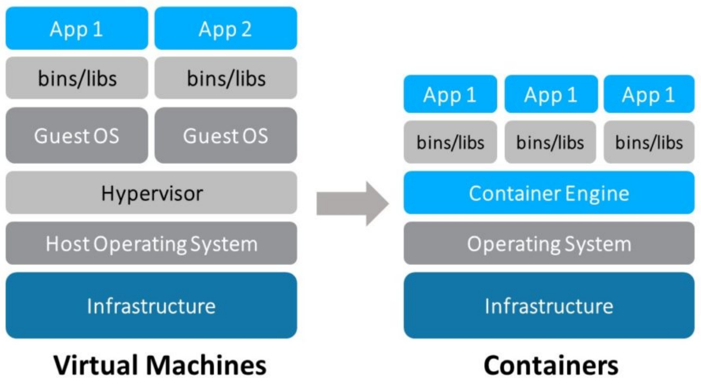


Figure 2.2: *Virtual Machine technology vs Container technology.*

lightweight nature with respect to classic VMs. A comparative study of Containers with respect to Virtual Machines is reported in [8].

Differently from a VM, a container does not need to be wrapped in a virtual OS, but it only requires to load some dependencies to be run on the underlying OS of the physical machine (see Figure 2.2) with the help of a Container Engine, such as Docker. Even if this comes at the cost of lower security and isolation for the applications (they do not have their dedicated virtual CPU and memory and are not wrapped in an isolated environment), the advantages are definitely relevant. First of all, the lack of a virtual OS means that the impact of a container on a server is much lower. Consequently the average number of containers that a single machine is able to host is higher with respect to VMs. Moreover, to launch a container, there is no need to retrieve the correct OS image from a disk, boot the OS and finally have it online. This apparently minor aspect allows to entirely avoid the problem of optimal VM images placements [9]. Therefore, the overhead for turning on/off a container is much lower with respect to a VM and the operation can be performed in a matter of seconds instead of minutes. Also, even if, as of today, containers' migration is not mature yet [8], the advantages seems promising. Migrating a container does not require to migrate the whole OS's state together with the application's state, while also loading the correct OS image on the new destination. Thus, even though a mature technology like the VMs pre-migration technique [7] is not available for containers, the much lower impact on physical links together with lower migration time and lower booting overhead vastly overcome the reduction in applications' downtime achieved by VMs pre-migration. Currently, containers migration can be implemented through the tool of Chekpoint/Restore in Userspace (CRIU), i.e.,

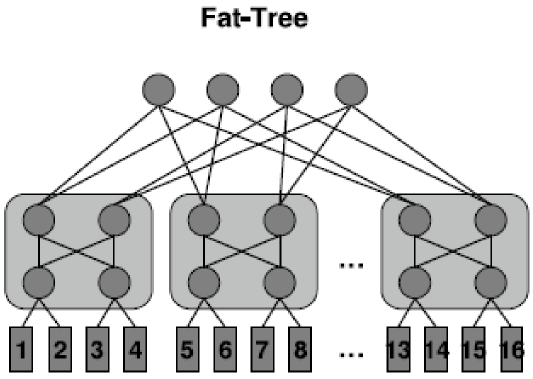


Figure 2.3: *Fat-Tree topology.*

freezing a running application and checkpoint it to persistent storage as a collection of files that can be later used to restore and run the application from the point it was frozen at.

2.1.4 Network topologies

The network is the core of a data center. Not any arbitrary kind of network is able to sustain the business of a cloud provider. A certain number of properties have to be fulfilled. The architecture has to provide a good scalability level to be able to face incremental data centers. For this reason, the successful topologies are organized in different levels of servers groups. Typically a handful of machines is connected to a common network switch called Top of Rack (ToR). This group of hosts, called *rack*, is then connected to other racks through one or two levels of aggregation nodes in such a way that the distance in terms of nodes among different groups of servers is always the same. This allows to partially hide the physical structure of the network and to make the geographical position virtually irrelevant, thus abstracting the distributed resources into a pool of resources. In case of a Fat-Tree topology , racks are themselves grouped into *pods*, and the equidistance property holds both among pods and among racks inside each pod, as well as among hosts inside each rack.

Figure 2.3 illustrates a Fat-tree topology with four core nodes connected to three pods. Each pod is constituted by two aggregation nodes and two edge nodes forming a bipartite graph. Each edge node at the bottom is connected to two servers constituting a rack. The main advantage of a Fat-tree topology with respect to a trivial Tree topology is that the bipartite graph in the intermediate

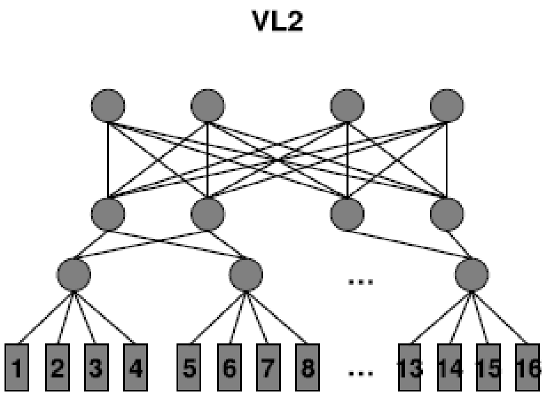


Figure 2.4: *VL2 topology.*

level provides more robustness in case of a link failure that would otherwise isolate a portion of the network.

Instead, the VL2 topology differs from the Fat-Tree for the organization of aggregation links/nodes and core/nodes. While in the latter each pod is organized with an internal bipartite graph among aggregation and edge (ToR) nodes, with the core nodes being outside, the VL2 topology is characterized by a bipartite graph among the entirety of core nodes and aggregation nodes. Figure 2.4 shows a VL2 topology featuring a bipartite node of four core nodes and four aggregation nodes. Such structure leads to a third level with three edge ToR nodes that are directly connected to the servers. The advantage of VL2 over Fat-Tree is that VL2 is designed to implement valiant load balancing, i.e., traffic generated by the edge nodes is forwarded first to a randomly selected core switch and then back to the actual destination. The rationale behind it is that when traffic is unpredictable the best way to balance load across all available links is to randomly select a core switch as an intermediate destination.

Finally, the Bcube topology is a more recent architecture that has its own peculiarity in a multi-level recursively-defined structure. Instead of being all located at the bottom level, servers are spread among the multi-level structure. Figure 2.5 shows a trivial Bcube topology with only one level of Bcubes, each Bcube containing $n = 4$ servers. In case of more levels, each Bcube would be connected to n other Bcubes in the second level, and so on. The advantage of such topology is the possibility to merge the hosts into the network infrastructure, having them acting also as routing nodes and forwarding packets on behalf of other servers.

Other than equidistance, each of these topologies is able to provide high con-

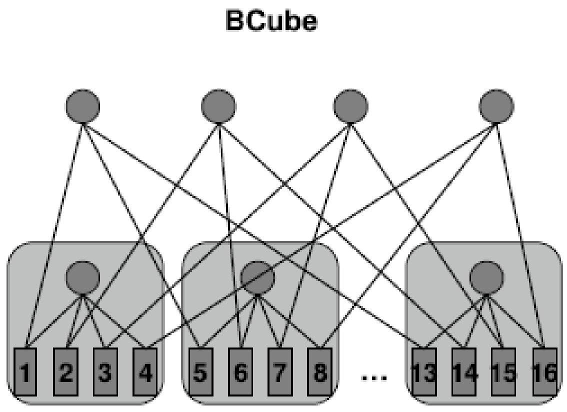


Figure 2.5: *Bcube topology.*

nectivity, meaning that each server pair can communicate at full bandwidth. To make this possible, the topology must be able to offer different alternative paths between server pairs in order to be as robust as possible towards occasional failures, congestion or traffic peaks occurring in any part of the network. To fully achieve this type of ideal property, however, the network topology itself is not enough and a second step in the form of traffic-aware optimization, which is the focus of this thesis, is needed.

2.2 Previous works

In the literature we found two main areas of research about data center optimization:

- VM placement optimization, which is about deciding the best allocation of new VMs each time a group of new requests arrives to the cloud provider,

- VM migration optimization, which is about deciding how to migrate VMs in order to apply a partial reconfiguration of the system and release the load on some resources to avoid bottlenecks.

In the latter case, often the optimization problem involves only the migration process of pre-selected VMs, leaving the issue of selecting which VMs need to be migrated to a pre-processing phase.

2.2.1 VM Placement Problem

The VM Placement Problem in general is defined as the problem of finding a suitable assignment between a set of VMs and a set of servers. At the core it is a generalization of the Bin Packing Problem, from which it inherits the overall structure, while adding particular features specific to the context. For example, typically each server is modeled as a multi-dimensional knapsack with constraints on four different dimensions: CPU, RAM, disk and network bandwidth. The bandwidth constraints can be extended to links, thus including a Virtual Network Embedding (VNE) problem inside the formulation. Moreover, different variants of the problem are defined through the choice of the objective function. Nowadays, with the growth of intra-application traffic it is a common idea that good solutions should aim for minimizing the traffic overhead in the whole network. Indeed, VMs and applications come with a traffic demand matrix whose corresponding overhead heavily depends on the allocation chosen by the cloud provider. Also, other than links utilization and traffic delays, servers in a data center consume an amount of energy dependent on their workload, and a proper allocation can help in minimizing this cost. VM placement is a known and well studied issue with several different approaches proposed in the literature. An overview on the related research area can be found in [10] and [11]. We are interested in works that cover the optimization problem related to VM placement, and in the following paragraphs we summarize the main contributions to which we refer.

In [12], Meng *et al.* consider the VM placement problem with a strong emphasis on the network constraints. Their version of the problem is called Traffic-Aware Virtual Machine Placement (TVMPP) and its peculiarity relies in the objective function that they decided to use:

$$min \sum_{i,j=1,..,n} D_{ij} C_{\pi(i)\pi(j)} + \sum_{i=1,..,n} e_i g_{\pi(i)} \tag{2.1}$$

where:

- n is both the number of VMs and VM *slots* considered, a *slot* being an atomic amount of resources in their model,
- π denotes a permutation function that creates valid VM-slots assignments, i.e., a function that assigns each VM to a valid location (slot),
- C is the communication cost matrix among servers,
- D is the traffic demand matrix among VMs,

- vector g is a cost vector between servers and the Internet,
- vector e is the traffic demand vector between VMs and the Internet.

Due to the regularity of common data centers network architectures, authors suggest that the mutual traffic among VMs pairs contributes to the global traffic load by a factor depending on, other than the traffic D volume itself, a distance measure C between the two. This distance is trivially the number of hops appearing in the shortest path between two VMs. In fact, the longer the path used by a communication, the higher the number of links on which the traffic is sent. Also, the second term of the objective function is shown to be a constant value, due to the identical distance between each server and the Internet in all network architectures. Therefore, it can be omitted. Another peculiarity of their formulation is that they removed the bin packing nature of the problem by simply considering the data center machines as collections of identical CPU/RAM slots to which any VM can be assigned. They also excluded the Virtual Network Embedding problem by considering static single-path communications among servers. Despite these simplifications, the problem is not an easy one, being a case of Quadratic Assignment Problem (QAP).

They proceed to show the NP-hardness of TVMPP by reduction from the Balanced Minimum K-cut Problem (BMKP), which is known to be NP-hard. Then, following this reduction, they claim that the problem of finding a VM allocation minimizing the weighted traffic overhead on the data center network is equivalent to the problem of finding an allocation that puts highly communicating VMs near to each other (e.g., in the same rack), therefore minimizing the traffic in the core part of the network.

Given this consideration, authors develop a heuristic algorithm based on a two-phase clustering procedure. This procedure applies separately a clustering over the *slots*, which are their units of placement, that aims to find groups of slots close to each other, then a clustering over the VMs with the goal of creating groups of VMs with high intra-group traffic and low inter-group traffic. Finally, given the two sets of clusters, they perform an assignment between the two.

The computational complexity of their heuristic is $O(n^4)$ that, despite being polynomial, is too heavy for computations on realistic instances. However, we find that their choice of the objective function as well as the hops distance measure are very interesting and we decided to use them too. The algorithm is tested on instances of 1024 slots and 1024 VMs and results are compared with other general QAP algorithms such as Local Optima Pairwise Interchange and Simulated Annealing. Different realistic network architectures are used (Fat-tree, VL2, Bcube), while traffic matrices among VMs are randomly generated following the Global Traffic Model, i.e., VMs can send traffic to any other VM.

Another interesting work is [13] by Belabed *et al.* In their version of the VM placement problem they take into account both the traffic demand and the energy consumption simultaneously by minimizing a composite objective function. However, their modeling of the traffic is a bit different from [12] in the fact that, here, the quantity to minimize is not the global weighted traffic itself but rather the upper bound of traffic on any link. This is a natural formulation for avoiding bottlenecks but needs a more accurate modeling of the network elements, making the problem more involved. The energy consumption factor consists trivially in the number of servers that host at least one VM and therefore cannot be turned off.

They propose a mathematical non-linear formulation for the problem that includes two particular features: multipath forwarding and virtual bridging. The first one is done by relaxing the domain of some variables in the formulation and represent the situation in which communications between servers is not forced on a single path but can be split along different ones allowing for a more granular level of decisions. Instead, virtual bridging is the possibility to have servers acting like routing nodes, therefore enabling more routing possibilities.

Then, they developed a repeated matching heuristic that exploits the similarity of the problem with the capacitated facility location problem, in particular the Single Source Facility Location Problem (SSFLP) and analyze the impact of the two features previously mentioned on different network architectures. In the end, they discovered that, when enabled at the same time, the two features enter in contrast with each other. Moreover, activating virtual bridging has an impact only on the traffic-term of the objective function and does not bring benefits from an energy perspective.

Another interesting aspect of this work is the attention put into parameters and data selection, in particular when it comes to VMs inter-traffic. Differently from the previous work, which used the Global Traffic Model, here the authors point out that in the real world VMs communicate only with the Internet or with VMs belonging to the same application/customer. Therefore, the traffic matrix can actually be partitioned into several smaller matrices. In this more reasonable scenario, also the amount of traffic volumes are not completely random. Instead, on average, they follow some predictable (to a certain extent) distributions due to the common structures of Internet applications. We will follow the same lines of reasoning when generating any set of instances.

A more recent work [14] by Larumbe and Sansò constitutes the most similar approach to the problem we are going to face. Their focus is entirely on the VM placement problem, but they claim that their framework is easily extendable to include VM migration. They define three possible inputs to the problem: add a new application (i.e., a set of new VMs), remove an application or resize an application (i.e., add or remove only a part of it). While removing VMs is a straightforward task that does not require particular decisions, the distinction between add and resize reflects on different procedures called when solving heuristically the problem. We will adopt a similar distinction. The authors provide a MIP (Mixed Integer Programming) formulation for the problem. Since the goal is to provide online-decisions, each application request is associated to a time index t. Therefore, solving the problem for time t implies that the solution is based on all the solutions obtained previously for time $t-1$, $t-2$ and so on. This is a quite natural approach that we will also use. However, their formulation considers some aspects in a different manner with respect to us. For example, the authors decide to include the network embedding aspect of the problem, i.e., they define each application as a Virtual Network and explicitly map it to the underlying substrate network of the data center, in a similar way to [13]. We think that this decision brings too much of an impact on a problem that aims to be solved online for large instances. Also, in practice, data center networks architectures have enough link redundancy that, on average, adding few VMs at a time (with respect to the overall system) does not require explicit path/link mapping to avoid exceeding bandwidth capacities.

Moreover, they include a power consumption term in the objective function. We decided to leave this aspect out of the online short-term decision process and consider it only on the longer time window when a more significant number of VMs/containers is involved. The idea is that we take a more meaningful power-aware decision that stays relevant for the entirety of the next long time interval. This way, VM placement is driven only by traffic-aware metrics, making the small decisions more consistent. Also, the traffic-aware metric already gives incentives for packing groups of VMs/containers on machines close to each other, or even on the same machine, without unnecessarily turning on too many servers. Finally, the authors consider also workload variations. They use a penalization term in the objective function to avoid to fully load each rack of servers. Thanks to this term, the obtained allocations are more stable during workload peak periods and subsequent time intervals.

After the MIP formulation, Larumbe and Sansò provide a two-phase heuristic to efficiently solve large instances of the problem. The algorithm is composed of a greedy procedure that quickly determines an initial solution and a Tabu Search procedure that repeatedly explores the neighborhood of the last best-solution found and selects the next one to explore. The neighborhood used consists of solutions that differ from the initial one for the placement of a single VM. Implementing a Tabu Search instead of a regular Local Search helps in avoiding to get stuck in early local minima far from the optimal solution.

Since authors are particularly concerned with the online efficiency of the solution, they run their tests on very large instances (up to 128000 servers) and make comparisons with other naive approaches used in practice, in particular with the First Fit algorithm. The results show significant advantages in terms of traffic delay of applications and links' utilization and minor advantages in terms of power consumption. Workloads data are taken from a real data center, but only one VM model and one server model are used, the latter with two different power configurations.

2.2.2 VM Migration Problem

VM live-migration is generally defined as the problem of moving some VMs from their current locations to other servers. The idea is that, after a small dowtime of the involved applications, VMs can return to their normal behavior in a transparent way. Therefore, booting a new VM on a different machine is not enough, because the state of VM before migration has to be sent throughout the network as well. Since it is not reasonable nor realistic that a data center would build an entire separate network just for these migrations, the problem requires to integrate the process with the normal traffic traversing the network, exploiting efficiently the residual link capacities.

Often the VMs to be moved as well as their new selected destinations are pre-computed by other procedures and assumed as input for the problem. If one wanted to include the decision of which VMs to move and where to move them, the already difficult problem would become more challenging. Therefore, the scope of the decision involves only the choice of which links should be used and how to split the traffic load among them in order to move all VMs in the shortest time possible while not saturating the links. This objective comes from the fact that the ultimate goal of live-migration is to be as less noticeable as possible for the end-user, meaning that the average downtime of applications should be minimized, but the cloud provider has also to make sure that the applications that are not involved in the migration process do not experience any delay due to the additional overhead on the network.

As for VM placement, VM migration is a relevant and well studied issue. An overview of the related research area can be found in [15]. Below we summarize two main works that deal with the optimization problem related to VM migration and to which we refer.

In [16], Liu *et al.* face the problem of optimizing live migration of VMs. The goal is to minimize the global migration time of a predefined set of VMs that need to be re-mapped. Their assumption is that, due to constraints on the resources, the best strategy might not be to move all the VMs simultaneously but rather assign them to different migration steps. These steps are defined as time intervals t_0, t_1 .. t_n of predefined length. Therefore, minimizing the global migration time translates into minimizing the number of steps needed to move all the VMs, thus resulting in a sort of scheduling problem. Indeed, they formulate the problem as a Migration Planning problem and provide both a mathematical non-linear formulation and a MILP formulation derived from the linearization of the previous one.

One of the peculiarities of their approach is the idea that the constraints on the resources change significantly throughout the migration process and, therefore, the interactions between resources consumed and/or freed by migrating VMs and resources used by other VMs at every step need to be modeled accurately. We will consider this concept even though our approach does not make use of multiple migration steps.

Unfortunately, the authors do not implement an heuristic on their own and just solve the optimization problem on small instances by applying standard Branch and Bound method and compare the results with a couple of very naive heuristics, in particular a one-by-one migration heuristic that migrates VMs one at a time and a completely random heuristic.

The main non-trivial result that they get is that inter-VM bandwidth requirements have a significant impact on the migration process and cannot be ignored. This confirms the intuition of the authors and the opposite would have been strange. Intuitively, having multiple VMs that suspend their traffic for some amount of time and then recover it elsewhere lately is almost guaranteed to add depth and complexity to the problem, unless the capacities involved are so large that the problem itself becomes trivial. We will follow these insights and include this aspect in our formulation, taking inspiration from some of the linear constraints used here.

In the recent work [17], the authors tackle the VM migration problem with a similar but yet different approach. As in [16], they formulate the problem as a Migration Planning problem with the goal of optimizing the total migration time so that the performance degradation of the system is minimized. Since pre-copy migration is a common standard for VM migration, their model includes parameters regarding the page dirty rate for each VM.

The authors propose their own MILP formulation of the problem. Similarly to [16], the idea is that not all VMs will be migrated simultaneously, but, depending on the state of the network and on the VM requirements such as the different page dirty rates, a proper sequencing of migrations should be chosen to efficiently utilize the network resources. Differently from [16], however, they do not pack groups of migrations into discrete time intervals of fixed duration. Instead, they try to have all the network resources saturated at any given time during the entire process of migration. This way, it becomes more difficult to explicitly define the total migration time, because it cannot be expressed neither as the sum of all single migration times nor as the sum of the duration of fixed time intervals.

To overcome this obstacle, they reformulate the objective function, claiming that minimizing the total migration time is equivalent to maximizing the average transmission rate during the whole migration process. This is justified by the fact that, in general, the migration time of a VM depends only on two factors: the size of the VM's state to be transferred, which is known and fixed, and the transmission rate. Their idea is to repeatedly select groups of VMs that maximize such criterion until no more VMs need to be moved. Therefore, their model has to be recomputed and solved for several iterations, specifically each time any migration comes to an end. Since there are no pre-defined rounds of migration, at each iteration some of the previously started migrations will be completed, while others might still be active and the following batch of migrations have to take their presence into account when allocating the network resources.

Then, since the problem is NP-hard, they propose a fully polynomial time approximation (FPTA) algorithm based on linear approximation, multicommodity flows and Dijkstra Shortest Path algorithm. The algorithm is guaranteed to find a k-approximate solution in $O(\frac{1}{(k-\frac{\sigma}{2})^2} m(m+n\log m)\log n)$ time, where n is the number of nodes, m is the number of links and σ is the accuracy of the underlying linear approximation.

Results are compared with a one-by-one migration heuristic, a grouping-migration heuristic and with the optimal solutions of the MIP on relatively small instances. The FPTA algorithm significantly outperforms the two heuristics and remain close to the MIP solutions, while the computing time required is less than the time needed to solve the LP of the MIP problem. Testbed instances are generated by looking at a real data center's topology and arbitrarily choosing some parameters for servers and VMs. Intra-data center architectures as well as inter-data center architectures are considered, making the algorithm more general and resilient to the choice of different network topologies.

2.2.3 Joint VM Placement and Migration

As far as we know, the authors of [18] are the only ones that try to tackle both the VM Placement Problem and VM Migration problem at the same time. Also, they even include the power consumption aspect of the problem in their formulation. They do so by defining a multi-objective function that minimizes a combination of inter-VM traffic, migration traffic overhead and servers' power consumption. Given the extreme difficulty of solving the overall problem all at once, they opt for a problem formulation that omits several details about both VM placement and migration.

In particular, they propose a simple MILP formulation that models the assignment of new VMs and the re-assignment of VMs already in the system at the same time, making sure that hosts' resources are not saturated, but without explicitly considering the network part of the problem, i.e., capacities on links. Moreover, while the power consumption term in the objective function is modeled quite precisely (our version will be similar), the inter-VM traffic is modeled solely as the amount of traffic traversing the network, without considering any distance measure, and the migration term is very roughly approximated just with the number of VMs selected for migration. Therefore, this type of formulation does not exploit in any way the intuition that highly inter-communicating VMs should be grouped in the same rack. It only recognizes the advantage of putting them in the exact same server. This does not fully address the concern about lowering the traffic overhead in the core part of the network, an issue recognized by many works in the field, including the ones already mentioned. Also, neglecting the modeling of link capacities, at least in the migration half of the problem, is a very strong assumption, given the results achieved by [16] and [17] in this field.

Although the formulation proposed is not very involved, the problem is still NP-hard and the authors solve a relaxation. In particular, they relax some integer variables (used for assignment) to accept values in the $[0,1]$ interval so that the formulation becomes an LP formulation that can be quickly solved with any state of the art solver. Then, they develop a heuristic procedure to build a feasible solution of the original problem starting from a solution of the relaxed one, i.e, a procedure to decide whether a variable in $[0,1]$ should be lifted to 1 or floored to 0 to obtain a feasible assignment.

Their work is validated on instances of relative small size with data centers hosting 250 VMs. However, the VM parameters are taken from a real vendor (Amazon EC2), an approach that we also decide to follow. Due to the lack of both network capacities and distance measures in the formulation, the approach used in this work does not really need to specify any kind of network topology. Results are compared to the ones obtained with a Random heuristic and a First Fit Decreasing heuristic, outperforming them on the quality of the solutions. However, as it can be expected, an approach based on repeatedly solving linear relaxations takes a lot more time than a fast greedy procedure.

Work	VMP	Migr.	Formulation	Heuristic	Test instances
[12]	Yes	No	traffic-aware, no bin packing	clustering, $O(n^4)$	medium size, homog. VMs/ hosts, 3 topologies
[13]	Yes	No	MIP, energy, bin packing, VNE	double-matching	small size, heterog. servers, realistic traffic
[14]	Yes	No	MIP, energy , bin packing, VNE, traffic-aware	greedy+ Tabu Search	large size, homog. VMs and hosts
[16]	No	Yes	MILP, migr. planning	None	small size
[17]	No	Yes	MIP, migr. planning	FPTAS based on LP relaxation	medium size, param. arbitrarely chosen
[18]	Yes	Yes	MIP, no network aspect	rounding procedure after LP relaxation	small size, real VMs

Table 2.1: *State of the art.*

2.2.4 Characterization of this work

Let us summarize in Table 2.1 the contributions brought by the works previously discussed in order to have an overview on the state of the research in this field.

With this thesis, our goal is to tackle both VM placement and VM migration problems, like in [18], in an joint way, providing a detailed MILP formulation for each of them. The formulations will take into account the bin packing nature of the problem as well as a strong emphasis on the network aspect through traffic-aware optimization, without forgetting to look at the power consumption. They will be built putting together the most significant constraints and aspects of the formulations discussed in the revised literature as well as including some new approaches. The models are meant to be directly solved through state of the art solvers only for small-to-medium instances. Then, we aim to solve reasonably large instances of the problems with two ad-hoc heuristics, whose goal is to combine speed and accuracy, like in [14] for the VM placement problem. The heuristic procedures will be tested on instances ranging from small to large sizes, generated with real heterogeneous data about VMs, as in [18], and servers and realistic traffic patterns, like in [13].

CHAPTER 3

VM Placement Problem

This chapter is devoted to a variant of the Virtual Machine Placement Problem, which consists in optimizing the allocation of a new set of VMs on an already running system, with an emphasis on traffic-awareness and bandwidth resources.

In Section 3.1 we describe the context of the problem, mentioning the most relevant aspects and the reasons behind some crucial assumptions. In Section 3.2 we provide a precise definition of the problem, which takes into account the main features of the setting under consideration. All the required parameters are specified, as well as the objective function and the constraints. In Section 3.3 we describe a first nonlinear programming formulation which is useful to understand the meaning of the objective function and the main constraints. In Section 3.4 we derive a Mixed Integer Linear Programming (MILP) formulation. This process is carried out step by step. The final version of the formulation, which can be directly tackled with a state of the art commercial solver, is summarized in the Appendix.

3.1 High level description

We are in the context of an operating data center constantly receiving requests for the allocation of new VMs and/or the shutdown of older ones. The placement of a new VM on a server needs to be performed within a short amount of time, e.g., five minutes, in order to keep up with incoming requests. For this reason, the decision process does not reconsider any allocation done before and considers the already placed VMs as fixed parts of the system. Given these assumptions, the VMs' shutdown becomes trivial and does not actually involve any kind of decision. Therefore, only the placement of new VMs is modeled, while any shutdown request is considered during a of pre-processing phase. We exclude some servers from the decision by looking at their CPU utilization. Only servers with a CPU utilization factor below a certain threshold ρ, e.g., 60%, will be taken into account. This criterion simultaneously reduces the combinatorial complexity of the problem and avoids the saturation of a resource that has a strong impact on servers' performances. The problem consists in finding the VMs placement that minimizes the global traffic load over the network.

Since the set of new VMs represents a very small percentage of the overall system, we do not consider the network as a bottleneck and we do not model it here. Moreover, the impact of the communication between VMs with respect to the traffic load directly depends on the number of hops traversed. For the sake of simplicity, we consider a pre-computed path between every pair of servers for which we know a-priori the cost, namely the number of hops. New and existing VMs are owned by a set of customers.

The focus of this thesis is on VM placement and migration. However, since containers placement differs from VM placement only for the value of some parameters, in the description of the problem we refer to containers.

3.2 Problem description

Given a data center composed by:

- a set of servers $\mathcal{S}$,
- a set of under-utilized servers $\mathcal{S}_u \subset \mathcal{S}$ ($< \rho$ cpu utilization),
- a set of already placed containers $\overline{\mathcal{C}}$,

we need to find a placement for a set of new containers $\mathcal{C}$ owned by a set of customers $\mathcal{R}$. Each container c is characterized by a set of resource demands:

$CPU_{cs}:$ CPU utilization contribution of container c over server s,

$MEM_c:$ RAM demand of container c,

$DISK_c:$ disk IOPS demand of container c,

$d_{cb}:$ outgoing traffic from container c to any other container b,

$d_{bc}:$ incoming traffic to container c from any other container b.

Since some containers communicate with the world outside of the data-center, we use a simple trick to model this factor: we define a special dummy container c_0 that generates and receives all the external traffic, and a special dummy server s_0 equipped with unlimited resources which is meant to only host c_0. For notation purposes, we keep them separated from $\mathcal{S}$ and $\overline{\mathcal{C}}$ respectively. For containers that belong to $\overline{\mathcal{C}}$ the known placement is specified by the following parameter:

$$\overline{x}_{cs} = \begin{cases} 1 & \text{if container } c \text{ is placed on server s} \\ 0 & \text{otherwise} \end{cases}$$

Of course, the assumption is that $\overline{x}_{c_0 s_0}$ is equal to 1 and that the above parameter represents a feasible assignment, meaning that each container is assigned to one and only one server without exceeding any capacity and the only container assigned to s_0 is c_0. Finally, each server s is modeled with a set of residual resource capacities:

$\overline{CPU}_s$: residual CPU capacity of server s,

$\overline{MEM}_s$: residual RAM capacity of server s,

$\overline{BDW}_s^{out}$: residual output bandwidth capacity of server s,

$\overline{BDW}_s^{in}$: residual input bandwidth capacity of server s,

$\overline{DISK}_s$: residual disk IOPS capacity of server s,

while the known communication cost between any couple of servers (s_1, s_2) is given by a parameter $COST_{s_1 s_2}$. To be precise, $\overline{BDW}_s^{out}$ and $\overline{BDW}_s^{in}$ already account for possible variations in the traffic demands between pairs of already existing containers caused by the new containers request. For example, a client may request a new container in order to reduce the workload of similar containers, thus changing some already existing values in the traffic matrix. These changes are already accounted for in our parameters so that we can focus only on new traffic.

The variant of the short-term VM/container placement problem that we address can be stated as follows.

> **Virtual Machine Placement Problem** (VMPP)**:** Given a set $\mathcal{S}$ of servers and a set of $\mathcal{C}$ of new VMs (containers), determine an assignment of each new VM (container) to exactly one server so as to minimize the traffic load introduced by the new VM (containers) weighted by the cost of the routing paths, while satisfying the mentioned constraints.

Old traffic appears as a constant and therefore can be neglected. The objective function only needs to account for the internal traffic since, for every possible placement, the hops distance between a server and the WAN outside of the data center is the same, resulting in a constant. This variant of the problem considers the objective function defined in [12] as well as the bin-packing structure used in [13], [14] and [18]. However, differently from [13] and [14], the bandwidth constraints are considered as an additional knapsack dimension, without explicitly modeling the network.

3.3 First nonlinear programming formulation

By considering decision variables:

$$x_{cs} = \begin{cases} 1 & \text{if new container } c \text{ is placed on server } s \\ 0 & \text{otherwise} \end{cases} \qquad \forall c \in \mathcal{C}, \quad \forall s \in \mathcal{S}_u$$

and the auxiliary variables:

$t_{s_1 s_2} \geq 0:$ amount of new traffic between servers s_1 and s_2 $\qquad \forall s_1, s_2 \in \mathcal{S},$

the problem can be formulated as follows:

$$min \quad \sum_{s_1, s_2 \in \mathcal{S}} COST_{s_1 s_2} \cdot t_{s_1 s_2} \tag{1.1}$$

s.t.

$$\sum_{s \in \mathcal{S}_u} x_{cs} = 1 \quad \forall c \in \mathcal{C} \tag{1.2}$$

$$\sum_{c \in \mathcal{C}} CPU_{cs} \cdot x_{cs} \leq \overline{CPU}_s \quad \forall s \in \mathcal{S}_u \tag{1.3}$$

$$\sum_{c \in \mathcal{C}} MEM_c \cdot x_{cs} \leq \overline{MEM}_s \quad \forall s \in \mathcal{S}_u \tag{1.4}$$

$$\sum_{c \in \mathcal{C}} DISK_c \cdot x_{cs} \leq \overline{DISK}_s \quad \forall s \in \mathcal{S}_u \tag{1.5}$$

$$\sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}) \cdot x_{c_1 s} + \sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \overline{\mathcal{C}} \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - \overline{x}_{c_2 s}) \cdot x_{c_1 s} + \sum_{c_1 \in \overline{\mathcal{C}}} \sum_{c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}) \cdot \overline{x}_{c_1 s} \leq \overline{BDW}_s^{out} \quad \forall s \in \mathcal{S}_u \tag{1.6}$$

$$\sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C}} d_{c_2 c_1} \cdot (1 - x_{c_2 s}) \cdot x_{c_1 s} + \sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \overline{\mathcal{C}} \cup \{c_0\}} d_{c_2 c_1} \cdot (1 - \overline{x}_{c_2 s}) \cdot x_{c_1 s} +$$
$$\sum_{c_1 \in \overline{\mathcal{C}}} \sum_{c_2 \in \mathcal{C}} d_{c_2 c_1} \cdot (1 - x_{c_2 s}) \cdot \overline{x}_{c_1 s} \leq \overline{BDW}_s^{in} \quad \forall s \in \mathcal{S}_u \qquad (1.7)$$

$$t_{s_1 s_2} = \sum_{c_1, c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot x_{c_1 s_1} \cdot x_{c_2 s_2} \quad +$$
$$\sum_{c_1 \in \mathcal{C}, c_2 \in \overline{\mathcal{C}}} d_{c_1 c_2} \cdot \overline{x}_{c_2 s_2} \cdot x_{c_1 s_1} \quad +$$
$$\sum_{c_1 \in \overline{\mathcal{C}}, c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot \overline{x}_{c_1 s_1} \cdot x_{c_2 s_2} \quad \forall s_1, s_2 \in \mathcal{S} \qquad (1.8)$$

$$x_{cs} \in \{0, 1\} \quad \forall c \in \mathcal{C}, \quad \forall s \in \mathcal{S}_u.$$

An explanation of the objective function and constraints is provided below:

The objective function, i.e., Equation (1.1), aims at minimizing the sum of the new traffic between each pair of servers, that is equivalent to minimize the overall new traffic introduced in the system. The traffic is given by Equations (1.8) and is weighted by the number of hops, which is known a-priori for each servers pair's path.

Constraints (1.2) express the assignment of containers to servers. Each new container *c* must be assigned to exactly one under-utilized server *s*.

Constraints (1.3)-(1.5) express the servers' capacity constraints for resources such as CPU, RAM and disk IOPS. For each server *s*, the sum of the demands of new containers assigned to *s* must not exceed *s*'s residual capacity for that resource. This holds for each of the three resources.

Constraints (1.6)-(1.7) express the servers' capacity constraints for in/out bandwidth resources. Differently from the three resources discussed above, the demands related to the bandwidth does not depend only on the single containers, but also on the placement of inter-communicating containers. This is due to the fact that the communication between two containers placed on the same server does not pass through any physical link. Constraints (1.6) impose that the new outgoing traffic demand exiting from each server s is given by the traffic between two new containers, or from a new container to an old container, or from an old container to a new container. In all these cases, the related term is nonzero only if the source container c_1 is assigned to s, while the destination container c_2 is not. External traffic is accounted for by the inclusion of c_0. Symmetrically, Constraints (1.7) deal with incoming traffic demands. The total demands cannot exceed the residual capacities of the servers.

Equations (1.8) are auxiliary constraints making the new traffic between every pair of servers explicit. The first part accounts for the traffic between new containers, while the second and the third parts account for the traffic new-to-old and old-to-new respectively.

The formulation is clearly nonlinear due to the presence of bilinear terms, in particular the multiplication of binary variables in Constraints (1.6), (1.7) and in the first term of Equations (1.8).

3.4 Linearization and improvements

As previously mentioned, the above formulation contains three types of bilinear terms, such as:

$$t_{s_1 s_2} = \sum_{c_1, c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot \underbrace{x_{c_1 s_1} \cdot x_{c_2 s_2}} \quad + \ldots \quad (1.8),$$

$$\sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot (1 - \underbrace{x_{c_2 s}) \cdot x_{c_1 s}} + \ldots \quad (1.6),$$

$$\sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C}} d_{c_2 c_1} \cdot (1 - \underbrace{x_{c_2 s}) \cdot x_{c_1 s}} + \ldots \quad (1.7).$$

Each product of two variables can be linearized by introducing a new binary variable w that maintains the same meaning and behavior thanks to three new constraints:

$$w_{c_1 s_1 c_2 s_2} \le x_{c_1 s_1} \quad \forall c_1, c_2 \in \mathcal{C}, \quad \forall s_1, s_2 \in \mathcal{S}_u \tag{1.9}$$

$$w_{c_1 s_1 c_2 s_2} \le x_{c_2 s_2} \quad \forall c_1, c_2 \in \mathcal{C}, \quad \forall s_1, s_2 \in \mathcal{S}_u \tag{1.10}$$

$$w_{c_1 s_1 c_2 s_2} \ge x_{c_1 s_1} + x_{c_2 s_2} - 1 \quad \forall c_1, c_2 \in \mathcal{C}, \quad \forall s_1, s_2 \in \mathcal{S}_u \tag{1.11}$$

$$w_{c_1 s_1 c_2 s_2} \in \{0,1\} \quad \forall c_1, c_2 \in \mathcal{C}, \quad \forall s_1, s_2 \in \mathcal{S}.$$

Indeed, the product between $x_{c_1 s_2}$ and $x_{c_2 s_2}$ is equal to 1 if and only if both variables have a value of 1, thus forcing $w_{c_1 s_1 c_2 s_2}$ to 1 thanks to Constraints (1.11). Instead, if either $x_{c_1 s_2}$ or $x_{c_2 s_2}$ have value 0, both their product and, thanks to Constraints (1.9)-(1.10), variable $w_{c_1 s_1 c_2 s_2}$ are equal to 0. This way, Constraints (1.6) become:

$$\sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot (x_{c_1 s} - w_{c_1 s c_2 s}) + \sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \overline{\mathcal{C}} \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - \overline{x}_{c_2 s}) \cdot x_{c_1 s} + \sum_{c_1 \in \overline{\mathcal{C}}} \sum_{c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}) \cdot \overline{x}_{c_1 s} \leq \overline{BDW}_s^{out} \quad \forall s \in \mathcal{S}_u, \tag{1.12}$$

while Constraints (1.7) become:

$$\sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C}} d_{c_2 c_1} \cdot (x_{c_1 s} - w_{c_1 s c_2 s}) + \sum_{c_1 \in \mathcal{C}} \Big(\sum_{c_2 \in \overline{\mathcal{C}} \cup \{c_0\}} d_{c_2 c_1} \cdot (1 - \overline{x}_{c_2 s}) \Big) \cdot x_{c_1 s} + \sum_{c_1 \in \overline{\mathcal{C}}} \sum_{c_2 \in \mathcal{C}} d_{c_2 c_1} \cdot (1 - x_{c_2 s}) \cdot \overline{x}_{c_1 s} \leq \overline{BDW}_s^{in} \quad \forall s \in \mathcal{S}_u \tag{1.13}$$

and Equations (1.8) become:

$$t_{s_1 s_2} = \sum_{c_1, c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot w_{c_1 s_1 c_2 s_2} + \sum_{c_1 \in \mathcal{C}, c_2 \in \overline{\mathcal{C}}} d_{c_1 c_2} \cdot \overline{x}_{c_2 s_2} \cdot x_{c_1 s_1} + \sum_{c_1 \in \overline{\mathcal{C}}, c_2 \in \mathcal{C}} d_{c_1 c_2} \cdot \overline{x}_{c_1 s_1} \cdot x_{c_2 s_2} \quad \forall s_1, s_2 \in \mathcal{S}. \tag{1.14}$$

The formulation is now a Mixed Integer Linear Program. Since we know a-priori that containers will communicate only with containers owned by the same customer, we improve the formulation by partitioning the sets of new and old containers and the traffic matrix. First of all we define some new sets:

- a set $\mathcal{C}_r \subset \mathcal{C}$ composed by all the new containers owned by customer $r \in \mathcal{R}$. There is one of these sets for every customer r and they constitute a partition of $\mathcal{C}$,

- a set $\overline{\mathcal{C}}_r \subset \overline{\mathcal{C}}$ composed by all the old containers owned by customer $r \in \mathcal{R}$. There is one of these sets for every customer r but they do not constitute a partition of $\overline{\mathcal{C}}$ because $\mathcal{R}$ collects only customers of the new request ,
- a set $\mathcal{S}_r \subset \mathcal{S}$ composed by all the servers that host at least one container owned by customer $r \in \mathcal{R}$. There is one of these sets for every customer r, but they do not constitute a partition of $\mathcal{S}$.

Then, we can partition the traffic matrix D into $|\mathcal{R}|$ traffic matrices D^r, and the linearization variables w into multiple w^r variables. Special container c_0 is added to the traffic matrix of every customer r. Constraints (1.6), already changed into (1.12), finally become:

$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_1c_2}\cdot(x_{c_1s}-w^r_{c_1sc_2s})+\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\overline{\mathcal{C}}_r\cup\{c_0\}} d^r_{c_1c_2}\cdot(1-\overline{x}_{c_2s})\cdot x_{c_1s}+ \sum_{r\in\mathcal{R}}\sum_{c_1\in\overline{\mathcal{C}}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_1c_2}\cdot(1-x_{c_2s})\cdot\overline{x}_{c_1s}\leq\overline{BDW}^{out}_s \quad \forall s\in\mathcal{S}_u. \tag{1.15}$$

Similarly Constraints (1.7), already changed into (1.13), finally become:

$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_2c_1}\cdot(x_{c_1s}-w^r_{c_1sc_2s})+\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\overline{\mathcal{C}}_r\cup\{c_0\}} d^r_{c_2c_1}\cdot(1-\overline{x}_{c_2s})\cdot x_{c_1s}+ \sum_{r\in\mathcal{R}}\sum_{c_1\in\overline{\mathcal{C}}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_2c_1}\cdot(1-x_{c_2s})\cdot\overline{x}_{c_1s}\leq\overline{BDW}^{in}_s \quad \forall s\in\mathcal{S}_u \tag{1.16}$$

The linearization Constraints (1.9)-(1.11) also need to be modified as follows:

$$w^r_{c_1s_1c_2s_2}\leq x_{c_1s_1} \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r,\quad \forall s_1,s_2\in\mathcal{S}_u \tag{1.17}$$

$$w^r_{c_1s_1c_2s_2}\leq x_{c_2s_2} \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r,\quad \forall s_1,s_2\in\mathcal{S}_u \tag{1.18}$$

$$w^r_{c_1s_1c_2s_2}\geq x_{c_1s_1}+x_{c_2s_2}-1 \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r,\quad \forall s_1,s_2\in\mathcal{S}_u \tag{1.19}$$

$$w^r_{c_1s_1c_2s_2}\in\{0,1\} \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r,\quad \forall s_1,s_2\in\mathcal{S}_u.$$

Finally, auxiliary Equations (1.8), already changed into Equations (1.14), can be re-written as:

$$
\begin{aligned}
t_{s_1 s_2} = \sum_{r\in\mathcal{R}} \sum_{c_1,c_2\in\mathcal{C}_r} & d^r_{c_1c_2} \cdot w^r_{c_1s_1c_2s_2} \quad + \\
\sum_{r\in\mathcal{R}} \sum_{c_1\in\mathcal{C}_r,c_2\in\overline{\mathcal{C}}_r} & d^r_{c_1c_2} \cdot \overline{x}_{c_2s_2} \cdot x_{c_1s_1} \quad + \\
\sum_{r\in\mathcal{R}} \sum_{c_1\in\overline{\mathcal{C}}_r,c_2\in\mathcal{C}_r} & d^r_{c_1c_2} \cdot \overline{x}_{c_1s_1} \cdot x_{c_2s_2} \\
& \forall s_1, s_2 \in \mathcal{S}_u. \qquad (1.20)
\end{aligned}
$$

Notice that by restricting t's and w^r's definition to just $\{... \times \mathcal{S}_u \times \mathcal{S}_u\}$, the objective function needs to be slightly updated in order to take into account traffic respectively going towards and coming from servers that are outside of the decision area, which means any server $s \in (\bigcup_{r\in\mathcal{R}} \mathcal{S}_r) \setminus \mathcal{S}_u$. Thus, Equation (1.1) becomes:

$$
\begin{aligned}
min \quad & \sum_{s_1,s_2\in\mathcal{S}_u} COST_{s_1s_2} \cdot t_{s_1s_2} \quad + \\
& \sum_{r\in\mathcal{R}} \sum_{s_1\in\mathcal{S}_u,s_2\in\mathcal{S}_r\setminus\mathcal{S}_u} COST_{s_1s_2} \cdot \sum_{c_1\in\mathcal{C}_r,c_2\in\overline{\mathcal{C}}_r} d^r_{c_1c_2} \cdot \overline{x}_{c_2s_2} \cdot x_{c_1s_1} \quad + \\
& \sum_{r\in\mathcal{R}} \sum_{s_1\in\mathcal{S}_r\setminus\mathcal{S}_u,s_2\in\mathcal{S}_u} COST_{s_1s_2} \cdot \sum_{c_1\in\overline{\mathcal{C}}_r,c_2\in\mathcal{C}_r} d^r_{c_1c_2} \cdot \overline{x}_{c_1s_1} \cdot x_{c_2s_2} \qquad (1.21)
\end{aligned}
$$

where the second term counts traffic going from $\mathcal{S}_u$ to $\mathcal{S}_r$, while the third counts

the traffic in the opposite direction. Notice that, within these two terms, a new container $c \in \mathcal{C}_r$ can only be placed on a server $s \in \mathcal{S}_u$, while a container $c \in \overline{\mathcal{C}}_r$ can only be placed on a server $s \in \mathcal{S}_r \setminus \mathcal{S}_u$ because other combinations are already counted by the first complex term. The final formulation is reported in the Appendix.

CHAPTER 4

VM Migration Problem

This chapter is devoted to VM consolidation, i.e., re-allocating some resources whose utilization is not optimal anymore with strong emphasis on reducing power consumption and energy costs. The consolidation is done through the tool of VM live-migration.

In Section 4.1 we provide a concise description of the context of the problem, its relevant aspects and the reasons behind some particular decisions. In Section 4.2 we provide a precise definition of the problem, listing all parameters and constraints as well as the objective. Since the problem is an extension of the VMPP, we build on the definition discussed in Chapter 3. In Section 4.3 we provide a first nonlinear programming formulation of the problem. As in the VMPP's case, it is useful to understand the main constraints and the objective function but cannot be directly used in practice. In Section 4.5 we derive a MILP formulation as well as discussing other improvements that make the formulation more compact. In Section 4.6 we describe how the subset of VMs to be migrated is selected from set of all VMs. Indeed, this is a choice that needs to be separated to the actual migration itself, otherwise the overall problem turns out to be too challenging. Therefore, we describe some possible criteria for selecting those VMs. Finally, Section 4.7 is devoted to a sub-problem that arises when, in the VMs allocation resulting from the migration, the routing paths need to be updated and their costs recomputed.

4.1 High level description

Solving the short-term VM Placement Problem to quickly allocate in a reasonable time new VMs requests can lead in the long period to resources' saturation due to the fact that, up until this point, we never reconsidered a decision/placement already done. This fact, together with VMs continuously entering and exiting the system, causes situations where a placement that was optimal some minutes before may not be optimal anymore and needs to be reconsidered. This complex decision process cannot be taken into account every few minutes because it would lead to a large overhead. The problem to be faced is to plan VM migration periodically on a longer time interval, e.g., every hour. The main idea is to update the placement of some VMs without stopping the whole system. To achieve this, live-migration of VMs is employed. This way, while most of the data center continues its normal operation, the VMs chosen for migration suspend their normal incoming and outgoing traffic and their state is sent through the network to their new placement with a traffic burst that temporarily generates an additional overhead on the physical links involved. The goal of the VM migration problem is to determine a new configuration that is more efficient in terms of energy consumption and traffic management.

To further simplify the problem, which is already very challenging, we assume the selection of VMs to migrate to be part of a pre-processing phase carried out by the monitoring system. This pre-processing is performed with two greedy algorithms executed on each server. Their goal will be to select a set of VMs whose migration is mandatory and a set of VMs whose migration is only suggested. For what concerns the migration time and the consequent downtime of applications, we do not consider it as a variable to be optimized but rather as part of the parameters. As for the VMPP, in the problem description we refer to containers instead of VMs.

4.2 Problem description

Given a data center network composed by:

- a set of nodes $\mathcal{N}$ that collects both switches and servers,
- a set of servers $\mathcal{S} \subset \mathcal{N}$,
- a set of arcs $\mathcal{E} \subset \mathcal{N} \times \mathcal{N}$ that represents the physical links,
- a set of containers $\mathcal{C}$ distributed across the servers,

we want to update the placement of some containers in order to better utilize the resources. To achieve this goal, live-migration of containers is employed. In particular, we have:

- a set $\mathcal{C}_{ob} \subset \mathcal{C}$ of containers whose migration is mandatory,

- a set $\mathcal{C}_f \subset \mathcal{C}$ of containers whose migration is suggested, but not mandatory.

We have to be sure that the system's resources are capable of satisfying the containers' demand in the new configuration as well as during the migration. In particular:

- each server s is characterized by:

  CPU_s : CPU capacity (i.e. number of cores),

  MEM_s : RAM capacity,

  BDW_s^{out} : OUT bandwidth capacity,

  BDW_s^{in} : IN bandwidth capacity,

  $DISK_s$: disk IOPS capacity.

  $P_s^{max} - P_s^{idle}$: power-consumption coefficient,

- each physical link (i,j) is characterized by:

  K_{ij} : bandwidth capacity,

- each container c is characterized by:

  CPU_{cs} : CPU utilization contribution normalized on server s,

  MEM_c : RAM demand,

  $DISK_c$: disk IOPS demand,

  d_{cb} : average traffic from c to any other container b,

  d_{bc} : average traffic from any other container b to c,

  Q_c : size of the container's state.

We also have the residual amount of each resource related to servers and links: $\overline{CPU}_s$, $\overline{MEM}_s$, $\overline{BDW}_s^{out}$, $\overline{BDW}_s^{in}$, $\overline{DISK}_s$, $\overline{K}_{ij}$. Finally, we know:

- the placement of each container before the migration phase, which is represented by a binary parameter

$$x_{cs}^{old} = \begin{cases} 1 & \text{if container } c \text{ is assigned to server } \quad s \text{ in the old configuration} \\ 0 & \text{otherwise} \end{cases} \qquad \forall c \in \mathcal{C}, \quad \forall s \in \mathcal{S},$$

- the average time T_1 needed for the migration and the maximum time T_2 ($> T_1$) that we allow the migration to take,

- the on/off state of each server s before the migration phase, represented by a binary parameter b_s^{old},
- the set of arcs $\mathcal{P}_{s_1 s_2}$ that constitute the pre-computed path between each couple of servers.
- the old traffic amounts $t_{s_1 s_2}^{old}$ between every couple of servers (s_1, s_2).

The objective function of the problem is a combination of three terms that aim at:

1. minimizing the global servers' energy consumption under the new configuration post-migration,
2. minimizing the global traffic's load on the network under the new configuration post-migration,
3. maximizing the number of migrated containers.

The traffic is weighted by the number of hops traversed, following the approach of Meng et al. [12]. External traffic is modeled with a dummy container c_0 and a dummy server s_0, exactly as described in Chapter 3 for the VMPP.

The variant of the long-term consolidation problem that we consider is the defined as follows.

> **Virtual Machine Migration Problem** (VMMP)**:** Given a set of servers $\mathcal{S}$ and two sets of possibly migrating VMs (containers) $\mathcal{C}_{ob}$ and $\mathcal{C}_f$ with an associated initial placement, determine a new assignment of every migrating VM (container) to a server so as to minimize the objective function previously described, while satisfying the mentioned constraints.

The objective function is a combination of the ones defined in [12], from which the traffic term is inspired, and [18], from which the energy and migration terms are inspired, while the network constraints share some similarities with the ones in [16], in particular regarding the traffic suspension of migrating VMs/containers.

4.3 First nonlinear programming formulation

For the sake of readability, let us define:

$$\mathcal{C}_m \equiv \mathcal{C}_{ob} \cup \mathcal{C}_f$$

$$P_s \equiv P_s^{max} - P_s^{idle}.$$

We consider decision variables:

$$b_s^{new} = \begin{cases} 1 & \text{if server } s \text{ is ON in the new configuration} \\ 0 & \text{otherwise} \end{cases} \qquad \forall s \in \mathcal{S},$$

$$z_c = \begin{cases} 1 & \text{if container } c \text{ is migrated} \\ 0 & \text{otherwise} \end{cases} \qquad \forall c \in \mathcal{C}_m,$$

$$x_{cs}^{new} = \begin{cases} 1 & \text{if container } c \text{ is assigned to server} \\ & s \text{ in the new configuration} \\ 0 & \text{otherwise} \end{cases} \qquad \forall c \in \mathcal{C}_m, \quad \forall s \in \mathcal{S},$$

$f_{ij}^c \geq 0$: amount of bandwidth consumed by the migration of container c on arc (i,j) $\quad \forall (i,j) \in \mathcal{E}, \quad \forall c \in \mathcal{C}_m,$

together with the auxiliary variables:

$u_s \geq 0$: cpu utilization of server s in the new configuration $\qquad \forall s \in \mathcal{S},$

$t_{s_1 s_2} \geq 0$: amount of inter-container traffic between servers s_1 and s_2 in the new configuration $\qquad \forall s_1, s_2 \in \mathcal{S}$

Notice that the δ notation is used to denote the cut of a set of nodes $\overline{\mathcal{N}}$ (in our case always singleton nodes), i.e., the set of arcs that connect $\overline{\mathcal{N}}$ to its complement set $\mathcal{N} \setminus \overline{\mathcal{N}}$. In particular, δ^+ indicates outgoing arcs while δ^- indicates incoming arcs. Then, the problem can be formulated as follows:

$$\begin{aligned} min \quad & \sum_{s \in S} P_s \cdot u_s + P_s^{idle} \cdot b_s^{new} \\ & +\alpha \cdot \sum_{s_1 \in S, s_2 \in S} COST_{s_1 s_2} \cdot t_{s_1 s_2}^{new} \\ & -\beta \cdot \sum_{c \in C_f} z_c \end{aligned} \tag{2.1}$$

s.t.

$$\sum_{s \in S} x_{cs}^{new} = 1 \quad \forall c \in \mathcal{C}_m \tag{2.2}$$

$$x_{cs}^{new} \leq b_s^{new} \quad \forall c \in \mathcal{C}_m, \quad \forall s \in \mathcal{S} \tag{2.3}$$

$$x_{cs}^{old} \leq b_s^{new} \quad \forall c \notin \mathcal{C}_m, \quad \forall s \in \mathcal{S} \tag{2.4}$$

$$\sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{new} \leq \rho_3 \cdot CPU_s - \sum_{c \notin \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} \quad \forall s \in S \tag{2.5}$$

$$\sum_{c \in \mathcal{C}_m} MEM_c \cdot x_{cs}^{new} \leq MEM_s - \sum_{c \notin \mathcal{C}_m} MEM_c \cdot x_{cs}^{old} \quad \forall s \in S \tag{2.6}$$

$$\sum_{c \in \mathcal{C}_m} DISK_c \cdot x_{cs}^{new} \leq DISK_s - \sum_{c \notin \mathcal{C}_m} DISK_{cs} \cdot x_{cs}^{old} \quad \forall s \in S \tag{2.7}$$

$$\begin{aligned} &\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{new}) \cdot x_{c_1 s}^{new} + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{new} + \\ &\sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{new}) \cdot x_{c_1 s}^{old} + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} \\ &\leq BDW_s^{out} \quad \forall s \in \mathcal{S} \end{aligned} \tag{2.8}$$

$$\begin{aligned} &\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_2 c_1} \cdot (1 - x_{c_2 s}^{new}) \cdot x_{c_1 s}^{new} + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_2 c_1} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{new} + \\ &\sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_2 c_1} \cdot (1 - x_{c_2 s}^{new}) \cdot x_{c_1 s}^{old} + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_2 c_1} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} \\ &\leq BDW_s^{in} \quad \forall s \in \mathcal{S} \end{aligned} \tag{2.9}$$

$$z_c = 1 \quad \forall c \in \mathcal{C}_{ob} \tag{2.10}$$

$$\sum_{j \in \delta^+(s)} f_{sj}^c = \frac{Q_c}{T_1} \cdot x_{cs}^{old} \cdot (1 - x_{cs}^{new}) \quad \forall c \in \mathcal{C}_m, \quad \forall s \in \mathcal{S} \tag{2.11}$$

$$\sum_{i \in \delta^-(s)} f_{is}^c = \frac{Q_c}{T_1} \cdot (1 - x_{cs}^{old}) \cdot x_{cs}^{new} \quad \forall c \in \mathcal{C}_m, \quad \forall s \in \mathcal{S} \tag{2.12}$$

$$\sum_{i \in \delta^-(n)} f_{in}^c - \sum_{j \in \delta^+(n)} f_{nj}^c = 0 \quad \forall n \in \mathcal{N} \setminus \mathcal{S}, \quad \forall c \in \mathcal{C}_m \tag{2.13}$$

$$\sum_{c\in\mathcal{C}_m} f_{ij}^c + \sum_{s_1,s_2\in\mathcal{S}\cup\{s_0\}:(i,j)\in\mathcal{P}_{s_1s_2}} t_{s_1s_2}^{old} -$$
$$\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}} \sum_{c_1,c_2\in\mathcal{C}_m} x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}\cdot d_{c_1c_2}\cdot(1-(1-z_{c_1})\cdot(1-z_{c_2})) \quad -$$
$$\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}} \sum_{c_1\notin\mathcal{C}_m,c_2\in\mathcal{C}_m} x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}\cdot d_{c_1c_2}\cdot z_{c_2} \quad -$$
$$\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}} \sum_{c_1\in\mathcal{C}_m,c_2\notin\mathcal{C}_m)} x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}\cdot d_{c_1c_2}\cdot z_{c_1} -$$
$$\sum_{s\in\mathcal{S}:(i,j)\in\mathcal{P}_{ss_0}} \sum_{c\in\mathcal{C}_m} x_{cs}^{old}\cdot d_{cc_0}\cdot z_c \quad - \sum_{s\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_0s}} \sum_{c\in\mathcal{C}_m} x_{cs}^{old}\cdot d_{c_0c}\cdot z_c$$
$$\leq K_{ij} \quad \forall (i,j)\in\mathcal{E} \quad (2.14)$$

$$z_c = 1-\sum_{s\in S} x_{cs}^{old}\cdot x_{cs}^{new} \quad \forall c\in\mathcal{C}_m \quad (2.15)$$

$$u_s = \frac{\sum_{c\in\mathcal{C}_m} CPU_{cs}\cdot x_{cs}^{new} + \sum_{c\notin\mathcal{C}_m} CPU_{cs}\cdot x_{cs}^{old}}{CPU_s} \quad \forall s\in S \quad (2.16)$$

$$t_{s_1s_2}^{new} = \sum_{c_1,c_2\in\mathcal{C}_m} d_{c_1,c_2}\cdot x_{c_1s_1}^{new}\cdot x_{c_2s_2}^{new} +$$
$$\sum_{c_1\in\mathcal{C}_m,c_2\notin\mathcal{C}_m} d_{c_1c_2}\cdot x_{c_2s_2}^{old}\cdot x_{c_1s_1}^{new} \quad +$$
$$\sum_{c_1\notin\mathcal{C}_m,c_2\in\mathcal{C}_m} d_{c_1c_2}\cdot x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{new} \quad +$$
$$\sum_{c_1,c_2\notin\mathcal{C}_m} d_{c_1c_2}\cdot x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old} \quad \forall s_1,s_2\in\mathcal{S} \quad (2.17)$$

$$x_{cs}^{new}, b_s^{new}, z_c\in\{0,1\} \qquad \forall c\in\mathcal{C}_m, \quad \forall s\in\mathcal{S}$$

$$f_{ij}^c\in\mathbb{R}^+ \qquad \forall (i,j)\in\mathcal{E}, \quad \forall c\in\mathcal{C}_m.$$

A description of the objective function and constraints is provided below.

Equation (2.1) is our composite objective function. The first term accounts for the energy consumption of all servers post-migration and is composed by the sum of a contribution that is linear in the servers' utilization and a fixed contribution due to the on/off state of the servers. The second term corresponds to the global traffic inside the network post-migration weighted by a cost factor that indicates the number of hops traversed. The third term gives an incentive for the containers' migration when it is not mandatory, because we would like to move most of them. Since we want an objective expressed in terms of energy cost, the second and third terms are weighted with two coefficients.

Constraints (2.2) ensure that every container c whose migration is considered must be assigned to exactly one server s under the new configuration.

Constraints (2.3)-(2.4) enforce the on/off state of each server s under the new configuration. A server is ON if any container considered for migration is assigned to it *or* if any container, whose migration was not considered, was (and still is) placed on it. The OFF state is forced by the objective function in case of an empty server.

Constraints (2.5)-(2.7) are knapsack constraints regulating server resources such as CPU, RAM and disk. Each container c whose migration is considered consumes a part of the resources of the server where it is placed. For each server s, the total amount of these demands must not exceed the difference between the total capacity and the amount of resource already consumed by other containers. The CPU constraints use a threshold ρ_3 (e.g. 0.8) because we do not want to end up with a configuration that has the same issue we were trying to solve.

Constraints (2.8)-(2.9) make sure that we do not exceed the out/in bandwidth capacities. These resources are less trivial to handle because the bandwidth demand of a container heavily depends on the placement of other containers. The out-bandwidth consumed on a server s is the sum of all traffic going from containers c_1, placed on s, towards containers c_2, not placed on s. These (c_1, c_2) pairs are decomposed in all four combinations of migrating/non-migrating containers. Same holds for in-bandwidth. External traffic is counted by dummy container c_0.

Constraints (2.10) make sure that, for every container c whose migration is mandatory, the related migration variable must be set to 1.

Constraints (2.11)-(2.12) regulate the traffic generated by the migration of containers in the source points and end points. For each server s and each migrating container c, the exiting (entering) migration traffic from (in) s must be equal to the size of c's state converted into bandwidth if and only if c's old (new) placement was (is) on s but new (old) one is not. In all other cases the traffic is $= 0$. We use T_1 instead of T_2 when converting the containers' size because we are interested in average times and average flows. Using T_2 would provide lower-bounds on flows which may not be safe when considering capacities that must not be exceeded.

Constraints (2.13) regulate the migration traffic's flow inside the network. For each intermediate switch n, the amount of exiting flow must be equal to the amount of entering flow. This must hold for the flow related to any possibly migrating container c.

Constraints (2.14) constitute the core of the problem. These constraints make sure that all physical links (i,j) are able to accommodate both the inter-container traffic and the migration traffic at the same time. In particular we have that the sum of seven different terms must not exceed the link's capacity. The first term takes into account all the migration flow traversing (i,j). The second term is a constant due to the inter-container traffic between servers before the migration. Only servers' pairs whose routing path traverses the link (i,j) are considered. Paths toward/from the dummy server (Internet) are included. This second contribute is inaccurate because some of this traffic will be suspended during the migration due to containers' downtime. This is taken into account by the last five terms, three of which subtract the traffic between pairs of both (possibly) migrating containers, one fixed and one migrating container, one migrating and one fixed container respectively. In particular, one of these contains a logical $z_{c_1} \vee z_{c_2}$ which, thanks to De Morgan rule, is reshaped into a logical $!(!z_{c_1} \wedge !z_{c_2})$, more suitable to be expressed in a mathematical formulation. The last two terms subtract the quantities related to traffic going to/coming from the Internet that is suspended.

Constraints (2.15) express the meaning of variables z by linking them to variables x. A container c is migrated if and only if its new placement differs from the old one, meaning that all products between x^{old} and x^{new} should be equal to 0.

Equations (2.16) are auxiliary constraints expliciting the utilization factor of a server s post-migration, given by the sum of all the CPU utilization contributions of both migrated and non-migrated containers placed on it. Since we want it to be in the interval [0,1], it is normalized on the number of cores of the server.

Equations (2.17) are auxiliary constraints highlighting the traffic between each pair of servers post-migration, composed by four terms that account for the traffic between couples of respectively both (possibly) migrated, one migrated and one fixed, one fixed and one migrated, both fixed containers.

4.4 Remarks

The formulation presented above is non-linear due to the presence of a bilinear term in Constraints (2.8), (2.9) and in Equations (2.17) and also due to Constraints (2.14), where a logical *or* between two variables appears. The linearization of these terms together with a further manipulation of most of the constraints will be discussed later on.

Another issue worth discussing is the fact that the pre-computed paths between any couple of servers used both here and in the short-term VMPP are never updated. This might lead in the long period to sub-optimal paths that together with their related costs guide the solutions towards mediocre results. It seems natural that the long-term VMMP, which is already performing a sort of system reconfiguration, should take care of the reconfiguration of these paths too. However, the formulation presented above is already quite heavy and adding other constraints, variables and sources of non-linearity (e.g. $COST_{s_1s_2}$ would become a variable) might increase the complexity too much. We would rather treat it separately as a sub-problem used to iteratively approximate the correct costs for the main problem. In this way we trade the exact optimum of the overall problem to gain something on the computational side (e.g. size of solvable instances), hoping that the solution remains not too far from the optimal one.

4.5 Linearization and improvements

As previously mentioned, the formulation of VMMP presented in Section 4.3 can be improved in order to get:

- a MILP formulation through exact linearization of the bilinear terms,
- a more compact formulation at the price of losing some readability.

Exactly as done in Chapter 3.4 for the VMPP, if we take the first term of Equations (2.17):

$$t^{new}_{s_1s_2} = \sum_{c_1,c_2 \in \mathcal{C}_m} d_{c_1,c_2} \cdot \underbrace{x^{new}_{c_1s_1} \cdot x^{new}_{c_2s_2}} + \ldots \quad \forall s_1, s_2 \in \mathcal{S}$$

and the following terms from Constraints (2.8) and (2.9):

$$\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - \underbrace{x^{new}_{c_2s}) \cdot x^{new}_{c_1s}} + \ldots \quad \forall s \in \mathcal{S}$$

$$\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_2c_1} \cdot (1 - \underbrace{x^{new}_{c_2s}) \cdot x^{new}_{c_1s}} + \ldots \quad \forall s \in \mathcal{S},$$

we can replace each product of two binary variables with a new binary variable $w_{c_1s_1c_2s_2}$ adding three new sets of constraints:

$$w_{c_1s_1c_2s_2} \leq x^{new}_{c_1s_1} \quad \forall c_1, c_2 \in \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.18}$$

$$w_{c_1s_1c_2s_2} \leq x^{new}_{c_2s_2} \quad \forall c_1, c_2 \in \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.19}$$

$$w_{c_1s_1c_2s_2} \geq x^{new}_{c_1s_1} + x^{new}_{c_2s_2} - 1 \quad \forall c_1, c_2 \in \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.20}$$

$$w_{c_1s_1c_2s_2} \in \{0, 1\} \quad \forall c_1, c_2 \in \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S}.$$

These together make sure that the w variable acts precisely as the logical *and* (product) of two binary variables. Equations (2.17) become:

$$t^{new}_{s_1s_2} = \sum_{c_1, c_2 \in \mathcal{C}_m} d_{c_1,c_2} \cdot w_{c_1s_1c_2s_2} + \ldots \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.21}$$

while Constraints (2.8) and (2.9) respectively become:

$$\begin{aligned}\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (x^{new}_{c_1s} - w_{c_1sc_2s}) + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot x^{new}_{c_1s} + \\ \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - x^{new}_{c_2s}) \cdot x^{old}_{c_1s} + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot x^{old}_{c_1s} \\ \leq BDW^{out}_s \quad \forall s \in \mathcal{S},\end{aligned} \tag{2.22}$$

$$\begin{aligned}\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_2c_1} \cdot (x^{new}_{c_1s} - w_{c_1sc_2s}) + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_2c_1} \cdot (1 - x^{old}_{c_2s}) \cdot x^{new}_{c_1s} + \\ \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_2c_1} \cdot (1 - x^{new}_{c_2s}) \cdot x^{old}_{c_1s} + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_2c_1} \cdot (1 - x^{old}_{c_2s}) \cdot x^{old}_{c_1s} \\ \leq BDW^{in}_s \quad \forall s \in \mathcal{S}.\end{aligned} \tag{2.23}$$

In a similar way, taking the third term of Constraints (2.14):

$$\dots \quad \sum_{s_1,s_2 \in \mathcal{S}:(i,j)\in \mathcal{P}_{s_1 s_2}} \Big(\sum_{c_1,c_2 \in \mathcal{C}_m} x^{old}_{c_1 s_1} \cdot x^{old}_{c_2 s_2} \cdot d_{c_1 c_2} \cdot \underbrace{(1-(1-z_{c_1})\cdot(1-z_{c_2}))}\Big)\Big) \quad \dots \qquad \forall (i,j) \in \mathcal{E},$$

we can express the same logical "z_{c_1} *or* z_{c_2}" with a new binary variable $\pi_{c_1 c_2}$ that behaves in the exact same way. As before, three new sets of constraints need to be added:

$$\pi_{c_1 c_2} \geq z_{c_1} \quad \forall c_1, c_2 \in \mathcal{C}_m \tag{2.24}$$

$$\pi_{c_1 c_2} \geq z_{c_2} \quad \forall c_1, c_2 \in \mathcal{C}_m \tag{2.25}$$

$$\pi_{c_1 c_2} \leq z_{c_1} + z_{c_2} \quad \forall c_1, c_2 \in \mathcal{C}_m \tag{2.26}$$

$$\pi_{c_1 c_2} \in \{0,1\} \quad \forall c_1, c_2 \in \mathcal{C}_m$$

and Constraints (2.14) become:

$$\begin{aligned}
&\sum_{c \in \mathcal{C}_m} f^c_{ij} + \sum_{s_1,s_2 \in \mathcal{S}:(i,j)\in \mathcal{P}_{s_1 s_2}} t^{old}_{s_1 s_2} - \\
&\qquad \sum_{s_1,s_2 \in \mathcal{S}:(i,j)\in \mathcal{P}_{s_1 s_2}} \sum_{c_1,c_2 \in \mathcal{C}_m} x^{old}_{c_1 s_1} \cdot x^{old}_{c_2 s_2} \cdot d_{c_1 c_2} \cdot \pi_{c_1 c_2} - \\
&\qquad \sum_{s_1,s_2 \in \mathcal{S}:(i,j)\in \mathcal{P}_{s_1 s_2}} \sum_{c_1 \notin \mathcal{C}_m, c_2 \in \mathcal{C}_m} x^{old}_{c_1 s_1} \cdot x^{old}_{c_2 s_2} \cdot d_{c_1 c_2} \cdot z_{c_2} - \\
&\qquad \sum_{s_1,s_2 \in \mathcal{S}:(i,j)\in \mathcal{P}_{s_1 s_2}} \sum_{c_1 \in \mathcal{C}_m, c_2 \notin \mathcal{C}_m} x^{old}_{c_1 s_1} \cdot x^{old}_{c_2 s_2} \cdot d_{c_1 c_2} \cdot z_{c_1} - \\
&\qquad \sum_{s \in \mathcal{S}:(i,j)\in \mathcal{P}_{s s_0}} \sum_{c \in \mathcal{C}_m} x^{old}_{cs} \cdot d_{c c_0} \cdot z_c - \sum_{s \in \mathcal{S}:(i,j)\in \mathcal{P}_{s_0 s}} \sum_{c \in \mathcal{C}_m} x^{old}_{cs} \cdot d_{c_0 c} \cdot z_c \\
&\qquad\qquad \leq K_{ij} \quad \forall (i,j) \in \mathcal{E}.
\end{aligned} \tag{2.27}$$

Notice a couple of further observations:

1. the containers' migration involves only a small subset of containers compared to the whole set $\mathcal{C}$. This means that the vast majority of quantities (e.g., traffic, resource demands etc.) used in the formulation remain fixed. Therefore we can get rid of these constant contributes and think

only in terms of variations (e.g. energy variations, traffic variations, demands variations etc.) and residual capacities. We can assume that, before starting the migration, we also have access to information about the residual amount of each resource related to servers and links: $\overline{CPU}_s$, $\overline{MEM}_s$, $\overline{BDW}_s^{out}$, $\overline{BDW}_s^{in}$, $\overline{DISK}_s$, $\overline{K}_{ij}$;

2. Similarly to the VMPP, the set of containers and the traffic matrix can be partitioned over the set of customers $\mathcal{R}$.

Constraints (2.5)

$$\sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{new} \leq \rho_3 \cdot CPU_s - \sum_{c \notin \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} \quad \forall s \in \mathcal{S},$$

which is equivalent to:

$$\sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{new} \leq CPU_s - (1-\rho_3)CPU_s - \sum_{c \notin \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} \quad \forall s \in \mathcal{S}, \tag{2.5.alt}$$

can be reformulated by absorbing all contributions of fixed containers into the residual capacity and leaving only positive and negative contributions of the ones considered for migration. A (possibly) migrating container brings a positive contribution if placed on a server and a negative contribution when leaving the server. The two contributions cancel out each other when a container remains in its place. In particular, the residual CPU on a server can be written as:

$$\overline{CPU}_s = CPU_s - \sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} - \sum_{c \notin \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} \quad \forall s \in \mathcal{S},$$

which can be rearranged into:

$$CPU_s = \overline{CPU}_s + \sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} + \sum_{c \notin \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} \quad \forall s \in \mathcal{S}$$

that, substituted into (2.5.alt) gives:

$$\sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{new} \leq \overline{CPU}_s + \sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot x_{cs}^{old} - (1-\rho_3)CPU_s \quad \forall s \in \mathcal{S}$$

which, bringing a term on the left-hand side, finally becomes:

$$\sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot (x_{cs}^{new} - x_{cs}^{old}) \leq \overline{CPU}_s - (1-\rho_3) \cdot CPU_s \quad \forall s \in \mathcal{S}. \tag{2.28}$$

Constraints 2.28 enforce that the variation in CPU utilization, caused by containers migrating *in* and *out* of any server *s*, does not exceed *s*'s residual CPU capacity (discounted by an additional factor in the case of these particular constraints). For readability, we do not report the steps to modify RAM and disk constraints because they follow the same reasoning and are, in fact, more trivial. Constraints (2.6) and (2.7) become:

$$\sum_{c \in \mathcal{C}_m} MEM_c \cdot (x_{cs}^{new} - x_{cs}^{old}) \leq \overline{MEM}_s \quad \forall s \in \mathcal{S} \tag{2.29}$$

$$\sum_{c \in \mathcal{C}_m} DISK_c \cdot (x_{cs}^{new} - x_{cs}^{old}) \leq \overline{DISK}_s \quad \forall s \in \mathcal{S}. \tag{2.30}$$

Constraints (2.8) and (2.9) and their linearized versions (2.22) and (2.23) follow the same approach too. However, since they are much more complex, it is useful to provide the starting point and some intermediate steps (full steps can be found in the Appendix). The residual servers' bandwidth can be expressed as:

$$\begin{aligned}\overline{BDW}_s^{out} = BDW_s^{out} - \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - x_{c_2s}^{old}) \cdot x_{c_1s}^{old} - \\ \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x_{c_2s}^{old}) \cdot x_{c_1s}^{old} - \\ \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - x_{c_2s}^{old}) \cdot x_{c_1s}^{old} - \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x_{c_2s}^{old}) \cdot x_{c_1s}^{old}\end{aligned}$$

Then, using the usual manipulation and substitution we can transform (2.22) into:

$$\begin{aligned}\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (x_{c_1s}^{new} - x_{c_1s}^{old} + (x_{c_1s}^{old} \cdot x_{c_2s}^{old}) - w_{c_1sc_2s}) \quad + \\ \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x_{c_2s}^{old}) \cdot (x_{c_1s}^{new} - x_{c_1s}^{old}) \quad + \\ \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot x_{c_1s}^{old} \cdot (x_{c_2s}^{old} - x_{c_2s}^{new}) \\ \leq \overline{BDW}_s^{out}\end{aligned}$$

and finally, applying partitioning by customers:

$$\begin{aligned}\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_m\cap\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_m\cap\mathcal{C}_r} d^r_{c_1c_2}\cdot(x^{new}_{c_1s}-x^{old}_{c_1s}+(x^{old}_{c_1s}\cdot x^{old}_{c_2s})-w^r_{c_1sc_2s}) \quad +\\ \sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_m\cap\mathcal{C}_r}\sum_{c_2\in(\mathcal{C}_r\setminus\mathcal{C}_m)\cup\{c_0\}} d^r_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot(x^{new}_{c_1s}-x^{old}_{c_1s}) \quad +\\ \sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r\setminus\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m\cap\mathcal{C}_r} d^r_{c_1c_2}\cdot x^{old}_{c_1s}\cdot(x^{old}_{c_2s}-x^{new}_{c_2s})\\ \le\overline{BDW}^{out}_s \quad \forall s\in\mathcal{S}\end{aligned} \tag{2.31}$$

Identically, (2.23) become:

$$\begin{aligned}\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_m\cap\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_m\cap\mathcal{C}_r} d^r_{c_2c_1}\cdot(x^{new}_{c_1s}-x^{old}_{c_1s}+(x^{old}_{c_1s}\cdot x^{old}_{c_2s})-w^r_{c_1sc_2s}) \quad +\\ \sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_m\cap\mathcal{C}_r}\sum_{c_2\in(\mathcal{C}_r\setminus\mathcal{C}_m)\cup\{c_0\}} d^r_{c_2c_1}\cdot(1-x^{old}_{c_2s})\cdot(x^{new}_{c_1s}-x^{old}_{c_1s}) \quad +\\ \sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r\setminus\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m\cap\mathcal{C}_r} d^r_{c_2c_1}\cdot x^{old}_{c_1s}\cdot(x^{old}_{c_2s}-x^{new}_{c_2s})\\ \le\overline{BDW}^{in}_s \quad \forall s\in\mathcal{S}\end{aligned} \tag{2.32}$$

Moving on, when it comes to Constraints (2.10):

$$z_c = 1 \quad \forall c\in\mathcal{C}_{ob}$$

clearly variables z are not really necessary and can be directly replaced by their Equations (2.15). This way, Constraints (2.10) become:

$$\sum_{s\in\mathcal{S}} x^{old}_{cs}\cdot x^{new}_{cs} = 0 \quad \forall c\in\mathcal{C}_{ob}. \tag{2.33}$$

Regarding Constraints (2.14), that we have already replaced with (2.27), we can also absorb the fixed $t^{old}_{s_1s_2}$ traffics into the residual link capacity $\overline{K}_{ij}$, substitute variable z with its expression and finally have traffic demands and variables π partitioned by customers. The final version of these constraints is:

$$\begin{aligned}
&\sum_{c\in\mathcal{C}_m} f_{ij}^{c} - \\
&\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}} \sum_{r\in\mathcal{R}} \sum_{c_1,c_2\in\mathcal{C}_r\cap\mathcal{C}_m} x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}\cdot d_{c_1c_2}^{r}\cdot \pi_{c_1c_2}^{r} \quad - \\
&\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}} \sum_{r\in\mathcal{R}} \sum_{c_1\in\mathcal{C}_r\setminus\mathcal{C}_m,c_2\in\mathcal{C}_r\cap\mathcal{C}_m} x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}\cdot d_{c_1c_2}^{r}(1-\sum_{s\in S} x_{c_2s}^{old}\cdot x_{c_2s}^{new}) \quad - \\
&\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}} \sum_{r\in\mathcal{R}} \sum_{c_1\in\mathcal{C}_r\cap\mathcal{C}_m,c_2\in\mathcal{C}_r\setminus\mathcal{C}_m} x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}\cdot d_{c_1c_2}^{r}(1-\sum_{s\in S} x_{c_1s}^{old}\cdot x_{c_1s}^{new}) \\
&\sum_{s\in\mathcal{S}:(i,j)\in\mathcal{P}_{ss_0}} \sum_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}_r\cap\mathcal{C}_m} x_{cs}^{old}\cdot d_{cc_0}^{r}\cdot(1-\sum_{s\in\mathcal{S}} x_{cs}^{old}\cdot x_{cs}^{new}) - \\
&\sum_{s\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_0s}} \sum_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}_r\cap\mathcal{C}_m} x_{cs}^{old}\cdot d_{c_0c}^{r}\cdot(1-\sum_{s\in\mathcal{S}} x_{cs}^{old}\cdot x_{cs}^{new}) \\
&\leq \overline{K}_{ij} \quad \forall (i,j)\in\mathcal{E}. \qquad (2.34)
\end{aligned}$$

Then, Equations (2.17), already substituted with (2.21), can be further modified with partitioning by customers. Moreover, the constant traffic among fixed containers can be cut, leaving only positive and negative traffic variations related to (possibly) migrating containers. For this reason, let us rename t^{new} into Δt^{new}:

$$\begin{aligned}
&\Delta t_{s_1s_2}^{new} = \\
&\sum_{r\in\mathcal{R}} \sum_{c_1,c_2\in\mathcal{C}_r\cap\mathcal{C}_m} d_{c_1c_2}^{r}\cdot(w_{c_1s_1c_2s_2}^{r} - x_{c_1s_1}^{old}\cdot x_{c_2s_2}^{old}) \quad + \\
&\sum_{r\in\mathcal{R}} \sum_{c_1\in\mathcal{C}_r\cap\mathcal{C}_m,c_2\in\mathcal{C}_r\setminus\mathcal{C}_m} d_{c_1c_2}^{r}\cdot x_{c_2s_2}^{old}\cdot(x_{c_1s_1}^{new} - x_{c_1s_1}^{old}) \quad + \\
&\sum_{r\in\mathcal{R}} \sum_{c_1\in\mathcal{C}_r\setminus\mathcal{C}_m,c_2\in\mathcal{C}_r\cap\mathcal{C}_m} d_{c_1c_2}^{r}\cdot x_{c_1s_1}^{old}\cdot(x_{c_2s_2}^{new} - x_{c_2s_2}^{old}) \qquad \forall s_1,s_2\in\mathcal{S}. \qquad (2.35)
\end{aligned}$$

Its domain changes from $\mathbb{R}^+$ to $\mathbb{R}$ accordingly. Notice that in the first term, a positive contribution is given if both containers are assigned to the respective servers after migration, while a negative contribution is given if both containers were placed on the respective servers before migration. The two contributions cancel out each other if and only if both containers remain in the same position.

The linearization Constraints (2.18)-(2.20) and (2.24)-(2.26), when taking into account partitioning by customers and z's substitution, become:

$$w^r_{c_1 s_1 c_2 s_2} \leq x^{new}_{c_1 s_1} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.36}$$

$$w^r_{c_1 s_1 c_2 s_2} \leq x^{new}_{c_2 s_2} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.37}$$

$$w^r_{c_1 s_1 c_2 s_2} \geq x^{new}_{c_1 s_1} + x^{new}_{c_2 s_2} - 1 \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.38}$$

$$\pi^r_{c_1 c_2} \geq 1 - \sum_{s \in \mathcal{S}} x^{old}_{c_1 s} \cdot x^{new}_{c_1 s} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m \tag{2.39}$$

$$\pi^r_{c_1 c_2} \geq 1 - \sum_{s \in \mathcal{S}} x^{old}_{c_2 s} \cdot x^{new}_{c_2 s} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m \tag{2.40}$$

$$\pi^r_{c_1 c_2} \leq 2 - \sum_{s \in \mathcal{S}} (x^{old}_{c_1 s} \cdot x^{new}_{c_1 s} + x^{old}_{c_2 s} \cdot x^{new}_{c_2 s}) \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m. \tag{2.41}$$

Constraints (2.15) can be discarded along with variables z. Finally, the objective function (2.1),after cutting constant energy contributions and considering energy variations instead, together with traffic variations found in (2.35), becomes:

$$\begin{aligned} min \quad & \sum_{s \in \mathcal{S}} P_s \cdot \Delta u_s + P^{idle}_s \cdot (b^{new}_s - b^{old}_s) \\ & + \alpha \cdot \sum_{s_1 \in \mathcal{S}, s_2 \in \mathcal{S}} COST_{s_1 s_2} \cdot \Delta t^{new}_{s_1 s_2} \\ & + \beta \cdot \sum_{c \in \mathcal{C}_f} \sum_{s \in \mathcal{S}} x^{old}_{cs} \cdot x^{new}_{cs} \end{aligned} \tag{2.42}$$

where:

$$\Delta u_s = \frac{\sum_{c \in \mathcal{C}_{ob} \cup \mathcal{C}_f} CPU_{cs} \cdot (x^{new}_{cs} - x^{old}_{cs})}{CPU_s} \quad \forall s \in \mathcal{S} \tag{2.43}$$

is the variation of the server CPU utilization, obtained similarly to Constraints (2.28). Its domain becomes $\mathbb{R}$.

Notice that the fixed constant P^{idle}_s contributes as a cost when turning Oon-server s, while contributes as a gain when turning s off. The last term of equation (2.42) comes from the substitution of variables z and acts as a penalization term for not moving a container whose migration was suggested. The final formulation is a MILP and is summarized in the Appendix along with some additional notes.

4.6 Selecting the migrating VMs

Ideally the definition of the set of migrating VMs/containers $\mathcal{C}_m$ would be part of the problem itself. However, this would make the already difficult problem significantly harder, to the point of not being able to solve even small instances. Giving the possibility to choose which VMs/ containers need to be moved, where they should be moved and which links they will traverse while migrating would add too many degrees of freedom to the problem.

Thus, as is usually done in the literature when considering VM migration, we assume the set $\mathcal{C}_m$ to be pre-defined by a pre-processing phase and we focus on the reduced problem of finding optimal destinations and migration paths. In this section we provide a possible criterion for this pre-processing phase, which is the criterion that we will implement.

For each server $s \in \mathcal{S}$ whose CPU utilization exceeds the established threshold ρ_3 (e.g. 0.8), we want to select a set of VMs/containers whose migration would cause s's CPU utilization to decrease *just enough* to go back under the threshold. Moreover, knowing that we are in the context of traffic-aware optimization, these VMs/containers should have a high traffic rate between each other so that the optimization process will be able to re-allocate them near to each other without introducing too much additional traffic inside the core of the network. These set of VMs/containers, when considering all the servers, constitutes exactly the set $\mathcal{C}_{ob}$. Therefore, given the following parameters:

$\mathcal{C}_s$: set of containers currently allocated on server s,
U_{cs} : CPU utilization demand of container c on server s,
$t_{c_1c_2}$: traffic requirement between a pair of containers,
CPU_s : total CPU capacity, e.g. number of cores, of server s,
and decision variable

$$x_c := \begin{cases} 1 \text{ if container} c \text{ is chosen to be part of } \mathcal{C}_{ob}, \\ 0 \text{ otherwise,} \end{cases}$$

the problem of choosing which server must be migrated from a generic server s can be formulated as:

$$min \quad \theta \sum_{c \in \mathcal{C}_s} U_{cs} \cdot x_c \quad - \quad (1-\theta) \sum_{c_1, c_2 \in \mathcal{C}_s} t_{c_1c_2} \cdot x_{c_1} \cdot x_{c_2}$$

s.t.

$$\sum_{c \in \mathcal{C}_s} U_{cs} \cdot (1 - x_c) \leq \rho_3 \cdot CPU_s$$

$$x_c \in \{0,1\} \quad \forall c \in \mathcal{C}_s$$

The problem is itself a NP-hard problem. Indeed, taking $\theta = 1$, the formulation can be manipulated to match the standard formulation of a knapsack problem by swapping the meaning of variable x with its complement (1-x) so that x identifies the non-migrating containers, and then transforming the minimization problem into a maximization one. Instead, taking $\theta = 0$, the problem becomes to *maximize* the inter-traffic of migrating containers while satisfying the CPU utilization threshold. The term in the objective function is non-linear but could be linearized with the addition of new variables and constraints, as extensively done for both VMPP's and VMMP's formulations. Also, this term has a negative coefficient because it is inserted into a minimization function.

Since this is meant to be a relatively fast pre-processing phase that has to be applied to possibly thousands of servers, we are not interested in solving this sub-problem in an exact way. Instead, we apply the greedy Algorithm 1. The procedure repeatedly removes a VM/container from the considered server until the CPU threshold is satisfied. At each step, the procedure selects the VM/container that communicates the most with the ones already selected.

```
Input: Server $s$ with its VMs/containers
Output: VMs/containers $Selected$ whose migration is mandatory
$Candidates \leftarrow$ get all containers allocated on $s$;
$Selected \leftarrow \emptyset$;
while $s$'s CPU utilization is above threshold $\rho_3$ do
    $c \leftarrow$ find container in $Candidates$ whose traffic with $Selected$ is the
      highest;
    $Candidates \leftarrow Candidates \setminus \{c\}$;
    $Selected \leftarrow Selected \cup \{c\}$;
    remove $c$ from $s$ and update $s$'s CPU utilization ;
end
return $Selected$;
```

Algorithm 1: Greedy procedure for selecting migrating VMs/containers.

Then, after the choice of which VMs/containers belong to the set $\mathcal{C}_{ob}$ has been done, the exact same sub-problem can be defined to select VMs/containers that should belong to the set $\mathcal{C}_f$, i.e., VMs/containers whose migration is optional. The only difference is in the CPU threshold considered, which is now ρ_2 (e.g. 0.6). The same procedure is therefore invoked. Additionally, we defined $\mathcal{C}_f$ to include also all VMs/containers allocated on almost empty servers, i.e., servers whose CPU utilization is under threshold ρ_1 (e.g. 0.10 or 0.15). Their selection is trivial by definition.

As a side note, other reasonable criteria that could be applied/integrated include the minimization of the cardinality of $\mathcal{C}_{ob}$ and $\mathcal{C}_f$ or the minimization of the global size of the VMs/containers belonging to $\mathcal{C}_{ob}$ and $\mathcal{C}_f$. These would reduce respectively the combinatorial complexity of the related VMMP and the bandwidth requirements associated to the migration phase, the latter being relevant in case of a fully loaded data center with high links saturation. Also, clearly the cardinality of $\mathcal{C}_{ob}$ and $\mathcal{C}_f$ is, by definition, bound to grow with the size of the data center, i.e., with the number of servers. It could be the case, when considering large instances of thousands of servers, that the cardinality is too high to allow the problem to be solved in a reasonable amount of time. In those cases, the sub-problem and Procedure 1 can be modified to include only a part of the $\mathcal{C}_{ob}$ and $\mathcal{C}_f$ defined above. This can be done, for example, by sampling with a specified probability the output of Procedure 1.

4.7 Path selection sub-problem

4.7.1 High level description

As previously mentioned, the choice of using pre-computed paths is reasonable only if we make sure that they are periodically updated. This way, our paths adapt themselves to the changing configurations of the system and can provide meaningful solutions. Our goal here is to find a new set of paths that minimize the global traffic inside the network once the traffic's volume is fixed and given. The traffic is weighted by the lenght of the paths selected. The solutions needs to take into account the capacity of the physical links involved in each path. This problem is meant to be solved iteratively to approximate the best paths and costs used in the VMMP. In addition to this, the paths and costs found in the final iteration will be used in the short-term VMPP until the next migration phase.

4.7.2 Problem description

Given a network composed by:

- a set of nodes $\mathcal{N}$ representing switches and servers,
- a set of servers $\mathcal{S} \subset \mathcal{N}$,
- a set of arcs $\mathcal{E} \subset \mathcal{N} \times \mathcal{N}$ representing the physical links,

the path selection sub-problem consists in determining a way to route the traffic $t_{s_1 s_2}$ between each couple of servers so that a certain fraction γ of the capacity K_{ij} of each physical link (i,j) is not exceeded. Each traffic $t_{s_1 s_2}$ must traverse one and only one path (i.e. splittable paths are not allowed). The objective is to find the set of paths that minimize the overall traffic weighted by the length of each path.

Since the paths between servers and the Internet also have to be selected, we define two particular servers s_0 and t_0 that together represent the WAN. Unlike in VMMP, here there is no need to separate them from the set $\mathcal{S}$, keeping the notation more readable. In the graph, they are connected to the core edges by unidirectional dummy links with infinite capacity. In particular, s_0 is touched only by outgoing arcs (it is a source), while t_0 is touched only by incoming arcs (it is a sink). They are modeled as two distinct servers so that no internal traffic can pass through them exploiting the dummy links.

4.7.3 MILP formulation

By considering decision variables:

$$\phi_{ij}^{s_1 s_2} = \begin{cases} 1 & \text{if link } (i,j) \text{ is used in the path} \\ & \text{from server } s_1 \text{ to server } s_2 \\ 0 & \text{otherwise} \end{cases} \qquad \forall (i,j) \in \mathcal{E}, \quad \forall s_1, s_2 \in \mathcal{S}$$

$COST_{s_1 s_2} \geq 0$: number of switch-nodes traversed in the path from server s_1 to server s_2 $\quad \forall s_1, s_2 \in \mathcal{S}$

The problem can be formulated as follows:

$$min \sum_{s_1 \in \mathcal{S}, s_2 \in \mathcal{S}} COST_{s_1 s_2} \cdot t_{s_1 s_2} \tag{3.1}$$

s.t.

$$\sum_{j \in \delta^+(s_1)} \phi_{s_1 j}^{s_1 s_2} - \sum_{i \in \delta^-(s_1)} \phi_{i s_1}^{s_1 s_2} = 1 \quad \forall s_1, s_2 \in \mathcal{S} \tag{3.2}$$

$$\sum_{j \in \delta^+(s_2)} \phi_{s_2 j}^{s_1 s_2} - \sum_{i \in \delta^-(s_2)} \phi_{i s_2}^{s_1 s_2} = -1 \quad \forall s_1, s_2 \in \mathcal{S} \tag{3.3}$$

$$\sum_{i \in \delta^-(n)} \phi_{in}^{s_1 s_2} - \sum_{j \in \delta^+(n)} \phi_{nj}^{s_1 s_2} = 0 \quad \forall n \in \mathcal{N} \setminus \mathcal{S}, \quad \forall s_1, s_2 \in \mathcal{S} \tag{3.4}$$

$$\sum_{s_1,s_2 \in S} t_{s_1 s_2} \cdot \phi_{ij}^{s_1 s_2} \leq \alpha \cdot K_{ij} \quad \forall (i,j) \in \mathcal{E} \tag{3.5}$$

$$COST_{s_1 s_2} = \sum_{(i,j) \in \mathcal{E}} \phi_{ij}^{s_1 s_2} - 1 \quad \forall s_1, s_2 \in \mathcal{S} \tag{3.6}$$

$$\phi_{ij}^{s_1 s_2} \in \{0,1\} \quad \forall (i,j) \in \mathcal{E}, \quad \forall s_1, s_2 \in \mathcal{S},$$
$$COST_{s_1 s_2} \in \mathbb{R}^+ \quad \forall s_1, s_2 \in \mathcal{S}$$

A brief description of objective and constraints is presented below.

Equation (3.1) is again the objective function used by Meng et al.[12]. However, this time the traffic is known while the costs are not. It expresses the global traffic in the network weighted by the number of switches traversed.

Constraints (3.2) express the fact that, for each servers pair (s_1, s_2),exactly one arc exiting from the source s_1 must be selected (unsplittable path assumption).

Similarly, Constraints (3.3) deal with arcs entering in the destination.

Constraints (3.4) express the flow balancing in each intermediate node (switch). The traffic entering in a switch must be equal to the traffic exiting from it, and this must hold for the traffic related to every servers pair. Due to unsplittable path assumption, the traffic can be considered unitary here.

Constraints (3.5) are the links capacity constraints. The traffic traversing each physical link $(i.j)$ must not exceed a certain fraction γ of its capacity. This is motivated by the fact that we do not want to saturate any link. Otherwise, we would a priori violate one of the assumptions of the short-term VMPP (i.e., the link capacities are not a bottleneck).

Constraints (3.6) express the linking between variables ϕ and variables $COST$. The cost of a path depends on its length. The "-1" comes from the fact that the number of hops traversed in a path is one less than the number of traversed arcs.

CHAPTER 5
Heuristics

This chapter is devoted to the heuristics that we devised for the VMPP and the VMMP. Heuristics are presented with the help of pseudo-code and some illustrative examples. In Section 5.1 we describe the VMPP's heuristic, while in Section 5.2 we describe the VMMP's heuristic.

5.1 A GRASP heuristic for the short-term VMPP

Since our goal with the short-term VMPP is to tackle large instances of the problem within a short time limit (e.g., 1-2 minutes) in order to keep up with the constantly arriving new requests, we choose to develop a greedy heuristic. However, since a purely greedy approach is prone to very bad quality solutions on certain instances, we decide to device a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm (e.g., see [19]). Such an approach consists in two main steps: a *greedy randomized construction* of an initial solution and a *local search* procedure that starts from the initial solution found and stops in a nearby local minimum w.r.t. the considered neighborhood. The two steps are repeated multiple times until a stopping condition is satisfied, typically involving a time limit and/or a prescribed maximum number of iterations. The idea behind GRASP is to combine the speed of a greedy heuristic to find reasonably good initial solutions, the power of randomization to explore different regions of the solution space, and the ability of local search to improve the solution quality.

The general GRASP framework is summarized in Algorithm 2. In Section 5.1.1 we enter into the details of the greedy randomized procedure. Then, in

Section 5.1.2 we describe the local search procedure and the considered neighborhoods.

```
Input: Instance, Max_Iterations, randomization parameter $0 \leq \alpha \leq 1$
Output: best solution found
$Best\_Solution \leftarrow \emptyset$;
foreach $k = 1$ .. $Max\_Iterations$ do
    $Solution \leftarrow$ Greedy_Randomized_Construction($\alpha$);
    $Solution \leftarrow$ Local_Search($Solution$);
    if $f(Solution) < f(Best\_Solution)$ then
        $Best\_Solution \leftarrow Solution$;
    end
end
return $Best\_Solution$;
```

Algorithm 2: GRASP metaheuristic.

5.1.1 Greedy randomized construction

The purpose of this procedure is to build a solution from scratch while achieving a compromise between two goals: diversity and quality. First of all, multiple runs of the procedure on the same instance should produce a sufficiently wide spectrum of solutions in order to allow the exploration of many different regions of the solution space. Secondly, the procedure should use criteria that facilitate the creation of initial solutions with an already acceptable quality. To develop this procedure we refer to the general scheme described in [19] and to adaptations to the multi-knapsack problem (e.g., see [20]), with which our problem shares some similarities. Resende and Ribeiro's scheme, adapted to our specific problem is shown in Algorithm 3, where $\mathcal{S}_u$ is the set of under-utilized servers defined in Chapter 3 and *removeFirst* is a procedure that grabs the first element of a list, removing it from the list.

Given a pre-computed ordering of the list of new VMs, we iteratively consider the first VM of the list, remove it from the list and assign it to a "good" server. The server is chosen among a restricted list of candidates (RCL) that is built by looking at the incremental cost that the current VM would cause if assigned to each considered server. The list of new VMs is ordered following different criteria, such as: descending RAM order, descending CPU order, descending bandwidth order, descending DISK order, random order and times-

```
Input: Data center, list of new VMs to place, $0 \leq \alpha \leq 1$
Output: assignment vector of new VMs to servers
$Solution \leftarrow \emptyset$;
while $VMs \neq \emptyset$ do
    $vm \leftarrow$ removeFirst(VMs);
    foreach $s \in \mathcal{S}_u$ do
        evaluate incrementalCost(vm,s);
    end
    RCL $\leftarrow$ buildRCL($\alpha$);
    $\bar{s} \leftarrow$ randomly select a candidate from RCL;
    $Solution \leftarrow Solution \cup (vm,\bar{s})$;
end
return $Solution$;
```

Algorithm 3: Greedy_Randomized_Construction.

tamp order. The incremental cost amounts to the partial objective function related to the specific VM-server pair $(\bar{c},\bar{s})$:

$$\Delta f(\bar{c},\bar{s}) = \sum_{s_1,s_2 \in \mathcal{S}} \sum_{c_2 \in \overline{\mathcal{C}}_r \cup \mathcal{C}_r : \bar{c} \in \mathcal{C}_r} (d^r_{\bar{c}c_2} \cdot COST_{s_1 s_2} + d^r_{c_2\bar{c}} \cdot COST_{s_2 s_1}) \cdot x_{\bar{c}s_1} \cdot x_{c_2 s_2}. \quad (5.1)$$

In particular, since VMs are placed one by one, some terms of Equation () will refer to VMs whose assignment has not been decided yet. Therefore, the incremental cost computed at runtime will be an inaccurate estimation of Equation (5.1.1). Only the last VM to be placed will have the complete information to accurately compute it. When all candidate servers' costs have been evaluated, the best candidates are grouped in the RCL. In particular, the RCL includes all the candidate elements whose incremental cost satisfies the following condition:

$$cost_{\bar{c},\bar{s}} \leq min\_cost + \alpha \cdot (max\_cost \text{ - } min\_cost),$$

where min_cost and max_cost are the incremental costs of the least expensive and most expensive candidate respectively, and α is a randomization parameter in the $[0;1]$ interval. A choice of $\alpha = 0$ makes the procedure a deterministic greedy heuristic that always selects the candidate with minimum cost, while a choice of $\alpha = 1$ corresponds to a completely random heuristic that can select any candidate with equal probability. Usually we pick an α in the 0.1 - 0.2 range. Each server in the RCL is given a uniform probability of being selected. The procedure randomly selects one of them, assigns the current VM to it and updates all the residual capacities involved. The whole process is repeated for each new VM in the list. The greedy randomized construction stops either when there are no more VMs to be placed or when there is no feasible assignment for a VM. The latter case means that the previous assignments caused an infeasibility situation and the current GRASP iteration is interrupted. However, even if all the GRASP iterations fail to find a feasible solution, it is not a proof of the infeasibility of the problem.

Algorithm 3 describes the allocation of a generic set of new VMs. However, in the case in which a new customer allocates an entire new application, the criteria used in the procedure to select the RCL do not work as intended. Indeed, since these VMs are not communicating with any existing VM, the partial increment in the objective function will be equal to 0 for every feasible assignment. Moreover, their allocation comes with no pre-existing "constraints" and can be managed in a different way in order to facilitate the assignment of the other new VMs and to better utilize residual resources. The alternative procedure used in this case is reported in Algorithm 4, where *computeApplicationRequirements* is a procedure that computes the total sum of a prescribed resource requirement among a collection of VMs and *computeResources* is a procedure that computes the total sum of the residual capacities related to the same resource among all the servers in a rack.

In this case, we perform a two-level assignment. The first-level assignment consists in selecting a rack where to allocate the VMs of the new application, while the second-level assignment consists in assigning the VMs to the servers of the selected rack. Both assignments are performed with a RCL mechanism. For the first level, we build a RCL of racks with a criterion similar to 5.1.1. The cost of a rack is defined as the difference between the rack's residual capacity (related to the prescribed resource) and the application's requirements. The idea is to match applications and racks so as to efficiently utilize the resources. Similarly to Algorithm 3, we allow different criteria for selecting the proper rack, e.g., residual RAM, residual bandwidth or a combination of both and their corresponding VM requirements. Then, we pick a random rack, according to a uniform probability distribution, from the RCL and we apply the second-level assignment to the servers of that rack. Such servers are grouped in a list ordered by descending RAM residual capacity. Following the order of the list, the procedures tries to fit as much VMs as possible in each server before considering the next one in the order. While performing this inner packing, the order of VMs is chosen dynamically, meaning that at each step we select one of the VMs that benefit the most from being placed on the server that is currently being filled, i.e., the VMs that communicate the most with VMs already placed there. This is done through another RCL whose candidate elements are

```
   Input: Data Center,list of VMs of the new application, $0 \leq \alpha \leq 1$
   Output: assignment vector of new application's VMs to servers, list rest
           of VMs whose allocation failed
   $Solution \leftarrow \emptyset$;
   $req \leftarrow$ computeApplicationRequirements(VMs);
   foreach $r \in Racks$ do
       computeResources($r$);
   end
   RCL $\leftarrow$ buildRCL($\alpha$) of racks based on difference between resources
     and requirements;
   select a rack $\bar{r}$ at random from RCL;
   $rackServers \leftarrow$ sort servers in $\bar{r}$ by descending residual RAM;
   $s \leftarrow$ removeFirst($rackServers$);
   while $VMs \neq \emptyset \wedge rackServers \neq \emptyset$ do
       foreach $vm \in VMs$ do
           computeProfit($v$,$s$);
       end
       RCL $\leftarrow$ build RCL($\alpha$) of VMs with most profits;
       $\overline{vm} \leftarrow$ select a cadidate at random from RCL;
16     if $canBeAllocated(\overline{vm}, s)$ then
           $Solution \leftarrow Solution \cup (\overline{vm}, s)$;
       end
       else
           if $rackServers = \emptyset$ then
               break;
           end
           $s \leftarrow$ removeFirst($rackServers$);
           goto line 16;
       end
   end
   $rest \leftarrow$ VMs;
   return ($Solution, rest$);
```

Algorithm 4: Greedy_New_App_Construction.

the new VMs not yet assigned and the profit related to each of them amounts to the traffic with VMs already assigned to the current server. In the case in which the procedure fails to assign some VMs to servers in the considered rack, these remaining VMs are collected in the list *rest* and their placement is later decided by invoking Algorithm 3.

Since we defined different criteria for the ordering of VMs and racks in the two procedures, our implementation of the GRASP runs several greedy randomized constructions in parallel, one for each combination of orderings, including the order in the invocation of Algorithms 3 and 4, for a total of 36 threads working simultaneously and independently from each other.

In its basic version, when building the RCL, Algorithm 3 considers all the feasible servers, but other criteria can be employed to improve the speed of the procedure while still preserving the quality of the initial solution. For example, we tried to develop a server indexing mechanism based on an ordered binary tree to access in logarithmic time the server whose residual resources are just enough to host a specified VM. This way, we can avoid to consider servers corresponding to a lower index in the ordering. Unfortunately, this technique did not provide significant advantages, probably due to the fact that, on average, when the servers are not fully loaded, the number of servers considered does not change significantly and the gain evens out with the slightly higher cost (in terms of computing time) of moving inside the binary tree.

5.1.2 Local search

Starting from a randomized greedy initial solution, we apply a local search procedure (see Algorithm 5) based on a deterministic neighborhood exploration with a best-improve strategy. Since the entire GRASP requires several iterations with randomization, the local search phase overall acts as a local search with multiple restarts, a common technique used to avoid poor quality local minima. In particular, we selected two types of neighborhoods:

```
Input: starting solution $Solution$
Output: solution that is a locally optimal
while Solution is not locally optimal do
    Find $s' \in$ Neighborhood($Solution$) with $f(s') < f(Solution)$;
    $Solution \leftarrow s'$;
end
return $Solution$;
```

Algorithm 5: Local_Search scheme.

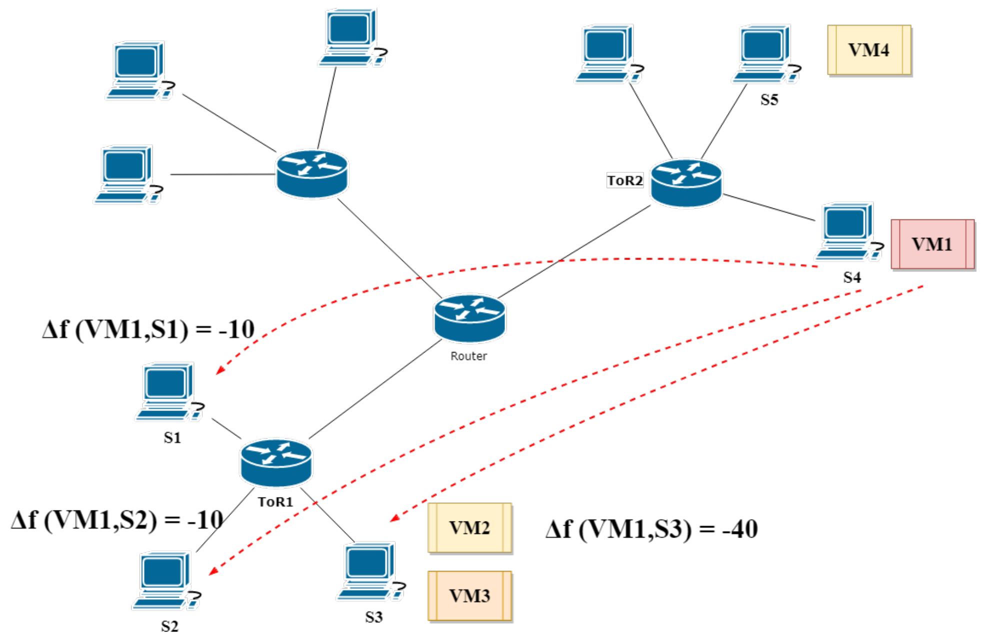


Figure 5.1: *One-Move-Neighborhood example - Part 1 - VM1 can be moved on servers S1, S2 or S3 to improve the objective function's value. Dashed-arrows represent improving moves.*

One-Move-Neighborhood: contains all the solutions that differ from the current solution for the placement of only one new VM. The size of the neighborhood is upper-bounded by the cardinality of the set of new VMs times the cardinality of S_u of under-utilized servers. Therefore,an exhaustive exploration has a computational complexity of $\Theta(|\mathcal{C}| \cdot |\mathcal{S}_u|)$ for each iteration of the local search. The same neighborhood is also used in [14],

One-Swap-Neighborhood: contains all the solutions obtained by exchanging the position of any pair of new VMs with respect to the current solution. The size of the neighborhood is upper-bounded by the square of the cardinality of the set of new VMs. Actually, due to symmetry of the pairs, half of the neighborhood is redundant. The computational complexity associated to an *exhaustive* visit is $\Theta(|\mathcal{C}|^2)$ for each iteration of the local search.

To illustrate these two neighborhoods we provide two toy examples.

Example 1 For the *One-Move-Neighborhood*, we consider a situation with three racks of servers and four new VMs (see Figure 5.1). VM1 and VM4 are placed in the same rack, but on different servers, namely S4 and S5, while VM2 and VM3 are placed in a different rack, but on the same server S3. We assume that this configuration is the result of either the greedy randomized construction or a previous iteration of the local search itself. Due to particular values in the traffic matrix, it is possible to improve the solution by changing the position of VM1. Given the objective function (see Equation 1.1 in Chapter 3), it is convenient to move VM1 the rack where VM2 and VM3 are located. Multiple alternatives (see dashed-lines in Figure 5.1) are possible, since servers in the same rack have all the same distance between each others. However, the best choice is to place VM1 on the same server as VM2 and VM3, i.e., server S3, so that the traffic between them does not need to traverse the network and its related costamounts to 0. Therefore, the local search procedure moves VM1 to server S3. The resulting solution, with an associated objective function's improvement of 40, is depicted in Figure 5.2. Due to this operation, the assignment of VM4 is no longer locally optimal and must be changed. Its new destination will be for sure in the same rack as the other VMs. However, putting VM4 in the exact same server as VM1, VM2 and VM3 is not an option. Indeed, server S3 is almost fully loaded and its residual resources are not enough to also accommodate VM4. The only other alternatives are the other servers in the same rack, i.e., S1 and S2. From the objective function's point of view they are equivalent, therefore some arbitrary tie-breaking criterion establishes which one of them will host VM4, e.g., server S1.

The resulting solution, whose objective function's value benefits from an improvement of 30, is locally optimal (but not optimal, as we shall see). The *One-Move-Neighborhood* allowed us to find a better placement for the VMs and to group them inside the same rack, which is consistent with the general VM Placement's goal of freeing the core of the network by concentrating the traffic down in the racks.

Example 2 For the *One-Swap-Neighborhood*, we continue from the scenario derived in Example 1. We have a solution which is locally optimal. There is no incentive to move any single VM alone because such change would only worsen or, at best, not improve, the objective function's value. In particular, moving any of VM1, VM2 and VM3 would introduce in the network some non-zero traffic that at the moment is managed in memory. On the other hand, VM4 cannot be moved on the same server as the other VMs due to capacity limits, and moving it anywhere else would mean canceling the last performed move, that was the best possible move available among single-VM moves. However, as shown in Figure 5.3, due to particular values in the traffic matrix, there is a better solution in which VM4 and VM2 are exchanged. Reaching this solution

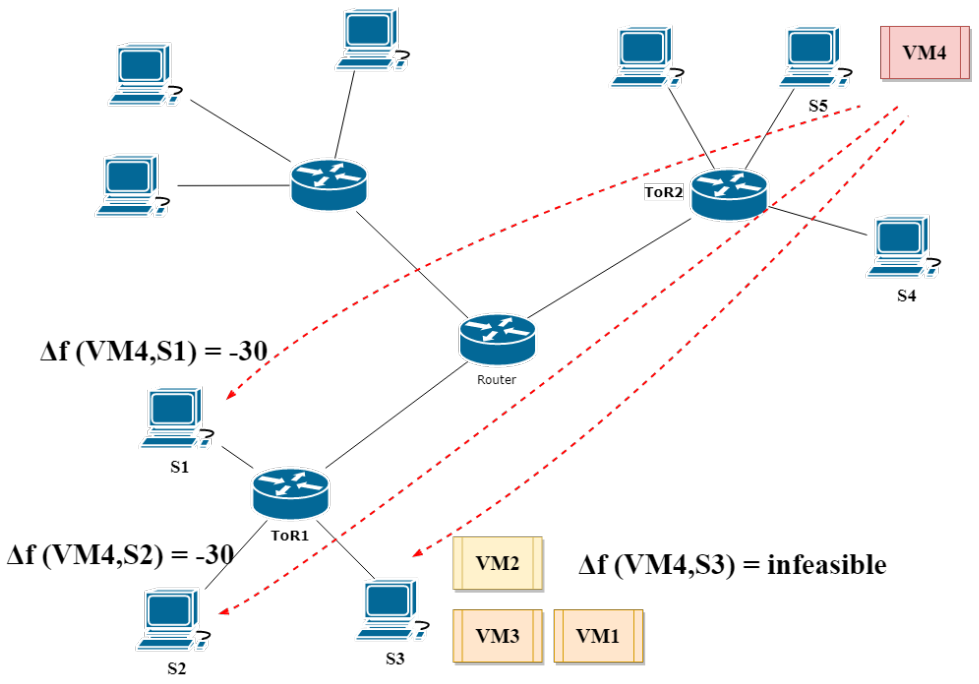


Figure 5.2: *One-Move-Neighborhood example - Part 2 - VM4 can be moved on servers S1 or S2 to improve the objective function's value. Dashed-arrows represent improving moves.*

from the current solution requires two single-VM moves, but any combination of the two leads to a worse solution at first, only to see the benefits of it with the second move. Therefore, the local exploration of the *One-Move-Neighborhood* is not able to find that solution. On the other hand, the *One-Swap-Neighborhood* is capable of detecting such solutions. Thus, VM2's location and VM4's locations are swapped to obtain a new better solution with an improvement of 10 in the objective function's value. Although there is no guarantee that this solution is globally optimal, the exploration of both neighborhoods can bring significant improvements.

These two types of neighborhoods are then refined with more efficient versions to avoid the exhaustive exploration of the neighborhood, at the price of possibly losing some good solutions. As we shall see, it very rarely happens in practice.

***One-Move-Neighborhood* variants** If the *One-Move-neighborhood* is implemented as defined, the local search becomes very slow in case of large data centers. At the same time, the structure of the problem allows us to easily discard a priori

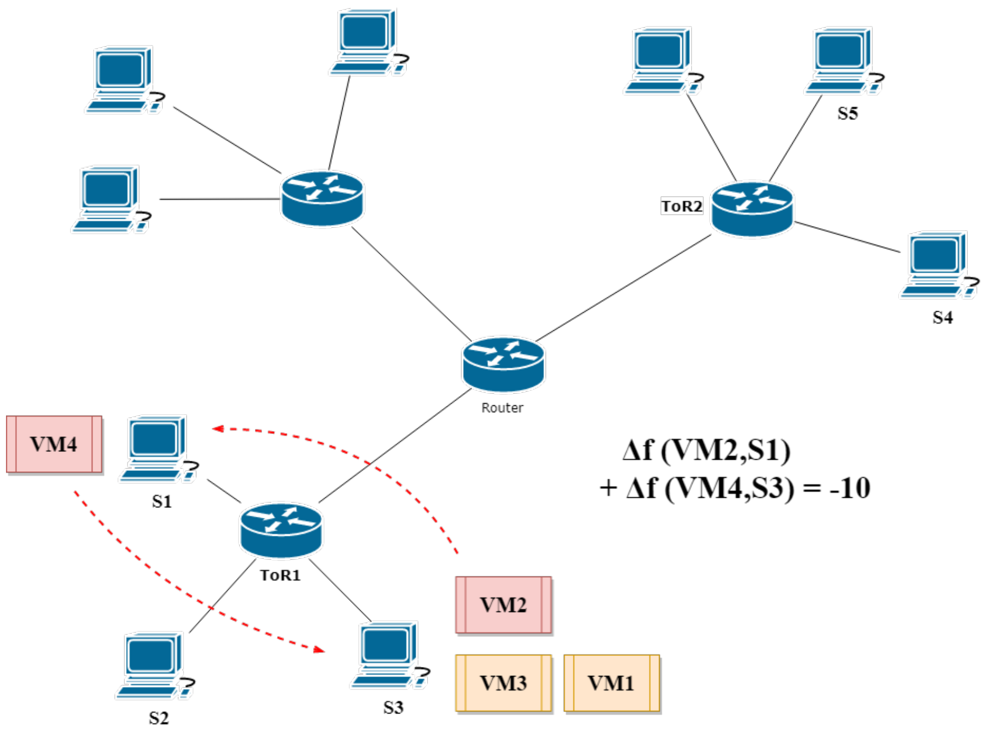


Figure 5.3: *One-Swap-Neighborhood example - VM2 and VM4's assignments can be swapped to improve the objective function's value. The dashed arrows represent the two single moves involved in the swap.*

some servers whose selection is (very) unlikely to provide good solutions. Since we know that VMs may communicate only with VMs of the same customer and that the cost of the communication depends on the distance between servers. Whenever we consider to move a VM, we can consider only pods that contain at least one VM belonging to the same customer.All the other ones, which are the majority of the pods, can be safely neglected during the exploration, providing a significant speed-up. We call this neighborhood *One-Move-Neighborhood-Large*. The size of such neighborhood is

$$O(|\mathcal{R}| \cdot |\mathcal{C}_r^{max}| \cdot (|\mathcal{C}_r^{max}| + |\overline{\mathcal{C}}_r^{max}|) \cdot PodSize)$$

for each local search iteration, where $\mathcal{R}$ is the set of customers, $\mathcal{C}_r^{max}$ refers to the largest set of new VMs belonging to a single customer, and $\overline{\mathcal{C}}_r^{max}$ refers to the largest set of old VMs belonging to a single customer. However, the typical scenario is very far from the worst case to which the bound refers. In a general situation, the term $|\mathcal{C}_r^{max}| + |\overline{\mathcal{C}}_r^{max}|$ amounts to at most 2, usually 1, because any reasonable optimization process should put VMs of a same application in the same pod or at most in two different pods in case of particularly tight resource capacities. Therefore, the size of the neighborhood in the general case is

$$\Theta(|\mathcal{R}| \cdot |\mathcal{C}_r^{max}| \cdot PodSize),$$

which, hiding the partitioning, is equivalent to

$$\Theta(|\mathcal{C}| \cdot PodSize),$$

clearly a significant improvement with respect to the original version of the neighborhood. However, we can do better. For realistic instances, the pods are so large that the above reasoning can be applied to more granular scales. The objective function of the problem is such that we know a priori that the distance between servers in the same rack is strictly smaller than the distance between servers in the same pod but in different racks. Therefore, we can consider only *racks* that contain VMs of the same customer or, pushing the reasoning to its limit, we can even consider only *servers* that contain VMs of the same customer. We refer to these two neighborhood versions as *One-Move-Neighborhood-Medium* and *One-Move-Neighborhood-Small* respectively. With a similar reasoning, we can provide a worst-case bound of

$$O(|\mathcal{R}| \cdot |\mathcal{C}_r^{max}| \cdot (|\mathcal{C}_r^{max}| + |\overline{\mathcal{C}}_r^{max}|) \cdot RackSize)$$

for *One-Move-Neighborhood-Medium*'s size, that becomes

$$\Theta(|\mathcal{C}| \cdot RackSize)$$

in the general case. Instead, the worst case and general case related to the *One-Move-Neighborhood-Small* are not different enough to require two distinct characterizations. Their size is roughly

$$O(|\mathcal{C}| \cdot (|\mathcal{C}_r^{max}| + |\overline{\mathcal{C}}_r^{max}|)).$$

The advantage of the small neighborhood version is that, in general, VMs of the same customer are not placed all on different servers, but are rather grouped in a small subset of servers, causing the second term to be a number smaller than $RackSize$.

Of course, the more restricted the search is, the more it is likely to miss good solutions. This chance increases when pods, racks and servers are heavily loaded. However, all these variants of the neighborhood are characterized by a nice property: each one is strictly included in the larger ones. Thanks to this property the local search can be set up as a Variable Neighborhood Search.

The idea is that the procedure repeatedly explores a small neighborhood until it stops in a local minimum. Then, a larger version of the neighborhood is selected to try to escape from the local minimum. This way, the reduced computational requirements of the smaller neighborhood is combined with the higher accuracy of the larger one. We found that a good choice is to use just *One-Move-Neighborhood-Small* followed by *One-Move-Neighborhood-Medium*.

***One-Swap-Neighborhood* variants** Similarly, we can provide a restricted version of this neighborhood. Since we know a priori that VMs of *different* customers do not communicate, it is unlikely that exchanging their positions will bring benefits to the objective function's value, so we can discard all those pairs. Intuitively, the swap's purpose is to explore combinations whose occurrence is prevented by the particular ordering criteria that we chose in the greedy randomized phase. Swapping VMs of different applications, in general, only put them further away from the VMs with which they communicate. This simple reasoning allows to exploit the partitioning of the traffic matrix to greatly reduce the size of the neighborhood, since only intra-customer swaps are performed. We call this version *One-Swap-Neighborhood-Small*. Its computational complexity for both worst case and general case is

$$\Theta(|\mathcal{R}| \cdot |\mathcal{C}_r^{max}|^2),$$

which is equivalent to

$$\Theta(|\mathcal{C}| \cdot |\mathcal{C}_r^{max}|),$$

clearly an improvement with respect to the basic version of the neighborhood. Again, half of the VM pairs are redundant, so the actual complexity can be further reduced. Similarly to the *One-Move-Neighborhood* case, we can use the smaller version of the *One-Swap-Neighborhood* combined with the larger one.

From empirical observations we found that, when used alone, the *One-Move-Neighborhood* is able to end up much closer to the optimal solution than the *One-Swap-Neighborhood*, but also that the latter allows to escape from local minima in which the other one does get stuck. Therefore, the final configuration of our procedure cycles between neighborhoods, specifically in this order: *One-Move-Neighborhood-Small*, then *One-Move-Neighborhood-Medium*, then *One-Swap-Neighborhood-Small*, then *One-Swap-Neighborhood*, then back to the first one. The stopping condition is satisfied when the search finds a solution that constitutes a local minimum for all four neighborhoods at the same time.

5.2 GRASP's extension for the long-term VMMP

In this section we show that the previously described heuristic can be extended to tackle the more challenging VMMP. As it clearly appears from the formulations provided in Chapters 3 and 4, the VM migration problem can be considered as an extension of the placement problem with the addition of constraints on the network resources. A VM that is selected to be reassigned can be viewed as a new VM entering the system, provided that the resources that it was consuming before are properly accounted for.

Given that the VMs selected for migration are the output of a separate preprocessing phase, discussed in Chapter 4.6, the heuristic can be divided in two steps: a short initial phase in which all VMs belonging to both sets $\mathcal{C}_{ob}$ (mandatory migration) and $\mathcal{C}_f$ (optional migration) have their assignments undone, and the actual re-assignment phase, where VMs are reassigned, possibly even on the same server as before in case of VMs belonging to $\mathcal{C}_f$. The first step accounts for the resources freed by VMs that are moved, while the second step accounts for resources consumed by the same VMs in the new allocation. In the end, the objective function f (see Equation 2.1) is the result of the difference between the following quantities computed for each VM:

$$f = \sum_{c \in \mathcal{C}_{ob} \cup \mathcal{C}_f} Cost_c^{t+1} - Cost_c^t$$

where the time index $t+1$ refers to the configuration post-migration and t to the configuration pre-migration. The two quantities are characterized by a fundamental difference: the cost and use of resources related to a VM pre-migration can be computed exactly without the need of any decision, while the cost post-migration depends on the new assignment yet to be determined. In case of a VM that is re-assigned to the same server as before, the two terms simply cancel out each other, bringing neither an increment nor a decrease of cost for that specific VM.

When dividing the procedure into two steps, the second step becomes almost exactly another VM Placement Problem, with a few differences:

- the objective function includes a power consumption term and an incentive term for migration,
- constraints on the links' utilization are now explicit, meaning that we have to provide actual migration paths that satisfy them.

The second point is not related to the objective function. Constraints on the links bandwidth only act on feasibility. This is due to the decision of aiming at optimizing the configuration of the system in the following long time interval rather than during the short transitory phase that is the Live-Migration phase.

As for the VMPP, the re-allocation step consists in repeated iterations of a greedy randomized construction step, described in Section 5.2.1, followed by a local search procedure, described in Section 5.2.2.

5.2.1 Greedy randomized construction

The greedy randomized construction for the VMMP is summarized in Algorithm 6. The main structure is the same RCL mechanism described in Section 5.1.1, with just a few variations that consist in a different definition of the incremental costs and an additional procedure, the *canMigrate* procedure, that checks the existence of a feasible migration path from the old placement of the current migrating VM to the target server selected for the new assignment. The main block is repeated twice to put emphasis on the fact that VMs in list $\mathcal{C}_{ob}$, whose migration is mandatory, are processed first, so that they are given more flexibility and are less likely to encounter saturated resources causing infeasibility. For both lists $\mathcal{C}_{ob}$ and $\mathcal{C}_f$ we consider the same ordering strategies discussed in the VMPP's GRASP, except for the timestamp order that has no meaning here.

As previously mentioned, the procedure depends on sub-procedures *incrementalCost* and *canMigrate*. The first one accounts for the increment in the objective function brought by the reassignment of each VM. The cost of re-assigning VM $\bar{c}$ to server $\bar{s}$ amounts to:

$$cost(\bar{c},\bar{s}) = (P_{\bar{s}}^{max} - P_{\bar{s}}^{idle}) \cdot u_{\bar{c},\bar{s}} + \beta \cdot \Delta f^{new}(\bar{c},\bar{s}) \quad (+P_{\bar{s}}^{idle}). \tag{5.2}$$

where β is the traffic coefficient, Δf^{new} is the quantity defined in Equation (5.1.1), except that variable x is replaced by variable x^{new}. Beside mixing power consumption and traffic, a crucial part is played by the fixed costs of turning on a server, indicated between parenthesis. Indeed, each VM can, occasionally, cause these fixed costs to appear or disappear in addition to its own partial cost. The procedure handles these fixed costs exactly in this trivial way, accounting them only when they occur, i.e., adding the fixed cost when a VM is re-assigned to an empty server. Note that the incremental cost defined above only accounts for $Cost_c^{t+1}$. This is due to the fact that, when building the RCL, the term $Cost_c^t$ does not depend on the candidate server and behaves as a constant that does not influence the decision. It will be taken into account only when attaching a value to the initial solution found at the end of the greedy randomized construction.

The second sub-procedure is instead summarized in Algorithm 7, where a Path is a collection (e.g., a list) of links and a flow is a set of paths, and therefore links, each attached with the information (flowsize) about the amount of

```
   Input: Data center, list of $\mathcal{C}_{ob}$, list of $\mathcal{C}_f$, $0 \le \alpha \le 1$
   Output: new allocation vector of VMs $\in \mathcal{C}_{ob} \cup \mathcal{C}_f$
   $Solution \leftarrow \emptyset$;
   while $\mathcal{C}_{ob} \neq \emptyset$ do
       $vm \leftarrow$ removeFirst($\mathcal{C}_{ob}$);
       foreach $s \in \mathcal{S}$ do
           evaluate incrementalCost(vm,s);
       end
       RCL $\leftarrow$ buildRCL($\alpha$);
8      $\bar{s} \leftarrow$ randomly select a candidate from RCL;
       if canMigrate($vm, \bar{s}$) then
           update links and go on ;
       end
       else
           goto line 8;
       end
       $Solution \leftarrow Solution \cup (vm,\bar{s})$;
   end
   while $\mathcal{C}_f \neq \emptyset$ do
       $vm \leftarrow$ removeFirst($\mathcal{C}_f$);
       foreach $s \in \mathcal{S}$ do
           evaluate incrementalCost(vm,s);
       end
       RCL $\leftarrow$ buildRCL($\alpha$);
23     $\bar{s} \leftarrow$ randomly select a candidate from RCL;
       if canMigrate($vm, \bar{s}$) then
           update links and go on ;
       end
       else
           goto line 23;
       end
       $Solution \leftarrow Solution \cup (vm,\bar{s})$;
   end
   return $Solution$;
```

Algorithm 6: Greedy_Randomized_Construction extended to VMMP.

```
Input: a VM $vm$ to migrate, destination server $s$
Output: True if migration is feasible, False if it is not, migration flows
$\bar{s} \leftarrow$ obtain old placement of $vm$ ;
$Paths \leftarrow k\text{-}shortestPath(\bar{s}, s)$;
try to saturate $Paths$ in order ;
if bandwidth capacity is enough then
    return (True, $(Paths, flowsize)$) ;
end
else
    return (False, $\emptyset$) ;
end
```

Algorithm 7: *canMigrate* sub-procedure.

bandwidth used by the migration of the current VM on that specific link. The splittable-paths feature of the problem is handled with a call to a *k-shortest path* procedure. The distance measure related to a link $l = (i, j)$ is equal to

$$d(i,j) = \frac{1}{residualCapacity(l)}.$$

This way, links whose capacity is close to saturation are less likely to be considered. We recommend a choice of $k = 2$ or $k = 3$ that is a trade-off between considering many multi-paths, thus providing more flexibility, and considering only the shortest path, which is the fastest option. We did not develop the *k-shortestpath* procedure. Instead, we approximate it invoking k times the implementation of the Dijkstra shortest path algorithm provided by the Java external library *JGraphT* [21]. The computational complexity of a Dijkstra shortest path algorithm is much lower than the computational complexity of a k-shortest paths algorithm, thus making the procedure faster.

Algorithm 7 has a special case. It may happen that a VM is re-assigned to the same server as before. In such case, there is no traffic related to the migration burst, but rather the links need to be loaded with the normal application traffic of the VM. This is due to the fact that these particular VMs, that are not known in advance, will actually remain active and continue their normal behavior during the migration phase.

As for the VMPP's heuristic, we then recognized a specific case in which Algorithms 6 and 7 can be replaced by a more specialized alternative. Indeed, when selecting in each server the VMs that need to be moved, the preprocessing phase described in Chapter 4.6 will sometimes select isolated VMs and other times will select groups of VMs belonging to the same application/customer.

In the latter case, we want the components of these groups to be close to each other also in the new configuration so that their traffic do not get scattered across the data center. Thus, instead of risking the chance of separating those VMs, we move them together. These groups, or *small clusters*, are considered each as a single aggregated VM to which the same $canMigrate$ sub-procedure is applied. The difference is that in this case the procedure does not take a single server as destination, but rather a Top of Rack (ToR) switch. After having verified that the cluster can migrate through the network and reach such ToR switch, a small inner GRASP similar to the one used for the VMPP is run for few iterations with the aim of finding a good assignment of the VMs inside the related rack. This inner GRASP is sound since the link capacities inside the rack perfectly match the bandwidth capacities associated to each server in the VMPP. The crucial part here is the choice of the rack. We try to estimate the incremental cost that a cluster would bring if destined to each possible rack. We do this by checking if the resource requirements of the cluster are less than the residual capacities of the *already turned on* servers inside the rack. If that is the case, the estimated incremental cost is simply the sum of the linear power-consumption terms and the weighted traffic terms of the whole cluster. This is a quite accurate estimate since, independently from the inner assignment that will be done later on, servers in the rack are homogeneous and they all have the same distance from servers in other racks. Instead, if the residual capacity of turned-on servers is exceeded, the fixed cost for turning on another server inside the rack is added to the incremental cost.

The two resource requirements/capacities considered are related to CPU and RAM. These two, together with the five different ordering considered for Algorithm cmp-greedy-rand-constr, create 10 possible configurations. therefore, we execute 10 GRASP procedures in parallel.

5.2.2 Local search

The local search is set up exactly as in the VMPP's heuristic. The two types of neighborhoods, namely the *One-Move-Neighborhood* and the *One-Swap-Neighborhood*, are adapted to account for the additional bandwidth constraints on the links, i.e., to establish if a solution in a neighborhood is feasible, the same procedure $canMigrate$(see Algorithm 7) is invoked. As for the VMPP, the two basic versions of the neighborhood are specialized in *One-Move-Neighborhood-Medium*, *One-Move-Neighborhood-Small* and *One-Swap-Neighborhood-Small*. They are visited in the same order as specified in Section 5.1.2 until a common local minimum is reached. Due to the heavier computational effort required by the network component of the problem, the *One-Swap-Neighborhood* in its largest version is discarded and only the small one is actually used.

5.2.3 Path Relinking

Since the VMMP turns out to be more challenging than the VMPP, we decided to add a further step to the GRASP procedure. This step is based on a technique called *Path Relinking*, which is described by Resende and Ribeiro in [22] in the GRASP framework. The idea behind it is to combine pairs of the best solutions found by the GRASP to find new solutions whose quality is better than the best one found so far.

A solution of the VMPP can be represented as an assignment vector $\underline{v} \in \mathcal{C} \times 1$ of VMs to servers, where $\mathcal{C}$ is the set of VMs to be assigned. Each position $i \in 1..|\mathcal{C}|$ in the vector stores the information about the server where one of the VMs will be placed. Since the VMMP is an extension of the placement problem, the same can be done for the set $\mathcal{C}_m = \mathcal{C}_{ob} \cup \mathcal{C}_f$. In addition to this, a solution of the VMMP has to store the path/paths used by each migrating VM. Given two solution vectors $\underline{s}$ and $\underline{t}$ such that $\underline{s} \neq \underline{t}$, it is always possible to start from $\underline{s}$ and reach $\underline{t}$ through repeated changes of the components of $\underline{s}$ whose content differs from the same positions in vector $\underline{t}$. This sequence of changes constitutes a path connecting the two solutions. The interesting aspect of traversing such path is that each step of the path represents an element in the solution space, i.e., an assignment vector of VMs to servers. Some of these intermediate solutions are infeasible or worse (in terms of the objective function) than $\underline{s}$ and $\underline{t}$, but there is also the possibility to find better solutions.

Example A toy example is shown in Figure 5.4, where two solutions are linked together by a path of lenght = 4, i.e., four elements in the starting solution's vector $\underline{s}$ need to be modified to reach the target solution $\underline{t}$. The vectors used in the example represent the assignment of five VMs to five servers. Each cell of the vector is related to a VM (order is assumed to be the same) and the content of each cell stores a unique server's identifier. The intermediate solutions $\underline{v}', \underline{v}''$ and $\underline{v}'''$ are new solutions that were not known before applying *Path Relinking* and that are possibly better than all the known solutions.

The Path Relinking procedure is shown in Algorithm 8. The procedure iteratevely transforms the starting solution into the target solution by changing one position of the current solution's vector at a time. The list $\Delta_{x,t}$ of the candidate positions to be modified is initialized as the list of all positions whose content differs in between the two solutions. At each iteration, the position whose update generates the best new solution is selected to be modified. Differently from the GRASP, this process is deterministic, with no RCL. When an intermediate improving solution is detected, a local search procedure is invoked to explore the neighborhood of such solution which is likely to contain even better ones,

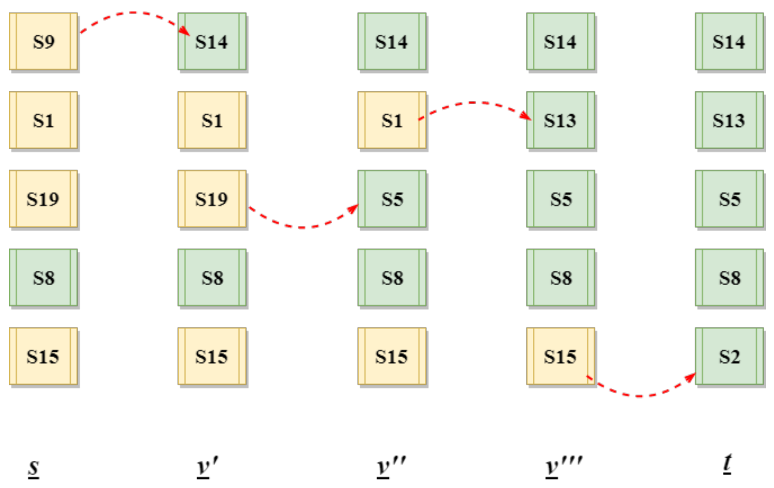


Figure 5.4: *Path Relinking example - path from solution $\underline{s}$ to solution $\underline{t}$, passing through intermediate solutions $\underline{v}'$, $\underline{v}''$ and $\underline{v}'''$.*

```
Input: starting solution $\underline{s}$, target solution $\underline{t}$
Output: solution better than $\underline{s}$ and $\underline{t}$

$\underline{x}_{best} \leftarrow$ best among $\underline{s}$ and $\underline{t}$;
$\underline{x} \leftarrow \underline{s}$;
$\Delta_{x,t} \leftarrow$ compute list of vector positions that differ between $\underline{s}$ and $\underline{t}$;
while $\Delta_{x,t} \neq \emptyset$ do
    foreach $i \in \Delta_{x,t}$ do
        evaluate benefit in objective function when changing element in
          position $i$;
    end
    $i \leftarrow$ position whose change benefits the most;
    $\underline{x}^i \leftarrow \underline{t}^i$;
    updateFlows();
    if $f(\underline{x}) < f(\underline{x}_{best})$ then
        $\underline{x}_{best} \leftarrow$ localSearch($\underline{x}$);
    end
    remove $i$ from $\Delta_{x,t}$;
end
return $\underline{x}_{best}$;
```
Algorithm 8: Path Relinking procedure for VMMP.

since the related region of the solution space was not visited by the GRASP. The local search procedure used here is exactly the same as the one used in the GRASP. At the end of the Path Relinking step, only the best solution found is kept.

The sub-procedure *updateFlows* is a procedure that cancels the flows used by the element in position i of $\underline{x}$ and applies the flows used in $\underline{t}$, i.e., since the destination of the i^{th} VM's migration has changed, the path/paths traversed during migration are updated accordingly. Be careful that, as in the case of the *canMigrate* procedure, if the i^{th} VM either was not migrating in solution $\underline{x}$ or is not migrating in solution $\underline{t}$, the flows needs to be considered differently, since the flows involved are normal traffic's flows instead of migration's flows. The Path Relinking procedure is implemented with some levels of flexibility:

- Instead of completing the entire path from the starting solution to the final solution, there is the option to stop at a prescribed percentage of the path. This is done by changing the condition on the while loop in Algorithm 8 so that the algorithm stops when list $\Delta_{x,t}$ is smaller than a a certain percentage of its original size. This can be helpful since for some problems it has been observed that good solutions are more likely to be found near the starting or target solutions. In such cases, focusing the search only on those specific portions of the path can save some computing time.

- The path of intermediate solutions connecting $\underline{s}$ and $\underline{t}$ is not unique. In fact, the path depends on the order in which the components of the vector $\underline{x}$ are modified. Therefore, the number of possible paths is $|\Delta_{x,t}|!$, i.e., factorial in the cardinality of the length of a single path. The basic version of the procedure exploits the reasonable criterion of choosing at each step the vector component whose change constitutes the best local choice. However, following the rationale behind the whole GRASP approach, this choice could be enriched with the addition of a randomization factor and a RCL.

- It is not mandatory to modify exactly one positions of the vector $\underline{x}$ at each step. There is also the possibility to change multiple vector components at the same time, thus producing fewer intermediate steps and visiting fewer solutions. The advantage is that this kind of approach speeds up the procedure.

Of course, the three modifications described above can be combined to obtain the best compromise for the particular problem's instances considered.

Algorithm 8 is used inside a larger procedure, i.e., Algorithm 9. As previously mentioned, the overall algorithm tries to combine multiple solutions coming from the GRASP to find new solutions. To achieve this goal, a pool of solutions of limited size is initialized at the beginning of the algorithm. At first, the pool contains exactly a subset of solutions coming from the GRASP algorithm. Then, the Path Relinkig procedure is called on each pair of solutions in the pool and the results are collected in a list of candidate solutions. The pool of solutions is updated using the list of candidate new solutions. Since we impose the pool's size to be limited, the insertion of a new solution can cause another solution to exit from the pool. The exiting solution is chosen followig two criteria: quality and diversity. We want to maintain a pool with good solutions that are as different as possible from each other so that they drive the path relinking procedure to explore more regions of the solution space. Therefore, when a new solution $\underline{y}$ is added, we compute the list of solutions in the pool that are worse than $\underline{y}$ and among them we discard the most similar to $\underline{y}$, i.e., the one whose number of different elements in the vector is the lowest. If the pool is actually updated, a new iteration of the algorithm is performed, applying the Path Relinking Algorithm 8 to all the new solution pairs available, i.e., all the pairs involving at least one soluton that has just entered the pool. The algorithm stops when the update of the solution pool does not cause the pool to change or when a prescribed maximum number of iterations is reached. Since all com-

```
Input: first pool of solutions
Output: best solution found with Path Relinking
Pool ← read_Input();
while Pool does not remain the same do
    Candidates ← $\emptyset$;
    foreach new pair of solutions $(\underline{s}_1, \underline{s}_2)$ do
        $\underline{s}'$ ← Path Relinking($\underline{s}_1, \underline{s}_2$);
        Candidates ← Candidates ∪ {$\underline{s}'$};
    end
    try to update Pool with Candidates;
end
return $\underline{s}_{best}$ from Pool;
```

Algorithm 9: Overall Path Relinking framework.

putations of *Path Relinking* are independent from each other, the path relinking procedure on each solution pair in the same iteration can be completely parallelized. A sort of Tabu List feature can also be included to avoid the situation in which some solutions exiting from the *Pool* are immediately found and added again, even though unlikely.

5.2.4 Sampling version of the VMMP's heuristic

An issue with the VMMP's heuristic is that, the larger the instance, i.e., the higher is the number of servers, the higher will be the number of VMs selected by the pre-processing phase described in Chapter 4.6. Thus, the complexity increases much faster than for the VMPP. In particular, in a data center with 10000 servers it happens that, on average, one VM per server is selected for either mandatory or optional migration, leading to the migration of thousands of VMs. This unrealistic situation can be mitigated by filtering part of the input, as hinted in Chapter 4.6, but the complexity of the problem remains high. Therefore, the previously described VMMP's heuristic will perform a limited number of iterations and will not exploit the power of randomization and the potential multiple restarts of the GRASP approach.

To overcome, at least partially, this issue, we propose a modified version of the algorithm. The idea is to avoid to consider at each step all available alternatives, but rather to restrict the choice according to some sort of randomized criterion. This way, faster iterations should favor search diversification, making up for the less accurate exploration of the neighborhood. Since the restriction criterion is partially randomized, it will filter some solutions in some iterations and other solutions in other iteration, thus acting like a sampling technique from a wider pool of alternatives. In particular, we aim to speed up the following aspects of the GRASP framework:

1. greedy randomized construction, i.e., the generation of an initial feasible solution,
2. visit of a neighborhood,
3. convergence of an iteration in a local minimum and beginning of a new iteration.

1) The first point is tackled by modifying Algorithm 6. The new version is shown in Algorithm 10. The key difference lays in the computation, that is no more exhaustive, of the incremental costs related to each VM. We restrict the set of considered servers only to those situated in pods containing VMs of the same customer/application, following the same reasoning already applied in the local search. While in principle this choice could make the procedure miss the only feasible solutions of particular instances, in practice the instances that are target of this modified version should have pods so large that such probability is negligible. Also, we assume that sub-procedure$Pods(Cust)$ returns an updated set of pods containing VMs whose assignment has been currently determined. Therefore, in the unlikely, but possible, case of a customer that has all his VMs

```
   **Input:** Data Center, list of $\mathcal{C}_{ob}$, list of $\mathcal{C}_f$, $\alpha$
   **Output:** new allocation vector of VMs $\in \mathcal{C}_{ob} \cup \mathcal{C}_f$
   $Solution \leftarrow \emptyset$;
   while $\mathcal{C}_{ob} \neq \emptyset$ do
       $vm \leftarrow$ removeFirst($\mathcal{C}_{ob}$);
       $Cust \leftarrow getCustomer(vm)$;
       foreach $r \in Racks : r \in Pods(Cust)$ do
           $s \leftarrow sampleFrom(r)$;
           evaluate incrementalCost(vm,s);
       end
       RCL $\leftarrow$ buildRCL($\alpha$);
10     $\bar{s} \leftarrow$ randomly select a candidate from RCL;
       if canMigrateTo($vm,\bar{s}$) then
           update links and go on ;
       end
       else
           goto line 10;
       end
       $Solution \leftarrow Solution \cup (vm,\bar{s})$;
   end
   while $\mathcal{C}_f \neq \emptyset$ do
       $vm \leftarrow$ removeFirst($\mathcal{C}_f$);
       $Cust \leftarrow getCustomer(vm)$;
       foreach $r \in Racks : r \in Pods(Cust)$ do
           $s \leftarrow sampleFrom(r)$;
           evaluate incrementalCost(vm,s);
       end
       RCL $\leftarrow$ buildRCL($\alpha$);
27     $\bar{s} \leftarrow$ randomly select a candidate from RCL;
       if canMigrateTo($vm,\bar{s}$) then
           update links and go on ;
       end
       else
           goto line 27;
       end
       $Solution \leftarrow Solution \cup (vm,\bar{s})$;
   end
   return $Solution$;
```

Algorithm 10: Sampling version of the greedy randomized construction.

considered for migration, the first VM will face an empty set of pods. In such scenario, the basic version of the procedure is called and then the set of pods related to the customer is initialized according to the chosen destination of the VM.

Then, inside the restricted set of pods a further selection is performed. For each rack, only a random subset of servers is considered. The details depend on the definition of $sampleFrom(r)$, but an example would be to sample servers with a probability of

$$p = \frac{b}{Racksize}$$

where b could be equal to 1 or 2. This way, the number of incremental costs computed is, on average, in the order of $O(\frac{|Racks|}{|Pods|})$, instead of $O(|\mathcal{S}|)$. The idea behind it is that, if racks are composed by identical servers, identifying a good rack (energy-wise) by looking at any of its representatives might be enough in this phase of the procedure. Indeed, once a VM is assigned to a *good* rack, the local search can easily identify if in the same rack there is a more suitable server.

2) Rather than exhaustively considering every solution in the neighborhood of the current solution, i.e., every possible VM-server pair for the *One-Move-Neighborhood* and every VM-VM pair for the *One-Swap-Neighborhood*, we avoid some of them. We already partially do that with the use of the more specialized neighborhood variants, but it is not enough. We introduce a sort of Tabu List that forbids some moves for a prescribed number of local search iterations. The idea is that, if the local search establishes that it is profitable to change the assignment of a VM, then it will chose the best new assignment for it and such assignment will remain locally optimal for some iterations. Therefore, it can be beneficial not to consider all VM-server or VM-VM pairs involving that particular VM for the next t iterations, i.e., for the visit of the next t successive neighborhoods, where t is a parameter that can be tuned. For example, t could be equal to 10% of $|\mathcal{C}_{ob} \cup \mathcal{C}_f|$, namely the cardinality of the sets of migrating VMs. The Tabu List will be integrated with the criterion appearing in the following point.

Point 3) This point is tackled simultaneously in two ways. First, we slightly modify the basic general scheme of the local search, like shown in Algorithm 11. Instead of continuing the visit of the neighborhoods whenever at least one improving solution is found, we introduce a minimum incentive δ that we require to explore a new neighborhood. The procedure treats any solution that do not improve at least by δ the objective function value, with respect to the best neighbor found so far, as a non-improving solution. The advantage is that this

```
Input: starting solution $Solution$
Output: solution that is a locally optimal
while Solution is not locally optimal do
    Find $s' \in$ Neighborhood($Solution$) with $f(s') < (f(Solution) - \delta)$;
    $Solution \leftarrow s'$;
end
return $Solution$;
```

Algorithm 11: Modified Local_Search scheme.

technique allows for lower-bounded finite improvements of the objective function that are guaranteed to converge in a finite amount of time. On the other hand, the disadvantage is that it may happen that the procedure stops without considering a solution whose short-term improvement is lower than δ but that would lead to more significant improvements later on. Parameter δ can be tuned so that its value is large enough to filter a significant number of solutions while being low enough not to likely incur in the scenario described before and to have on average the only disadvantage of a precision's loss in the order of δ. This approach can be integrated with the one described in the previous point by adding to the Tabu List also moves (i.e., VMs) that are filtered according to the δ threshold, with the idea that, at least temporarily, it is unlikely that they will offer a better improvement.

Then, to avoid that the procedure follows a sequence of neighborhoods that is most likely to end up in a solution that is worse than the best one found so far, we can introduce the option of discarding a solution if at any moment it becomes clear that computing time is being wasted. In particular, we define two 'discard conditions':

- check that the solution found by the greedy randomized construction is not worse of a certain threshold γ_1, e.g., 80% of the value of the best among the initial solutions found,
- after the exploration of each neighborhood, check that the solution currently considered is not worse than a certain threshold γ_2, e.g., 70 % of the best solution found so far by the overall GRASP.

Whenever a solution does not meet any of such conditions, the current GRASP iteration is stopped, the solution is discarded and a new iteration begins. Note that such conditions also imply that different GRASPs working in parallel on the same instance will share information about the best solutions found, enforcing stronger thresholds.

5.2.5 Path selection sub-problem

In Chapter 4.7 we defined a part of the VMMP that completes the main problem. We defined it separately to be able to focus on other more interesting aspects of the problem. The idea is that the two should be solved in a fixed-point-iteration fashion, like shown in Algorithm 12.

```
$Solution \leftarrow \emptyset$ while not convergent do
    $Solution \leftarrow$ solve VMMP's main problem ;
    fix the values of the main problem's variables ;
    $Solution \leftarrow$ solve Path selection sub-problem ;
    fix the values of the sub-problem's variables ;
end
return $Solution$;
```

Algorithm 12: Fixed-point iteration scheme.

```
Input: Fixed traffics and allocations found the main procedure
Output: single path for each server pair
$Solution \leftarrow \emptyset$;
foreach server $s_1 \in \mathcal{S}$ do
    foreach server $s_2 \in \mathcal{S}$ do
        $p \leftarrow shortestPath(s_1, s_2)$ ;
        $Solution \leftarrow Solution \cup \{ p \}$ ;
        foreach link (i,j) $\in p$ do
            update ResidualCapacity$(i, j)$ with traffic $t_{s_1 s_2}$ ;
            update distance $d(i, j)$ as the reciprocal of the residual
              capacity ;
        end
    end
end
return $Solution$;
```

Algorithm 13: Path selection sub-problem heuristic.

In this thesis we do not design a sophisticated procedure to solve the Path selection sub-problem. We simply suggest a procedure based on repeated shortest path computations (see Algorithm 13). We use the Dijkstra's Shortest Path algorithm, therefore the inner loop on the set of servers can be omitted, since such algorithm can simultaneously compute the shortest path from a single node to each other node in the graph. However, especially for large instances, it is not necessary to specify all the $|\mathcal{S}|^2$ paths between server-pairs. Indeed, the only paths that need to be specified are the ones between servers with non-zero traffic $t_{s_1 s_2}$, thus making the inner loop much more computationally light.

CHAPTER 6

Testbed instances

Testbed instances play a crucial role in the process of validating our heuristics and properly analyzing the algorithms' behavior. Unfortunately, it is very difficult to obtain data directly from real data centers. Due to this practical limitation, we need to craft our own instances using our own parameters. However, an inaccurate parameters' selection would lead us to computational results that are not representative of real-world scenarios. Therefore, we aim to validate our models and algorithms on realistic instances. We generate them randomly, but using parameters from the literature and/or obtained by reports coming from real systems. In particular, we focus on gathering information about:

- common network topologies and capacities. In particular, how many hosts, nodes and links are generated when creating an instance, and how are they connected. We refer to what is usually done in the literature;
- real servers' configurations. How many cores, RAM and disk IOPS are available on real data-centers' machines and how much power they consume. We inspect benchmarks to have an idea about these data;
- VMs characteristics. We inspect real cloud vendors' catalogs to determine plausible VM requirements. A strong emphasis is put on CPU utilization to which an entire section is devoted;
- traffic requirements and distributions. Given the most common type of application, which traffic patterns are reasonable and which is the workload of a generic application. We mostly refer to the literature here.

Then, since we take data and information from different sources, a few adjustments need to be made in order to make all parameters coherent, such as: conversion of some VM measures otherwise not suitable for our formulation; generation of missing data omitted by some VM vendors on their catalogs; increase of values related to link capacities (w.r.t. literature) and hosts' network interfaces (w.r.t. benchmark used) that otherwise would result in a bottleneck when combined with the rest of the data.

In Section 6.1 we describe how the network topology of our instances is generated. In Sections 6.2, 6.2 and 6.4 we explain how servers' and VMs' parameters are obtained. Finally, in Section 6.5 we describe how the traffic matrices are generated.

6.1 Network

The most widely used network topologies in data-centers are the so called Fat-tree, VL2 and Bcube, as depicted in [12]. Their popularity derives from their natural affinity with the most common network architecture, the three-tier architecture, which fits well with the context of our analysis, i.e. three-tier web applications.

Fat-Tree topology when testing fat-tree topologies of different sizes we consider the work by Al-fares *et al.*[23] This kind of topology is a variation of the original fat-tree design but provides better scalability and it is often adopted in the literature. The idea is to build the network starting by the choice of the number of pods. A pod represents a group of racks and its size is not a-priori fixed. Instead, the number of racks within a pod as well as the number of servers within a rack depends on the global size of the network (i.e., on the number of pods itself). The same holds for the number of core and aggregation nodes/links. Given the choice of k pods, each pod is composed by k routing nodes divided in two layers of equal size. In the lower layer (edge nodes), each node is directly connected with $\frac{k}{2}$ hosts that represent a rack. Moreover, each node in the lower layer is directly connected with each node in the upper layer (aggregation nodes) of the same pod and viceversa. The communications between machines placed in different pods are routed through $(\frac{k}{2})^2$ core nodes. Each of these has one direct connection with every pod. Thus, given k pods, there are exactly $\frac{k^2}{4}$ core nodes, k^2 aggregation/edge nodes and $\frac{k^3}{4}$ servers.

A small example with four pods ($k = 4$) is depicted in Figure 6.1. Each pod contains two layers of two ($\frac{k}{2}$) switch nodes . Each node in the lower layer (edge layer) is connected directly to two ($\frac{k}{2}$) machines. The upper layer (aggregation layer) provides connectivity to each switch node in the core layer, making the distance between every pod relatively small independently of the size of the

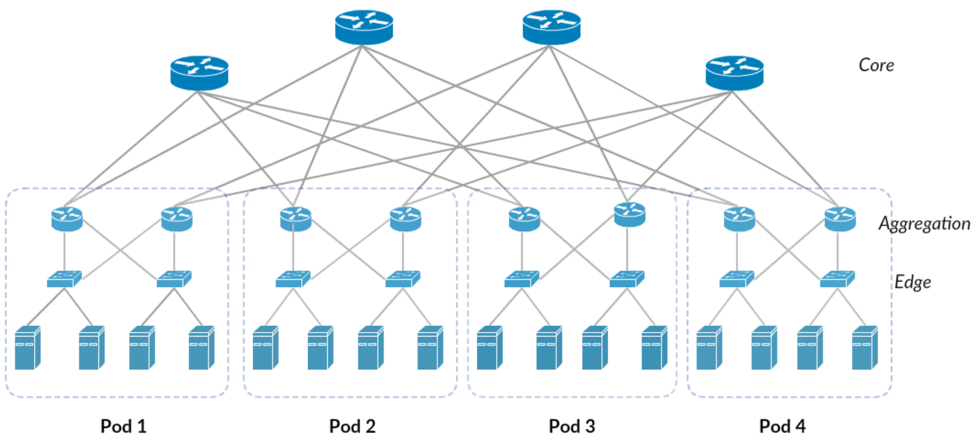


Figure 6.1: *Fat-tree topology with $k = 4$.*

data center. Since there are four core nodes ($\frac{k^2}{4}$), each aggregation node is connected to two ($\frac{k}{2}$) of them. The global number of servers is 16 ($\frac{k^3}{4}$), four ($\frac{k^2}{4}$) in each pod, two ($\frac{k}{2}$) in each rack.

Concerning the link capacities, 1 Gbit/s links were often considered in the literature for small/medium test instances until a few years ago. However, in a recent work [17], authors consider a private data center, named PRV2, that features link capacities ranging from 1 Gbit/s to 10 Gbit/s, even though for their tests they only use 1 Gbit/s links. Similarly, in [13], authors use 10 Gbit/s links for some links (i.e., access links). In [23], authors were already pointing out that, in a three-tier architecture, all links in the core layer as well as some in the lower layers needed to be 10 Gbit/s ethernet links, a technology that at the time was becoming cost-competitive. Moreover, all these works considered physical machines with much lower resources as well as VMs with much lower requirements (sometimes, too much, as in [16]) than today's standard (as will be shown in following sections). Since we aim to simulate as close as possible today's systems and use values from VMs provided by real vendors as of today, and on top of that we need to perform VMs migration, we think that the only realistic choice is to consider 10 Gbit/s links everywhere. Given that we need to host several VMs on each physical machine, and that some of them are web servers that will singularly require more that 1 Gbit/s bandwidth, it is just not possible to adopt lower capacities.

6.2 Servers

In our MILP formulations we model a server as a sort of knapsack with multiple dimensions. These represent four commonly considered resources, such as:

- number of CPU cores,
- quantity of RAM,
- disk IOPS,
- in/out bandwidth.

Other than that, the VMMP problem requires information about power consumption in order to define a cost coefficient in the objective function. As in the previous section, we find that in the literature the tested configurations involve machines too small. In [24], authors consider hosts with 4 GB of RAM, while in [13], Belabed *et al.* use 20 GB RAM machines. As of today, these values are close to the ones provided by most laptops or small servers, therefore are not compatible with our goals. Instead, for this type of data we look at some benchmarks. In particular, the benchmark SPECpower_ssj2008 [25] collects submissions about server resources and energy consumption from different vendors. Looking at submissions made in the third and fourth quarters of 2017, we select the ones listed in Table 6.1 .

Vendor	Model	P-max(W)	P-idle(W)	Cores	RAM(GB)
Inspur Corp.	NF5280	245	45.7	44	128
Sugon	I620-G30	419	67.2	56	192
Dell	R640	469	55	56	192
Hewlett Packard	ML350Gen10	459	58.1	56	192
Lenovo	SR650	183	45.6	28	96
Quanta Comp. Inc.	D52BQ-2U	426	39.5	28	192

Table 6.1: *SPECpower benchmark.*

We tried to select servers whose core/RAM ratio is compatible with the target of our analysis, i.e., mainly three-tier web applications that are not CPU intensive but rather memory intensive. Selecting servers with a lot of cores but few RAM would cause a quick memory saturation and a waste of CPU most of the times, causing one resource to be less relevant. Concerning the bandwidth, the submissions on the benchmark reported most of the times only one network interface of 1 Gbit/s connected, but often with the possibility to connect

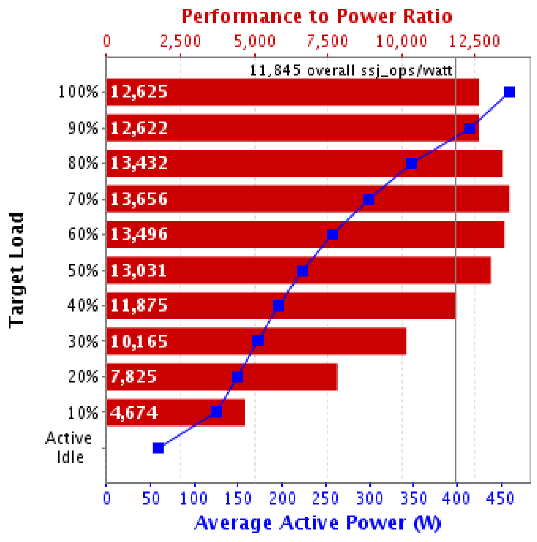


Figure 6.2: *Hewlett Packard ML350Gen10 power consumption.*

a 10 Gbit/s interface. We consider only 10 Gbit/s network interfaces so that we stay coherent with the network capacities. This consideration is supported by the data collected about Microsoft and Amazon virtual machines that often singularly require more than 1 Gbit/s , as shown in the following chapter.

Regarding the disk, there is no direct data about IOPS. The SSD's or HDD's model are the only information available and it can be difficult to find the exact data that we need. We estimate these data by looking at VMs requirements and at the average number of VMs that a server should be able to host. This way we make sure that the disk does not become a bottleneck for other resources and represents a meaningful part of the problem. Since the VMs' RAM (GB)/IOPS requirements range from $1.5 \cdot 10^{-2}$ to $4 \cdot 10^{-2}$, we establish that all servers can offer an amount of IOPS equal to their RAM (GB) multiplied by 200.

Last but not least, the power consumption of a server is modeled as if it was linear with its CPU utilization. This kind of approximation is common in the literature (e.g., see [26] and [27]) and, given the latest SPECpower's results, is reasonably accurate (see Figures 6.2 and 6.3). The slope is obtained by considering the difference between P-max (the power consumption under maximum load) and P-idle (the power consumption when the machine is not working). P-idle is also the offset from which the linear growth starts, as indicated in the problem formulation in Chapter 4.

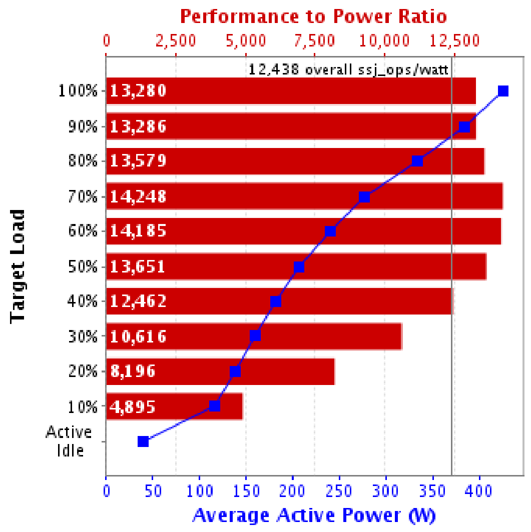


Figure 6.3: *LenovoSR650 power consumption.*

6.3 Containers and Virtual Machines

Virtual machines are widely treated in the literature, while containers are relatively new. Our formulation of the problem is not strictly related to any of the two technologies. It can be applied to both VMs and containers, the only difference being in the quantities involved, i.e., the same data center configuration can host a much higher number of containers compared to VMs. Let us recap the parameters that represent VMs/containers in our mathematical models:

- CPU utilization, normalized on each physical machine,
- RAM demand,
- disk IOPS demand,
- in/out bandwidth demand.

Unfortunately, containers' supply is currently limited, therefore our testbed instances are tuned with VMs data. As with the servers data, values used in [16] are too small (e.g., VMs using onlly a few Mbit/s of bandwidth). Wang *et al.* in [17], as well as authors of [28], use VMs whose memory ranges in the 1 GB - 10 GB interval, which we found to be compatible with today standards. However, they do not specify any other resource measure.

We follow the approach of Duong *et al.* [18], that use data about real VMs taken from Amazon EC2's catalog. However, they only use four different VMs, while we prefer to consider more and from at least two different vendors. For example, on the official website of Microsoft Azure there is a catalog of VMs grouped by categories, with prices and resources offered [2], but there is nothing similar about containers. Again, we look for VMs compatible with three-tier applications. In particular, the Microsoft Azure B-series is described as ideal for workloads that do not require full and intensive CPU performance, like web servers and small databases. Their configuration's details are reported in Table 6.2 .

Name	vCPU	RAM(GB)	Max disk IOPS	Max NICs
Standard_B1s	1	1	400	2
Standard_B1ms	1	2	800	2
Standard_B2s	2	4	1600	3
Standard_B2ms	2	8	2400	3
Standard_B4ms	4	16	3600	4
Standard_B8ms	8	32	4320	4

Table 6.2: *Microsoft Azure B-series [2].*

Since in our formulation we consider the bandwidth requirements expressed in Mbit/s or Gbit/s, we have to adapt the data on the last column that is expressed in number of network interfaces. We do this by looking at other series of virtual machines that report both measures. For example, looking at the Dsv2-series for a comparison, the Standard_b1s VM's bandwidth can be converted into 700 Mbit/s, while the Standard_B8ms VM's bandwidth can be translated into 3 Gbit/s. The others can take proportional values in between, e.g., 1.1 Gbit/s, 1.5 Gbit/s, 2.2 Gbit/s and 2.7 Gbit/s respectively. It is not by any means a precise conversion but it makes enough sense for the purpose of our analysis. Regarding the first column, we do not directly use the number of virtual CPUs as a resource requirement. Instead, we determine the CPU utilization of a VM on a specific server with the technique described in Chapter 6.4 .

Then, since the data about servers come from different vendors, we opt do the same here and consider what Amazon EC2 offers [3]. We select the VMs listed in Table 6.3 .

Name	vCPU	RAM(GB)	badwidth(Mbit/s)	IOPS
m4.large	2	8	450	2400
m4.xlarge	4	16	750	3600
m4.2xlarge	8	32	1000	5000

Table 6.3: *Amazon EC2 M4-series [3]. IOPS are generated by looking at the Azure's equivalent series.*

Instead of looking at the t2-series which is almost identical to the already selected Azure's series, we take a more resource-intensive series and discard the top-end instances because they alone would saturate most of our servers, causing infeasibility issues. Amazon's series provides bandwidth measures, but, unfortunately, does not report a disk IOPS measure. Again, we choose these values (see last column) within some ranges that resemble the Azure's ones, i.e., we have them scaled with similar vCPU/IOPS and/or RAM/IOPS ratios.

6.4 CPU utilization

Our formulation represents the CPU utilization of a VM in a more detailed way than simply using its number of virtual cores. In fact, in our model this parameter depends both on the VM itself (VM type and workload) and on the server hosting it. Therefore, we generate this parameter from other data. In particular, we consider the SPECvirt_sc2013 [29] benchmark that collects information about the impact of some predefined VM workloads on some machines. The workflows used are taken from the SPECweb benchmark [4] (used also in the next chapter) and meant to simulate the traffic workloads of a generic web application. The submitted results report a performance metric that measures the throughput of web servers and application servers when the hosting machine is fully loaded. To load a machine, the benchmark asks to progressively instantiate standardized tiles (sets) of VMs (e.g., a web server, an application server, a database every four tiles etc.). A machine is fully loaded when Quality of Service metrics start to drop under acceptable values. Using these data, we refer to Lazowska's Utilization Law [30] to derive Equation 6.4 and 6.5. In particular, let us denote with:

U_k**:** utilization of resource k,

X**:** throughput,

D_k**:** service demand of resource k,

B_k: busy time of resource k,

C_k: number of completion observed,

T: length of observation interval.

The utilization law states:

$$U_k = X \cdot D_k \tag{6.1}$$

and the utilization is also linked to the service demand through:

$$D_k = \frac{B_k}{C} = \frac{U_k \cdot T}{C} \tag{6.2}$$

In our case, resource k is the CPU of a single server s, and its utilization is the sum of contributes from each VM/container on that server:

$$U_s = \sum_{c \in \text{VMs}} U_{cs} \tag{6.3}$$

$$U_{cs} = X_c \cdot D_{cs} \tag{6.4}$$

Finally, the utilization is normalized on each server's number of cores to obtain the exact parameter used in the mathematical formulation:

$$CPU_{cs} = \#core_s \cdot U_{cs} \tag{6.5}$$

By a quick inspection of different submissions we noticed that, while there are variations among different machines' performance (ratio between machine's number of cores and number of tiles of VMs hosted), the ratio between web servers' and application servers' throughput is always almost identical, with very little variations, meaning that their impact on a host does not depend on the specific host. Therefore, since the VMs used as web servers and application servers in the benchmark have almost equal size, we decide that, once fixed a server s, the parameter D_{cs} can have only three distinct values: D_{web}, D_{app} and D_{db}. The relation between the first two is estimated in the following way:

$$D_{app} = D_{web} \cdot \frac{score_{web}}{score_{app}} =\sim 1.05 \cdot D_{web} \tag{6.6}$$

while, in the database case, we arbitrarily choose to use:

$$D_{db} = 2 \cdot D_{app} \tag{6.7}$$

Then, we roughly estimate D_{web} by adapting Equation 6.2. In particular, we do not have directly C_s, the number of completions in the time interval on machine s, and neither we have any time interval T to refer to. Instead, the benchmark offers a way to estimate $\overline{C}_s = \frac{C_s}{T}$. Knowing that:

- a certain number of tiles are hosted in the server,
- every tile is composed of a web server and an application server,
- a database is allocated every four tiles to serve four application servers,

we just sum up the average score of web servers, the average score of application servers and a quarter of the score of application servers and multiply this quantity by the number of tiles:

$$C_s = \#tiles \cdot (score_{web} + score_{app} + \frac{score_{app}}{4}) \tag{6.8}$$

Indeed, the scores of the benchmark are defined as the throughput at maximum load, i.e., the number of completions per unit of time at maximum load.

Moreover, Equation 6.2 requires to know U_k, the resource utilization in the configuration we are considering. We have an indication of it by looking either at the normalized per-tile score or at the total score. This metric is defined as the performance of the VMs against the theoretical maximum pre-computed for that system configuration, i.e., the Quality of Service metrics start to drop when the system is managing $\sim$ 93% of its maximum workload. Therefore we take $U_k = 0.93$.

Since the submissions on SPECvirt do not include the exact same machines that we selected from SPECpower benchmark in chapter 6.2, we need to adapt the data about C_s. We do this by using values of machines with a comparable number of CPU cores. In particular, we use:

- Hewlett Packard DL380Gen10 (56 cores) to estimate the value for Hewlett Packard ML350Gen10 (56 cores),
- Lenovo ThinkSystem SR650 (56 cores) to estimate the value for Lenovo ThinkSystem SR650 (28 cores) by simply taking half the value,
- a combination of Hewlett Packard DL380Gen10 (56 cores) and Lenovo ThinkSystem SR650 (56 cores) to estimate values for Sugon I620-G30 (56 cores) and Dell R640 (56 cores),
- Huawei Fusion RH2285H (36 cores) to estimate the value for Inspur Corp. NF5280 (44 cores) by scaling up the value of a factor $\frac{44}{36}$,
- Hangzhou H3C (16 cores) to estimate the value for Quanta Comp. D52BQ-2U (28 cores) by scaling up the value of a factor $\frac{28}{16}$.

Values are reported in Table 6.4 .

Vendor	Model	C_s
Inspur Corp.	NF5280	8030
Sugon	I620-G30	13700
Dell	R640	13800
Hewlett Packard	ML350Gen10	13655
Lenovo	SR650	6907
Quanta Comp. Inc.	D52BQ-2U	4567

Table 6.4: *Hosts' maximum throughput.*

The last element required to apply Equations 6.4 and 6.5 is the throughput of each VM. This quantity is generated following the rules described in Chapter 6.5 .

We are aware that there exist more accurate ways to estimate these parameters, however, given the data publicly available, this is enough for our analysis. We ran some tests loading our VMs with workloads (see next section) and CPU utilizations generated with these parameters and we found that, on average, the CPU happens to be comparable with the other resources, in particular the RAM resource, meaning that they all act as reasonable constraints with none being a bottleneck for the others. Also, our VMs with our workload achieve utilization factors similar to the ones of the benchmark. Finally, rather than merging together all independent data, these parameters give a meaningful correlation between the workload, and therefore bandwidth requirements, of a VM and its CPU utilization, which in turn is correlated to the power consumption of the hosting machine.

6.5 Traffic matrices

As pointed out by Belabed *et al.* in [13], using completely randomly generated traffic matrices may not be the best way to proceed. Instead, we should at least try to mimic real traffic as much as possible. First of all, the traffic matrices should be partitioned by customers since VMs are only aware of the existence of other virtual entities owned by the same customer, as done in [13]. This property was already took into account in our problem formulations and we exploited it to get more compact models.

We also need to consider meaningful traffic distribution and volumes. In particular, we look at the work by Greenberg *et al.* [31]. They analyze the network traffic in a particular type of data center network topology, the VL2 topology. Despite the focus on a specific topology and the year of publication, their results can be a good starting point. For example, they found that the vast majority of traffic flows in a data center are made of few bytes. These small traffics, called "mice flows", are clearly related to the countless ACK messages, meta-data requests and routing messages that have nothing to do with the actual traffic generated by the customers' applications. Their weight on the system is low and they should not impact the placement and migration decisions, therefore we decide to ignore them. Indeed, the remaining (fewer) flows account for the 90% of the whole traffic volume. These are the flows that we are going to consider. The authors found that these flows follow some common patterns: their size is usually uniformly distributed between 100 MB and 1 GB, a consequence of system design decisions (e.g 100 MB chunk size to amortize disk-seek times over read times), and a machine rarely communicates with more than a dozen of other hosts. Of course, these numbers might be outdated and need to be translated and adapted to our measures. Since a flow is meant as a traffic chunk between physical machines, this analysis tells us that the inter-VM communication has to take on average a fraction of that 100 MB-1 GB (i.e., 800 Mbit - 8 Gbit) range. Given that a VM mainly communicates with a small number of other VMs and that these will probably be put on a near, if not on the same, physical machine, the fact that on average a host has traffic towards a dozen of other machines is coherent with the demands/capacities ratio brought by our selection of servers and VMs (i.e., a server on average is able to host up to 10 VMs which generate traffic towards other VMs placed on a dozen of nearby machines). Moreover, if we roughly translate the 8 Gbit flow in 8 Gbit/s bandwidth usage, it almost saturates the bandwidth available on a typical host as well as the edge link capacities.

Furthermore, in [31], Greenberg *et al.* discover that the traffic matrices do not show particular regularities exploitable by some a priori traffic engineering. Instead, they are highly variable, cannot be trivially summarized in clusters/categories and are also quite unstable, meaning that, given the current traffic, it is almost impossible to predict how it will be in the near future. This lack of short-term predictability comes from the exploit of randomness to improve performance (e.g., data randomly spread on servers for redundancy and load distribution reasons). This consideration may suggest that some robustness and uncertainty measures should be taken into account. We do not directly include them in our formulation, leaving the topic open for future improvements. Our approach instead relies on periodical system reconfiguration through VMs migration and update of the routing paths. Given this consideration about short-term unpredictability, the average traffic requirements and patterns of well known types of applications as well as time and resource requirements for VMs migration can be reliably estimated.

Similar results are obtained by Benson *et al.* in [24]. They inspect different data centers used by different type of customers (e.g., Universities customers or Cloud customers). One aspect that they point out is that more than half the traffic generated inside a rack remains inside the rack, even though this phenomenon is stronger in cloud environments. This observation of course confirms that the objective function of our formulation makes sense and that trying to put VMs with high inter-traffic close to each other is what is actually done in real systems.

Finally, even if our problem formulations are quite general, the context of our analysis is meant to be mainly the common context of three-tiers web applications. Thanks to this consideration, we can feed our model and our algorithms with input data that resemble realistic virtual systems (owned by customers) composed by web servers, application servers and DBMS. This is done through an initialization phase that selects some VMs for each tier and assigns them a traffic coherent with their role. The assumption is that a customer will have components of the same tier hosted on equivalent VMs and the workload evenly shared among them.

To guarantee diversity in the VMs' choices, however, web servers, application servers and DBMS are instantiated on random VMs types among the ones presented in the previous chapters. In particular, for each customer, we select the average number of HTTP requests that their system will need to sustain. This number is randomly picked in a range that goes from 400 requests per second to 4000 requests per second. The interval is chosen by looking at some Wikipedia traces [32] from 2007-2008, following the approach of [33] and [34]. By manually inspecting some of these traces we found that the number of requests per second mainly ranges in the 1400-3000 range. Given that not every customer has workloads similar to Wikipedia, we allow a lower lower-bound and an higher upper-bound in order to have an order of magnitude in the spectrum of values. Unfortunately, we were not able to obtain other traces like the Yahoo's ones cited in the mentioned works.

Given a customer with workload incoming from the Internet, we select a number of identical VMs to act as web servers. Then, the bidirectional flow between web servers and the WAN is equally shared among the web servers. They need to receive all the traffic flow, so their number is decided based on their VM model's bandwidth requirements. Traffic between web servers and application servers is again evenly shared, and the same of course is applied to traffic between application servers and DBMSs. To be more precise, the in/out flow ratio in the web tier is not chosen randomly. Instead, we look at the benchmark

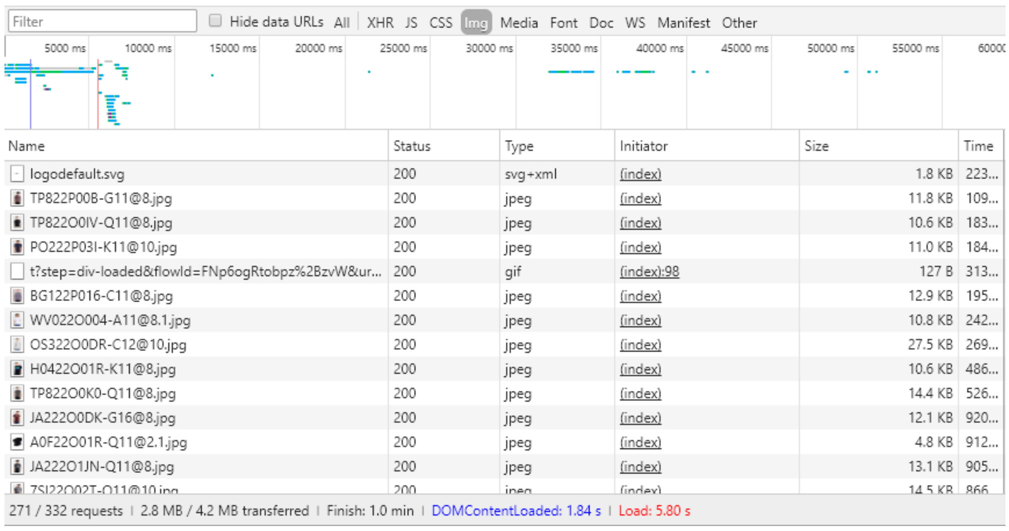
SPECweb2009 [4]. Unfortunately, the most recent entry is dated 2012. We find data collected about three categories of applications: Banking, E-commerce and Support. Depending on the category, requests to the web servers are different and characterized by different average sizes of the responses, as shown in Table 6.5 .The test is repeated three times for each category, as indicated by the iterations column.

Category	Average Bytes/Request	Iteration
Banking	30753	1
Banking	30753	2
Banking	30752	3
E-commerce	141842	1
E-commerce	141844	2
E-commerce	141834	3
Support	536247	1
Support	535318	2
Support	534711	3

Table 6.5: *SPECweb2009_PHP, 2012 first quarter [4].*

The average bytes/requests ratios are quite stable. Support requests are on average heavier due to download operations. Knowing that an HTTP request (HTTP GET or HTTP POST) from a client to a server usually can be estimated to be approximately 1 KB, we can estimate a reply/request ratio of 30 for banking applications, 141 for e-commerce applications and 535 for support applications. When generating virtual entities for a customer, we select randomly one of these categories and the incoming traffic, then compute the outgoing traffic accordingly. Then, knowing the traffic volume at the web tier, we can estimate the volumes at the application tier and database tier.

The output of the web-servers are data wrapped into HTML pages. Their size, due to HTML, CSS and Javascript code is larger than the related data sent by the application tier. Therefore we scale the application-level traffic down by an empirical factor compared to the web-level traffic. To estimate the range of this factor we inspect some web pages related to E-commerce and banking websites, both of which nowadays contain also aspects of the support category (mostly the download of images). We do this little inspection browsing different pages from the websites of Amazon [35], Mediaworld [36] and Zalando [37] for the E-commerce category, and from the website of Intesa SanPaolo [38] for the banking category. In all cases the Italian version of the websites is used. When downloading a page, we open the Google Chrome browser's console and look



Name	Status	Type	Initiator	Size	Time
logodefault.svg	200	svg+xml	(index)	1.8 KB	223...
TP822P00B-G11@8.jpg	200	jpeg	(index)	11.8 KB	109...
TP822O0IV-Q11@8.jpg	200	jpeg	(index)	10.6 KB	183...
PO222P03I-K11@10.jpg	200	jpeg	(index)	11.0 KB	184...
t?step=div-loaded&flowId=FNp6ogRtobpz%2BzvW&ur...	200	gif	(index):98	127 B	313...
BG122P016-C11@8.jpg	200	jpeg	(index)	12.9 KB	195...
WV022O004-A11@8.1.jpg	200	jpeg	(index)	10.8 KB	242...
OS322O0DR-C12@10.jpg	200	jpeg	(index)	27.5 KB	269...
H0422O01R-K11@8.jpg	200	jpeg	(index)	10.6 KB	486...
TP822O0K0-Q11@8.jpg	200	jpeg	(index)	14.4 KB	526...
JA222O0DK-G16@8.jpg	200	jpeg	(index)	12.1 KB	920...
A0F22O01R-Q11@2.1.jpg	200	jpeg	(index)	4.8 KB	912...
JA222O1JN-Q11@8.jpg	200	jpeg	(index)	13.1 KB	905...
7SI22O02T-O11@10.jpg	200	jpeg	(index)	14.5 KB	866

271 / 332 requests | 2.8 MB / 4.2 MB transferred | Finish: 1.0 min | DOMContentLoaded: 1.84 s | Load: 5.80 s

Figure 6.4: *Chrome's console shows the size of images with respect to the whole page.*

at the window regarding files and data transiting the network. These data are grouped into several categories, such as Images, HTML documents, CSS files, Javascript files and XHR requests.

As a demonstration, we use screenshots from one of Zalando's catalog pages. First of all, we look at the weight of images (see Figure 6.4) with respect to the whole page. Their weight ranges from 30% to 50% for the banking category and from 50% to 65% for the E-commerce category. Images are stored on the web servers' disks, so we subtract their contribution when considering the traffic between web tier and application tier. Then, we consider the weight of XHR requests with respect to HTML, CSS and Javascript files. XHR requests contain data fetched from the databases to fill the content of dynamic pages when the structure of the page has already been sent. This means that, if we focus our attention on pages with dynamic content (i.e., pages whose content depends on parameters and/or the user doing the request), we can compare the size of the fetched data with the size of the pages' static structure (HTML, CSS, Javascript files). This dynamic content roughly constitutes the data sent from application tier to web tier and whose ratio w.r.t. the whole page we are trying to quantify. The numbers that we found are shown in Table 6.6 . In particular, each page has been visited after a reset of the browser's cache. This way, even though the estimate might be too conservative (real users will have a certain amount of files saved locally in cache), the numbers are not influenced by a browser caching that would cause a less controllable behavior.

Website	Page type	data%
Amazon	Search by keyword	3%
Amazon	Product	1%
Amazon	Offers	2%
Amazon	My Orders	1%
Amazon	My Kindle	32%
MediaWorld	Catalog	7%
MediaWorld	Product	8%
Zalando	Catalog	15%
Zalando	Product	5%
Intesa SanPaolo	My Home	26%
Intesa SanPaolo	Archive	11%
Intesa SanPaolo	Rubric	15%
Intesa SanPaolo	Patrimony	10%
Intesa SanPaolo	My Cards	18%

Table 6.6: *Web pages inspection. The third column shows the percentage of data coming from the database tier with respect to the whole page.*

Apart from the 'My Kindle' page on the Amazon website, which is likely an outlier, we can approximately select a 1% - 15% range for the E-commerce category and a 10% - 25% range for the Banking category. For our instances we randomly pick a value within these two ranges.

Then, we need to do a similar process to estimate requests rate that arrives to the last tier, the database tier. When looking at the same web pages listed above, we manually estimate the size of the dynamic content in terms of common data representation on a computer, e.g., 4 bytes for an integer value, 8 bytes for a double precision value and a quantity that depends on the number of characters for strings, just to have a clue on the order of magnitude of the dimension of this type of content w.r.t to the whole web page.

As an example, a catalog page of the Amazon website display a list of items, each one containing some numerical and literal information on the item itself. We estimate roughly the amount of these data and we multiply them for the number of items in the displayed list. Since the number found are less precise than the previous ones and it is difficult to group them in categories, we simply establish that the traffic rates at this level are generated between the minimum value and the maximum value observed, i.e., between 5% and 50% with respect to the traffic rate at the application level.

To recap, when generating a customer (either new or already present if the data center), we apply this sequence of operations:

1. randomly choose if the customer's application belongs to E-commerce category or Banking category (Support category elements will be included in both),
2. sample a number of requests per second from a Uniform distribution ranging from 400 (minimum) to 4000 (maximum),
3. an HTTP request is roughly 1 KB, equal to 8 Kbit, so we multiply the previous number by 8 to obtain the incoming traffic expressed in Kbit/s,
4. depending on the category selected at step 1, select a percentage of images w.r.t to the web page size and use the in/out factor of the chosen category together with the in/out factor of images (Support Category) to determine the percentage of image requests w.r.t . all requests selected at step 2,
5. compute the response traffic by multiplying image requests and other requests by their respective in/out factor, call this number t0, and call t1 the traffic obtained by subtracting images traffic from t0, i.e., only data and HTML/CSS/JS traffic,
6. compute the application-to-web traffic by multiplying t1 by a random factor in the ranges indicated in the previous paragraph, call it t2,
7. compute the database-to-application traffic by multiplying t2 by a random factor in the ranges indicated in the previous paragraph, call it t3,
8. the traffic in the other direction follows the same rates chosen above for each tier and determines the throughput of every VM, used to estimate the CPU utilization in Section 6.4,
9. for each tier, randomly select a type of VM,
10. for each tier, determine the number of VMs needed dividing the overall related traffic (more precisely, the maximum between input and output related traffics) by the chosen VM's bandwidth requirements. Since we need an integer number, we round the result up,
11. generate the customer's traffic matrix by equally splitting the traffic at each tier among the tier's VMs,
12. convert all the numbers obtained into the preferred unit of measure, e.g., Gbit/s or Mbit/s.

Finally, when generating the set of new VMs, we randomly assign them to some customers. This means that a part of those customers' traffic matrix has to be updated accordingly.

CHAPTER 7

Computational results

In this chapter we report the computational results of the tests that we performed. For both the VMPP and the VMMP we:

- describe how the tests were set up, addressing the choice of particular parameters that were not discussed in previous chapters. We also specify how the initial configuration is obtained, i.e., how the data center of each particular instance is populated to simulate a generic plausible scenario,
- show the numerical results obtained. The goal is to analyze the quality of the solutions and the behavior of the two heuristics. Results are presented in tables, each followed by a brief analysis.

The general approach guiding all the tests is to compare our heuristics with either an exact mathematical solver or another heuristic procedure widely used in practice and/or in the literature.

In Section 7.1 we describe how the VMPP tests were prepared, while in Section 7.2 we report the related computational results. Similarly, in Section 7.3 we describe how the VMMP tests were set up and in Section 7.4 we report the related computational results.

7.1 VMPP experimental setup

Instances for the tests are generated in a pseudo-random way starting from a specified *seed*. In this way we can reproduce them and use them multiple times for comparisons. The Random Number Generator used is provided by the Java SecureRandom implementation [39]. In particular, for portability, the underlying algorithm that we use is the SHA1PRNG algorithm that is available on all Operating Systems. VMPP's instances are generated as follows:

1. specify the data center's size and topology, the number of customers' applications already allocated, the number of new VMs and the number of new applications to be allocated;

2. an empty data center is generated according to the structure described in Chapter 6 and both already existing and new applications are generated following the rules described at the end of Chapter 6. Then, other new VMs are randomly added to some applications so that the specified number of new VMs is reached. In particular, each of these has a 0.66 probability of being assigned to the same customer as the previous one. This criterion is enforced because otherwise, for large instances, the probability of generating more than one new VM for a customer would be close to zero, thus trivializing a portion of the problem;

3. all non-new VMs are placed in the data center, one application at a time. In particular, for each application a random rack is chosen. Racks are kept in descending residual RAM order and only the first half is considered when randomly picking a rack for an application. Having selected a rack, the procedure tries to allocate the application's VMs while visiting the rack's machines in lexicographical order. The VMs here are ordered following the pattern "web server - app server - dbms" so that it is likely to save a lot of bandwidth by putting them on the same server. Otherwise, a random scattered placement would quickly saturate the servers' bandwidth capacities. Also, while the initial configuration is assumed to be non-optimal, it would be unrealistic to have a completely inefficient configuration. During this allocations, capacities are constantly updated. If, for any reason, some VMs cannot be placed in the chosen rack, another one is selected and the same process is applied again. For the allocation of a single application we give a 10 seconds time-out, after which the all process is stopped and the current *seed* is discarded due to supposedly generating infeasible configurations.

4. during the placement described in the previous point, we specify a threshold for the resources utilization of any server. This can be tuned down to avoid resource saturation that would cause infeasibility when trying to solve the problem later, but, on the other hand, the lower the value, the more difficult it is for the initialization procedure to allocate VMs. For these tests we use a value of 0.9.

Computations were performed on different machines:

- the CPLEX solver (version v12.8.0.0) was run on a machine equipped with 8 CPU cores Intel Xeon E31245(3.30GHz) and 16 GB of RAM,
- heuristics were run either on a CHRP IBM 8246-L2T cluster equipped with 64 POWER7 CPU cores (4.2GHz) and 128 GB of RAM or on a VM from Microsoft Azure's [2] catalog, in particular the VM D8s Standard v3, equipped with 8 virtual CPU cores and 32 GB of RAM.

7.2 VMPP's computational results

In this section we are going to evaluate different aspects of the VMPP's heuristic, such as: optimality gap on instances up to 128 servers, comparison between runs of different length on large instances with 10000 servers, impact of the GRASP's randomization parameter, impact of the local search and comparison with a naive First Fit heuristic.

Optimality gap First of all, we are interested in knowing the quality of the solutions found by the algorithm described in Chapter 5. We compared the quality of the solutions obtained by such algorithm and the solutions found by solving the MILP formulation with the CPLEX solver. We ran CPLEX on the MILP formulation reported in the Appendix. The formulation was written in the AMPL language. We identified the set of the largest instances solvable by CPLEX in reasonable time, i.e., Fat-Tree data centers with up to $k = 8$ pods, corresponding to 128 servers, 40 new VMs to be placed, including three new applications, and 210 applications already allocated. Beyond this value of k, our machine would saturate the memory before being able to terminate a computation, a situation that already happens with some instances for $k = 8$. CPLEX was given a time limit of 3600 seconds, while the algorithm was given a time limit of 180 seconds. Due to memory concerns, we did run CPLEX using options '`nodefile=3, treemem=128, mipgap=0.00001`', The first two options allow to send part of the RAM content to the disk during the computation so that the memory saturation is less likely to happen. However, together with the enforced time limit, they may occasionally cause the solver to prematurely stop in an non-optimal solution. The objective function's value is expressed in MB/s. Results are summarized in Tables 7.1, 7.2 and 7.3. The relative gap is computed as

$$rel\_gap = \frac{heur\_value - best\_known}{best\_known},$$

where *best_known* is the solution found by CPLEX. The GRASP was run on the Azure VM.

Instance	best_known	grasp_value	rel_gap	abs_gap	cplex_time	grasp_iter
1	1525	1525	0	0	511s	27730
2	374,129	374,129	0	0	152s	33457
3	1073,238	1073,238	0	0	706s	20089
4	955,265	955,265	0	0	460s	29684
5	2436,869	2436,869	0	0	1234s	19739
6	2439,231	2439,231	0	0	334s	15280
7	1608,337	1608,337	0	0	328s	25421
8	2034,833	2031,087	-0,002	-3,746	810s	69208
9	1167,8	1167,8	0	0	343s	32986
10	1212,977	1212,977	0	0	179s	9213
11	1586,196	1586,196	0	0	2268s	28392
12	1988,447	1988,447	0	0	462s	21649
13	2691,288	2691,288	0	0	932s	16838
14	882,039	882,039	0	0	221s	27087
15	1234,859	1183,427	-0,042	-51,432	3604s	66666
16	854,095	842,987	-0,013	-11,108	677s	8203
17	548,815	477,804	-0,129	-71,011	836s	11906
18	1499,806	1499,806	0	0	1073s	20248
19	1428,515	1428,515	0	0	1116s	35508
20	2038,881	2038,881	0	0	655s	23373
21	182,994	182,994	0	0	3602s	14048

Table 7.1: *CPLEX one hour run vs GRASP three minutes run, Part 1.*

In all the considered cases the GRASP procedure is able to match the solution found by CPLEX and several times, when CPLEX fails to reach the optimum due to time limit or `treemem` limit, it manages to outperform it. Such results are obtained within just $\frac{1}{20}$ of the time limit given to CPLEX.

Large instances Having shown that, on average, the performance of the algorithm on reduced, but non-trivial, instances are promising, we proceeded to analyze the solutions obtanied on very large instances. Our target here are instances with approximately 10000 servers, corresponding to Fat-Tree topologies with $k = 34$ pods. Since we cannot have any proof of optimality for this

Instance	best_known	heur_value	rel_gap	abs_gap	cplex_time	heur_iter
22	1186,281	484,705	-0,591	-701,576	3603s	15398
23	170,536	155,803	-0,086	-14,733	3602s	25220
24	757,039	757,039	0	0	1237s	18884
25	928,787	928,787	0	0	719s	25623
26	2296,537	2296,537	0	0	1666s	33118
27	1554,956	1475,842	-0,051	-79,114	627s	14048
28	735,558	735,558	0	0	1532s	13269
29	376,157	365,973	-0,027	-10,184	1257s	27890
30	2311,468	2297,239	-0,006	-14,229	2155s	115711
31	1898,568	1902,127	0,002	3,558	1318s	8653
32	1140,626	1041,817	-0,087	-98,81	412s	15120
33	1456,954	1456,954	0	0	1900s	11054
34	2063,4	2063,4	0	0	317s	23493
35	1526,868	1526,868	0	0	2037s	23587
36	1460,027	1460,027	0	0	223s	17065
37	657,284	657,284	0	0	2622s	23713
38	638,768	602,915	-0,056	-35,854	1558s	12626
39	767,34	649,863	-0,153	-117,477	2677s	91311
40	1362,447	1362,447	0	0	479s	37203
41	1396,74	1391,083	-0,004	-5,658	1547s	12973
42	1225,041	1225,041	0	0	2155s	19199

Table 7.2: *CPLEX one hour run vs GRASP three minutes run, Part 2.*

Instance	best_known	heur_value	rel_gap	abs_gap	cplex_time [s]	heur_iter
Average Gaps			-0,03	-28,842		

Table 7.3: *CPLEX one hour run vs GRASP three minutes run, Part 3.*

type of instances, we compare the solutions obtained within a short time limit (three minutes) with the solutions obtained in a time interval of 60 minutes by the same procedure, keeping the same $\frac{1}{20}$ ratio of the comparison with CPLEX. This way, given that we have previously shown that the algorithm is able to obtain solutions with very good quality, we can moderately trust the quality of solutions coming from a long run. Our goal is to show that the algorithm approaches very good solutions already in the first minutes of execution, meaning that it is useful for obtaining good solutions also in an online setting.

Instances feature 10000 servers, 100 new VMs, including 7 new applications, and 10000 already placed applications. Both the 60 minutes run and the three minutes run were performed on the IBM POWER7 server. Results are

Instance	60m_value	3m_value	rel_diff	abs_diff	3m_iter
100	1858,147	1858,147	0	0	259
101	1300,375	1300,375	0	0	283
102	2635,806	2635,806	0	0	314
103	2059,404	2059,404	0	0	318
104	2381,551	2381,551	0	0	323
105	1616,105	1616,105	0	0	333
106	3651,693	3651,693	0	0	339
107	1951,525	1951,525	0	0	448
108	1521,956	1521,956	0	0	388
109	1426,905	1426,905	0	0	374
110	2111,27	2111,27	0	0	457
111	2996,871	2996,871	0	0	421
112	2129,931	2129,931	0	0	354
113	3025,222	3025,222	0	0	333
114	2327,82	2327,82	0	0	339
115	4199,377	4199,377	0	0	371
116	3496,824	3496,824	0	0	379
117	1947,74	1947,74	0	0	294
118	1848,531	1848,531	0	0	295
119	3177,857	3177,857	0	0	278
120	1934,648	1934,648	0	0	335
121	1421,112	1421,112	0	0	352
122	2155,911	2155,911	0	0	373
123	1277,823	1277,823	0	0	395
124	1025,74	1025,74	0	0	368
125	4642,782	4642,782	0	0	383
126	2656,147	2656,147	0	0	432
127	1791,76	1791,76	0	0	370
128	1828,314	1828,314	0	0	334
129	3014,654	3014,69	0	0,037	342
Average Gaps			0	0,001	

Table 7.4: *GRASP one hour run vs GRASP three minutes run.*

reported in Table 7.4. The relative differnce is computed as

$$rel\_diff = \frac{3min\_value \text{ - } 60min\_value}{60min\_value}.$$

Apparently, a long run does not bring significant benefits to the solutions found within the first minutes. Only one out of 30 instances shows a difference in the results, and such difference is not significant. Actually, the algorithm converges within a 1% difference in less than one minute.

Impact of the GRASP randomization factor Then, we would like to know how the randomization parameter α_grasp, used by the sub-procedure Greedy Randomized Construction, affects the quality of the results. Until now, tests have been run with $\alpha\_grasp = 0.15$. We ran again the GRASP on the same large instances previously described, but this time with $\alpha\_grasp = 0$, corresponding to a deterministic greedy algorithm. Apparently, even without the randomization factor, the algorithm is able to find results of the same quality, therefore we do not report them again. In particular, despite the variance brought by the parameter α_grasp, the local search manages to converge to solutions of the same quality in both cases. This result enforces the choice of a greedy deterministic procedure in [14].

Impact of the Local Search We would like to see the impact of the local search on the solutions' quality to understand if it is worth dedicating time to it. Taking again the previous set of computations (in particular, the short run), with $\alpha\_grasp = 0.15$, we compare the value of the best solution found with local search against the best initial solution found with only the greedy randomized phase of the procedure. The comparison is shown in Table 7.5.

It appears that, on average, approximately 15% of the final solution's value is brought by the Local Search, while the remaining 85% is due to an already good choice of the initial solution built in the greedy randomized construction's phase.

Comparison with First Fit heuristic As frequently done in the literature, e.g., in [14] , we compare the performance of our procedure against a naive heuristic that is often used in practice, the First Fit algorithm. Such procedure is not traffic-aware and simply assigns a VM to the first server that has enough CPU, RAM and DISK residual capacity to host it. New VMs were consideredd in customer order to implicitly help the First Fit. Results are shown in Table 7.6. Instances used are the large ones with 10000 servers. First Fit was run on the IBM POWER7 server.

Instance	Greedy_Rand	Local_Search	rel_diff	abs_diff
100	2094,687	1858,147	0,127	236,539
101	1816,938	1300,375	0,397	516,563
102	3024,598	2635,806	0,148	388,792
103	2562,859	2059,404	0,244	503,455
104	2746,234	2381,551	0,153	364,683
105	2385,422	1616,105	0,476	769,317
106	4004,909	3651,693	0,097	353,216
107	2104,391	1951,525	0,078	152,866
108	1758,783	1521,956	0,156	236,827
109	1775,634	1426,905	0,244	348,728
110	2290,712	2111,27	0,085	179,441
111	3180,598	2996,871	0,061	183,727
112	2498,397	2129,931	0,173	368,465
113	4255,089	3025,222	0,407	1229,867
114	2418,992	2327,82	0,039	91,172
115	4390,178	4199,377	0,045	190,801
116	3824,298	3496,824	0,094	327,473
117	2808,746	1947,74	0,442	861,005
118	2138,067	1848,531	0,157	289,536
119	3252,048	3177,857	0,023	74,191
120	1976,465	1934,648	0,022	41,817
121	1656,332	1421,112	0,166	235,219
122	2524,39	2155,911	0,171	368,479
123	1487,23	1277,823	0,164	209,408
124	1134,539	1025,74	0,106	108,8
125	4860,777	4642,782	0,047	217,994
126	2873,961	2656,147	0,082	217,814
127	1898,119	1791,76	0,059	106,358
128	1999,56	1828,314	0,094	171,246
129	3141,134	3014,69	0,042	126,444
Average Gaps			0,153	315,675

Table 7.5: *Greedy Rand Construction vs Local Search for the three minutes run.*

Instance	grasp_value	FF_value	rel_diff	abs_diff
100	1858,147	8778,074	3,724	6919,927
101	1300,375	4246,531	2,266	2946,156
102	2635,806	8084,619	2,067	5448,813
103	2059,404	11031,533	4,357	8972,129
104	2381,551	10122,152	3,25	7740,602
105	1616,105	8445,514	4,226	6829,409
106	3651,693	15054,529	3,123	11402,836
107	1951,525	13533,606	5,935	11582,082
108	1521,956	9687,143	5,365	8165,187
109	1426,905	9223,528	5,464	7796,623
110	2111,27	9003,186	3,264	6891,916
111	2996,871	18135,324	5,051	15138,453
112	2129,931	8336,305	2,914	6206,374
113	3025,222	14050,626	3,644	11025,403
114	2327,82	8428,128	2,621	6100,308
115	4199,377	17321,552	3,125	13122,175
116	3496,824	13557,479	2,877	10060,655
117	1947,74	8715,223	3,475	6767,483
118	1848,531	10524,831	4,694	8676,299
119	3177,857	8032,113	1,528	4854,257
120	1934,648	11399,646	4,892	9464,998
121	1421,112	7432,711	4,23	6011,599
122	2155,911	10359,538	3,805	8203,627
123	1277,823	5600,332	3,383	4322,51
124	1025,74	7283,067	6,1	6257,327
125	4642,782	12917,685	1,782	8274,903
126	2656,147	17383,513	5,545	14727,367
127	1791,76	7864,082	3,389	6072,321
128	1828,314	7852,113	3,295	6023,799
129	3014,69	14379,665	3,77	11365,012
Average Gaps			3,772	8245,685

Table 7.6: *GRASP three minutes run vs First Fit.*

As expected, the GRASP procedure performs better than a naive First Fit, with a relative difference ranging from 150-200% to 600%, with an average of approximately 377%. This means that, on average, our heuristic is able to reduce the traffic introduced in the data center network to almost $\frac{1}{5}$ of the traffic that would be introduced using First Fit. Such improvement becomes extremely significant considering the placement of hundreds of thousands of VMs over the lifetime of the data center. Since th objective function of the VMPP translates almost directly into links utilization, this result appears to be better than [14], where only a 9.9% improvement of links utilization is obtained.

7.3 VMMP experimental setup

As an extension of the VMPP, the initialization of a VMMP's instance is naturally built on the procedure described in Section 7.1. Additionally, after all the applications are given an allocation, there is a step where traffic paths and links' utilization are computed. According to the formulation in Chapter 4, the input of the problem should include the specification of a pre-defined traffic path between each servers' pair. However, for instances with more than a thousand servers, the number of required paths have caused memory issues. Luckily, only a subset of paths are actually needed to solve the problem since, for any server s, there will be only a few servers towards which s will send a non-zero amount of traffic and vice versa. Therefore, the number of explicitly required paths is in the order of $O(|\mathcal{S}|)$ rather than $O|\mathcal{S}|^2$, where $\mathcal{S}$ is the set of servers, thus making a huge difference memory-wise. Paths are selected according to a repeated invocation of JGrapht's [21] implementation of the Dijkstra's Shortest Path algorithm. In particular, the distance measure associated to each link $l = (i, j)$ is

$$distance(i, j) = \frac{1}{ResidualCapacity(l)}$$

For each pair of servers associated to a non-zero traffic, the shortest path procedure is called and all the capacities associated to links belonging to the selected path are updated accordingly to the traffic volume. In this way, an almost saturated link is not likely to be selected by successive invocations of the procedure. Again, as for the VMPP's initialization, if at any moment there are no feasible alternatives, the procedure is stopped and the *seed* is discarded.

Then, the problem requires to specify which VMs require a mandatory migration and which do not. The general procedure to select them is described in Chapter 4.6. However, such criterion does not scale well for large instances of the problem, causing several thousands of VMs to be selected. Therefore, when running tests on 10000 servers instances we apply a filtering factor of $\frac{1}{5}$ to the selection of VMs belonging to the set $\mathcal{C}_f$, i.e., each server has a $\frac{4}{5}$ chance of being neglected for the selection of VMs whose migration is not mandatory.

Another crucial aspect that needs to be addressed is related to the choice of migration parameters:

- parameter Q_c, which expresses the amount of traffic overhead required for the migration of a VM/container c, is roughly computed as $Q_c = \frac{RAM_c[Gbit]}{T_1[s]}$, where T_1 is the specified migration time expressed in seconds;
- migration time T_1 can be crucial for the existence of feasible solutions. On small instances the chance of such a problem is low and migration time is kept to 240 seconds, i.e. four minutes. For large instances, however, we use a less tight migration time of 600 seconds, i.e., 10 minutes,
- the objective function of the problem includes coefficients related to the power consumption, the traffic and an incentive/penalization factor for migration. The power consumption coefficient is fixed to 1 because we want the objective expressed in energy terms. The traffic coefficient used is $\alpha = 50$ (considering the traffic term expressed in Gbit/s), while the migration coefficient is sometimes used as $\beta = 1$ or $\beta = -1$, depending on the criterion given to help the procedure in breaking symmetries among the solutions. In particular, it is used as $\beta = -1$ for large instances.

The choice of $\alpha = 50$ has a rational. We did try some values on small/medium instances to understand how different values impact the type of solutions found. The values that we considered are:

- $\alpha = 0$, giving importance only on the servers' power consumption,
- $\alpha = 5$, giving little, but non-zero, importance to the traffic term,
- $\alpha = 50$, giving medium importance to the traffic. With this value, the power consumption's term and the traffic terms assume comparable values, i.e., they both provide a significant impact on the problem,
- $\alpha = 500$, giving little importance to servers' power consumption,
- $\alpha = +\infty$, equivalent to forcing the power consumption's coefficient to zero, giving all importance to the traffic, like in the VMPP.

The first two choices produce almost identical solutions, with several servers turned off and a significant energy saving. The last two choices produce very poor solutions with more servers turned on and increasing energy costs. Instead, a choice of $\alpha = 50$ causes almost as many servers to be turned off as the first two choices, while better distributing the traffic, therefore reducing the costs even more.

7.4 VMMP's computational results

In this section we evaluate different aspects of the VMMP's heuristic, such as: optimality gap on instances with up to 54 servers, comparison between the basic version and the modified version with sampling, comparison between runs of different length on large instances with 10000 servers, impact of the GRASP's randomization parameter and impact of the local search and Path Relinking.

Optimality gap As for the VMPP, we start by comparing the quality of the solutions found by our heuristic and the ones found by solving the MILP formulation with CPLEX on a set of instances. The GRASP's time limit was set at five minutes, while CPLEX's time limit was set at 60 minutes. Other CPLEX's options were set as described in Section 7.2. Since this problem is more challenging than the placement problem, instances correspond to 6-pods Fat-Tree data centers, i.e., systems with 54 servers. Results are shown in Table 7.7. The objective function's value is expressed in Watts. The number of migrating VMs ranges between 40 and 80. The GRASP was run on the IBM POWER7 server.

With this setting, as expected, the heuristic algorithm does not reach the optimal solution (or the best known) as easily as for the VMPP. However, the solutions found are still quite close to the optimum, with an average relative gap of approximately 11%. In particular, gaps vary between 3% and 35% for the considered instances. In one instance, the heuristic procedure is not able to find any feasible solution even though there exists at least one. Such particular case was not considered for the computation of the gaps.

Impact of the GRASP randomization factor As in the VMPP's case, we analyze how the value of the parameter α_grasp affects the quality of the solutions. In particular, we are interested in the comparison between $\alpha\_grasp = 0.15$ and $\alpha\_grasp = 0$, the latter corresponding to a deterministic greedy procedure.

For this problem, the two alternatives do influence the solutions' quality, with the deterministic version achieving an average relative gap of 16% against the 11% gap of the randomized version. Intuitively, being the problem more involved with respect to the VMPP and being the objective function a combination of different quantities, it is reasonable that a deterministic greedy approach does not perform as well.

Instance	best_known	grasp_value	rel_gap	abs_gap	cplex_time	grasp_iter
200	-1372,449	-1325,398	0,034	47,051	3603s	53654
201	-1123,262	-1083,786	0,035	39,476	1705s	18175
202	-798,771	-665,324	0,167	133,447	3602s	64317
203	-114,444	-74,264	0,351	40,18	117s	72443
204	-977,917	-682,906	0,302	295,011	3602s	43315
205	-778,907	-	-	-	2875s	36937
206	-861,826	-785,937	0,088	75,889	694s	66166
207	-1633,917	-1333,694	0,184	300,222	3602s	46902
208	-1045,72	-987,9	0,055	57,82	1038s	20175
209	-1033,846	-949,202	0,082	84,644	3603s	32850
210	-1938,221	-1751,068	0,097	187,153	3602s	33689
211	-1164,607	-1064	0,086	100,607	2107s	39554
212	-452,918	-364,794	0,195	88,124	345s	176011
213	-1582,989	-1435,95	0,093	147,039	3602s	34982
214	-1980,554	-1858,243	0,062	122,311	3602s	9342
215	-1957,39	-1728,093	0,117	229,297	3603s	37228
216	-885,067	-801,337	0,095	83,73	1366s	42907
217	-1386,773	-1167,803	0,158	218,97	3603s	55367
218	-1096,183	-997,469	0,09	98,714	3603s	51700
219	-540,411	-471,329	0,128	69,082	3604s	54171
220	-778,437	-695,484	0,107	82,953	998s	51458
221	-2011,68	-1851,399	0,08	160,28	3602s	36828
222	-1006,892	-955,191	0,051	51,702	3602s	56035
223	-378,421	-360,837	0,046	17,585	2311s	53462
224	-863,164	-790,264	0,084	72,9	3603s	77300
225	-1524,099	-1330,261	0,127	193,838	1053s	49792
226	-1926,007	-1858,954	0,035	67,053	1686s	35097
227	-1298,629	-1197,956	0,078	100,673	3603s	69097
228	-1528,78	-1467,336	0,04	61,444	3605s	56134
229	-1714,472	-1541,092	0,101	173,381	3602s	43838
230	-759,86	-710,426	0,065	49,434	3604s	41565
231	-738,488	-700,084	0,052	38,404	3604s	41632
232	-663,164	-616,609	0,07	46,555	3604s	87555
233	-479,03	-331,758	0,307	147,272	2891s	48413
Average Gaps			0,111	111,583		

Table 7.7: *CPLEX one hour run vs GRASP five minutes run.*

Path Relinking Table 7.7 refers to the outcomes of the VMMP's heuristic until the local search phase. If we also include the *Path Relinking* module, the solutions are subject to further improvement, as shown in Table 7.8. Path Relinking was used in its deterministic version. Only 25% of the path between two solutions was explored.

The relative gap is reduced from an average value of 11% to an average of 8%, ranging between 2% and 18% except for one unfortunate case where the gap is 31%. Such particular instance is the one for which the GRASP had the worst outcome (35% gap), therefore it is reasonable that the Path Relinking procedure, starting from a pool of poor quality solutions, was not able to obtain a solution close to the optimum.

Sampling We already discussed in Chapter 4 that the number of migrations, due to how we have defined the input of the problem, grows alongside the size of the data center so much that a single exhaustive iteration of the VMMP's heuristic becomes too computationally heavy for large instances. Therefore, for those instances, we designed a variant of the algorithm that we need to test. Such variant is described in Chapter 11. Table 7.9 shows the quality of the solutions found by such algorithm on the same instances previously considered. The heuristic was given again a five minutes time limit and was run on the Azure VM.

First of all, looking at the number of iterations performed (last column), it appears that this version of the algorithm has a throughput approximately 10 times higher than the basic version. It also appears that, while the quality of the solutions, with an average of 13%, is not far from the basic version, the algorithm is more inclined to fail when looking for an initial solution. Indeed, although a higher throughput allows it to consider a higher number of alternatives, sometimes beating the basic version, when there are fewer feasible options available it can miss them. However, this fact might be inflated by the small size of the instance causing "tighter" situations and by a non-optimal choice of internal parameters.

Large instances Finally, we move to instances with up to 10000 servers. As for the VMPP, we compare a run of the algorithm within its target time limit to a longer, more trustworthy, run. In particular, we did run the basic version

Instance	best_known	PR_value	rel_gap	abs_gap	cplex_time	PR_iter
200	-1372,449	-1349,015	0,017	23,434	3603s	4
201	-1123,262	-1096,988	0,023	26,274	1705s	3
202	-798,771	-685,985	0,141	112,786	3602s	6
203	-114,444	-78,593	0,313	35,85	117s	3
204	-977,917	-792,899	0,189	185,019	3602s	5
205	-778,907	-	-	-	2875s	0
206	-861,826	-792,097	0,081	69,729	694s	2
207	-1633,917	-1389,867	0,149	244,05	3602s	8
208	-1045,72	-989,218	0,054	56,502	1038s	2
209	-1033,846	-977,576	0,054	56,269	3603s	4
210	-1938,221	-1781,382	0,081	156,839	3602s	4
211	-1164,607	-1104,75	0,051	59,857	2107s	5
212	-452,918	-371,871	0,179	81,047	345s	4
213	-1582,989	-1491,759	0,058	91,23	3602s	9
214	-1980,554	-1888,812	0,046	91,742	3602s	3
215	-1957,39	-1786,289	0,087	171,101	3603s	5
216	-885,067	-807,593	0,088	77,474	1366s	2
217	-1386,773	-1200,449	0,134	186,324	3603s	4
218	-1096,183	-1033,872	0,057	62,312	3603s	4
219	-540,411	-496,346	0,082	44,064	3604s	4
220	-778,437	-709,042	0,089	69,394	998s	4
221	-2011,68	-1893,276	0,059	118,404	3602s	2
222	-1006,892	-981,781	0,025	25,112	3602s	5
223	-378,421	-360,837	0,046	17,585	2311s	1
224	-863,164	-808,252	0,064	54,912	3603s	3
225	-1524,099	-1360,008	0,108	164,091	1053s	3
226	-1926,007	-1882,641	0,023	43,366	1686s	4
227	-1298,629	-1224,85	0,057	73,779	3603s	3
228	-1528,78	-1491,275	0,025	37,505	3605s	4
229	-1714,472	-1578,654	0,079	135,819	3602s	5
230	-759,86	-732,306	0,036	27,554	3604s	5
231	-738,488	-706,874	0,043	31,614	3604s	3
232	-663,164	-623,589	0,06	39,576	3604s	4
233	-479,03	-450,076	0,06	28,953	2891s	5
Average Gaps			0,081	81,805		

Table 7.8: *CPLEX one hour run vs Path Relinking.*

Instance	best_known	grasp_value	rel_gap	abs_gap	cplex_time	grasp_iter
200	-1372,449	-1278,171	0,069	94,278	3603s	281962
201	-1123,262	-1060,964	0,055	62,298	1705s	182639
202	-798,771	-684,574	0,143	114,197	3602s	290507
203	-114,444	-39,67	0,653	74,774	117s	191525
204	-977,917	-849,675	0,131	128,243	3602s	211097
205	-778,907	-	-	-	2875s	145516
206	-861,826	-	-	-	694s	152919183
207	-1633,917	-1291,884	0,209	342,033	3602s	225648
208	-1045,72	-1021,793	0,023	23,927	1038s	244564
209	-1033,846	-	-	-	3603s	659702
210	-1938,221	-	-	-	3602s	3826122
211	-1164,607	-1049,856	0,099	114,75	2107s	254765
212	-452,918	-343,322	0,242	109,596	345s	500681
213	-1582,989	-1436,361	0,093	146,629	3602s	273143
214	-1980,554	-1904,114	0,039	76,44	3602s	280575
215	-1957,39	-1701,174	0,131	256,216	3603s	162253
216	-885,067	-790,691	0,107	94,375	1366s	206232
217	-1386,773	-	-	-	3603s	287157
218	-1096,183	-947,592	0,136	148,591	3603s	211968
219	-540,411	-467,266	0,135	73,145	3604s	322165
220	-778,437	-	-	-	998s	394144
221	-2011,68	-	-	-	3602s	317893
222	-1006,892	-930,026	0,076	76,866	3602s	339068
223	-378,421	-287,929	0,239	90,493	2311s	189896
224	-863,164	-811,004	0,06	52,16	3603s	277721
225	-1524,099	-	-	-	1053s	2842505
226	-1926,007	-1867,211	0,031	58,796	1686s	122529
227	-1298,629	-1212,661	0,066	85,968	3603s	272029
228	-1528,78	-1262,274	0,174	266,505	3605s	223897
229	-1714,472	-1458,854	0,149	255,618	3602s	117059
230	-759,86	-697,849	0,082	62,012	3604s	200625
231	-738,488	-680,281	0,079	58,207	3604s	212159
232	-663,164	-599,993	0,095	63,172	3604s	315243
233	-479,03	-426,648	0,109	52,382	2891s	302783
Average Gaps			0,132	114,68		

Table 7.9: *CPLEX one hour run vs GRASP five minutes run with sampling and no Path Relinking.*

Instance	180m_value	60m_value	rel_diff	abs_diff	60m_iter
300	-106321,196	-119266,331	-0,122	-12945,135	179
301	-104513,291	-116672,808	-0,116	-12159,517	163
302	-91115,437	-103617,068	-0,137	-12501,631	155
303	-122946,575	-138335,166	-0,125	-15388,592	167
304	-80926,369	-92247,604	-0,14	-11321,235	166
305	-121755,942	-135229,549	-0,111	-13473,607	160
306	-125174,559	-138411,834	-0,106	-13237,276	159
307	-106642,891	-121798,891	-0,142	-15156	158
308	-78157,902	-89165,115	-0,141	-11007,213	175
309	-104807,494	-118686,075	-0,132	-13878,581	152
310	-105096,18	-119100,553	-0,133	-14004,373	169
311	-115304,542	-129103,187	-0,12	-13798,645	172
312	-98689,667	-112268,532	-0,138	-13578,865	149
313	-121388,763	-134317,652	-0,107	-12928,889	172
314	-109383,437	-123560,194	-0,13	-14176,757	109
315	-101132,93	-114521,197	-0,132	-13388,267	105
316	-100353,116	-113222,041	-0,128	-12868,925	88
317	-87213,151	-97983,145	-0,123	-10769,994	113
Average Gaps			-0,127	-13143,528	

Table 7.10: *GRASP three hours run no sampling vs GRASP one hour run with sampling.*

of the VMMP's heuristic on a set of instances with a time limit of 180 minutes. Then, we ran the modified version of the algorithm (with sampling) on the same instances with a time limit of 60 minutes. These particular instances where derived from the large instances used for the VMPP. A filtering factor of 2 was applied to the set $\mathcal{C}_f$ to lighten the instance a bit. Results are collected in Table 7.10. The number of migrating VMs ranges from 6000 to 7000, while the total number of VMs is around 50000. The three hours run was performed on the IBM POWER7 server, while the 60 minutes run was performed on the Azure VM. The relative difference is computed as

$$rel\_diff = \frac{60m\_value \text{ - } 180m\_value}{180m\_value}.$$

Options chosen for the sampling, according to what discussed in Section 5.2.4 were:

- discard factor of 80% for the initial solution,

- discard factor of 65% for any solution found during the local search,
- $\delta = 50$ and Tabu List penalty of $0.1 \cdot |\mathcal{C}_{ob} \cup \mathcal{C}_f|$ for the Local Search's filtering and sampling,
- probability of $\frac{2}{Racksize}$ for the selection of a server in the greedy randomized construction step.

Unfortunately, despite our effort in improving the speed of the procedure, even the modified version of the procedure struggles in performing the desired number of iterations on such instances. A GRASP approach should aim at thousands of iteration to guarantee a satisfying exploration of the space of alternative solutions. Despite this consideration, the version of the procedure with sampling performs better than the regular version, even if the latter was given much more time. In particular, it manages to find better solutions with an average relative difference of approximately 13%.

Path Relinking for large instances Unfortunately, due to memory saturation, we were not able to perform Path Relinking on the solution found by the long run with the regular version of the procedure. However, we applied it to the solutions found by the modified version, which are anyway better. Solutions obtained with Path Relinking are compared to the solutions obtained by the GRASP with sampling, since they are the best known solutions. Path Relinking was run on the Azure VM. Results are shown in Table 7.11.

As expected, Path Relinking is able to provide a *further* average improvement of approximately 9%, carrying out approximately 10 iterations before meeting a stopping condition.

Instance	grasp_value	PR_value	rel_diff	abs_diff	PR_iter
300	-119266,331	-130321,952	-0,093	-11055,622	12
301	-116672,808	-126619,632	-0,085	-9946,823	11
302	-103617,068	-114437,625	-0,104	-10820,557	12
303	-138335,166	-148559,657	-0,074	-10224,49	9
304	-92247,604	-103311,609	-0,12	-11064,004	11
305	-135229,549	-144960,606	-0,072	-9731,058	11
306	-138411,834	-147982,984	-0,069	-9571,15	10
307	-121798,891	-131769,482	-0,082	-9970,591	9
308	-89165,115	-98833,862	-0,108	-9668,747	9
309	-118686,075	-128330,478	-0,081	-9644,403 4	11
310	-119100,553	-128507,647	-0,079	-9407,094	7
311	-129103,187	-140076,862	-0,085	-10973,675	11
312	-112268,532	-123308,945	-0,098	-11040,413	8
313	-134317,652	-144658,063	-0,077	-10340,411	11
314	-123560,194	-132285,968	-0,071	-8725,774	9
315	-114521,197	-125506,851	-0,096	-10985,654	11
316	-113222,041	-124587,562	-0,1	-11365,521	9
317	-97983,145	-108318,742	-0,105	-10335,597	8
Average Gaps			-0,089	-10270,644	

Table 7.11: *GRASP one hour run with sampling vs Path Relinking.*

CHAPTER 8

Concluding remarks

In this thesis we have studied the overall problem of optimal resource utilization in cloud data centers. In particular, the task consists in the optimal allocation of Virtual Machines and/or containers on physical servers. Since a data center is a dynamic system continuously evolving, the placement of a VM may need to be updated over time. We defined the variants of two different, but interconnected, problems: the VM Placement Problem (VMPP) and the VM Migration Problem (VMMP). The first problem deals with the online deployment of new VMs selecting their first optimal allocation at the time of entering in the system. The main concern here is the traffic awareness of the decision process. The second problem aims at periodically updating VM allocations that are no longer optimal due to the evolution of the system. In this context, traffic awareness is balanced with energy-awareness.

For each problem, we provided a MILP formulation that can be directly tackled by state of the art solvers in case of instances with up to 128 servers (VMPP) and 54 servers (VMMP). Then, we designed a heuristic framework to tackle large realistic instances of both problems. In particular, we devised two Greedy Randomized Adaptive Search (GRASP) algorithms composed by different modules that incrementally improve the quality of the solutions. The heuristic for the VMPP includes a greedy randomized procedure, that builds an initial feasible solution, and a local search procedure, that finds a nearby local minimum starting from such initial solution. The heuristic for the VMMP consists in a greedy randomized procedure, a local search procedure and a further procedure implementing the Path Relinking technique.

We tested the heuristics on randomly generated instances built with realistic data gathered from real vendors, benchmarks and renowned studies in the literature. The goal was to validate the quality of the solutions provided by the heuristics on instances for which the optimal solution was known thanks to the MILP formulation, and then to also test the algorithms on large instances. Computational results are promising, in particular for the VMPP's heuristic that achieves an optimality gap of approximately 0% from the optimal solution on instances with up to 128 servers. It is also able to find very good solutions in no more than one or two minutes on instances with up to 10000 servers, on which the procedure was shown not to produce better solutions even with 60 minutes runs. Instead, the VMMP's heuristic achieves an optimality gap of approximately 8% on instances with up to 54 servers and is able to tackle large instances with up to 10000 servers thanks to a computationally lighter refined version employing solution sampling on top of the basic procedure. There are no guarantees on the quality of the solutions of large instances of the problems, but all techniques and refinements that have been applied provide evident benefits.

Possible further improvements include improvements of the MILP formulations, algorithmic improvements and improvements regarding the considered instances. For example, the formulations could be modified not to impose strict constraints on the resources, but rather allow the possibility of violations together with a penalization factor. Instead, the algorithms could benefit from improvements aimed at increasing the number of iterations performed in the same computational time in order to have a stronger confidence in the results. Finally, even though the problems were defined with both VMs and containers in mind, the heuristics were tested only on VMs data, since we were not able to gather real data about containers. As for future work, the heuristics could be tested on instances generated with containers data and on network topologies different from the Fat-Tree topology, such as the VL2 and Bcube topologies.

CHAPTER 9

Appendix

9.1 Short-term VM Placement Problem's final formulation

$$
\begin{aligned}
min \quad & \sum_{s_1,s_2 \in \mathcal{S}_u} COST_{s_1 s_2} \cdot t_{s_1 s_2} \quad + \\
& \sum_{r \in \mathcal{R}} \sum_{s_1 \in \mathcal{S}_u, s_2 \in \mathcal{S}_r \setminus \mathcal{S}_u} COST_{s_1 s_2} \cdot \sum_{c_1 \in \mathcal{C}_r, c_2 \in \overline{\mathcal{C}}_r} d^r_{c_1 c_2} \cdot \overline{x}_{c_2 s_2} \cdot x_{c_1 s_1} \quad + \\
& \sum_{r \in \mathcal{R}} \sum_{s_1 \in \mathcal{S}_r \setminus \mathcal{S}_u, s_2 \in \mathcal{S}_u} COST_{s_1 s_2} \cdot \sum_{c_1 \in \overline{\mathcal{C}}_r, c_2 \in \mathcal{C}_r} d^r_{c_1 c_2} \cdot \overline{x}_{c_1 s_1} \cdot x_{c_2 s_2} \qquad (1.21)
\end{aligned}
$$

s.t.

$$\sum_{s \in \mathcal{S}_u} x_{cs} = 1 \quad \forall c \in \mathcal{C} \qquad (1.2)$$

$$\sum_{c \in \mathcal{C}} CPU_{cs} \cdot x_{cs} \leq \overline{CPU}_s \quad \forall s \in \mathcal{S}_u \qquad (1.3)$$

$$\sum_{c \in \mathcal{C}} MEM_c \cdot x_{cs} \leq \overline{MEM}_s \quad \forall s \in \mathcal{S}_u \qquad (1.4)$$

$$\sum_{c \in \mathcal{C}} DISK_c \cdot x_{cs} \leq \overline{DISK}_s \quad \forall s \in \mathcal{S}_u \qquad (1.5)$$

$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_1c_2}\cdot(x_{c_1s}-w^r_{c_1sc_2s})+\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\overline{\mathcal{C}}_r\cup\{c_0\}} d^r_{c_1c_2}\cdot(1-\overline{x}_{c_2s})\cdot x_{c_1s}+$$
$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\overline{\mathcal{C}}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_1c_2}\cdot(1-x_{c_2s})\cdot\overline{x}_{c_1s}\le\overline{BDW}^{out}_s \quad \forall s\in\mathcal{S}_u \qquad (1.15)$$

$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_2c_1}\cdot(x_{c_1s}-w^r_{c_1sc_2s})+\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r}\sum_{c_2\in\overline{\mathcal{C}}_r\cup\{c_0\}} d^r_{c_2c_1}\cdot(1-\overline{x}_{c_2s})\cdot x_{c_1s}+$$
$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\overline{\mathcal{C}}_r}\sum_{c_2\in\mathcal{C}_r} d^r_{c_2c_1}\cdot(1-x_{c_2s})\cdot\overline{x}_{c_1s}\le\overline{BDW}^{in}_s \quad \forall s\in\mathcal{S}_u \qquad (1.16)$$

$$w^r_{c_1s_1c_2s_2}\le x_{c_1s_1} \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r, \quad \forall s_1,s_2\in\mathcal{S}_u \qquad (1.17)$$

$$w^r_{c_1s_1c_2s_2}\le x_{c_2s_2} \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r, \quad \forall s_1,s_2\in\mathcal{S}_u \qquad (1.18)$$

$$w^r_{c_1s_1c_2s_2}\ge x_{c_1s_1}+x_{c_2s_2}-1 \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r, \quad \forall s_1,s_2\in\mathcal{S}_u \qquad (1.19)$$

$$t_{s_1s_2}=\sum_{r\in\mathcal{R}}\sum_{c_1,c_2\in\mathcal{C}_r} d^r_{c_1c_2}\cdot w^r_{c_1s_1c_2s_2} \quad +$$
$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r,c_2\in\overline{\mathcal{C}}_r} d^r_{c_1c_2}\cdot\overline{x}_{c_2s_2}\cdot x_{c_1s_1} \quad +$$
$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\overline{\mathcal{C}}_r,c_2\in\mathcal{C}_r} d^r_{c_1c_2}\cdot\overline{x}_{c_1s_1}\cdot x_{c_2s_2}$$
$$\forall s_1,s_2\in\mathcal{S}_u \qquad (1.20)$$

$$x_{cs}\in\{0,1\} \quad \forall c\in\mathcal{C}, \quad \forall s\in\mathcal{S}_u$$

$$w^r_{c_1s_1c_2s_2}\in\{0,1\} \quad \forall r\in\mathcal{R} \quad \forall c_1,c_2\in\mathcal{C}_r, \quad \forall s_1,s_2\in\mathcal{S}_u$$

$$t_{s_1s_2}\in\mathbb{R}^+ \quad \forall s_1,s_2\in\mathcal{S}_u$$

Table 9.1: VMPP *parameters and variables.*

Sets	
$\mathcal{R}$	customers of the new requests
$\mathcal{S}$	physical servers
$\mathcal{S}_u \in \mathcal{S}$	under-utilized physical servers (<60% cpu utilization)
$\mathcal{S}_r \in \mathcal{S}$	servers that host containers of client r
$\mathcal{C}$	new containers to be placed
$\mathcal{C}_r \in \mathcal{C}$	new containers owned by client r
$\overline{\mathcal{C}}$	already esisting containers
$\overline{\mathcal{C}}_r \in \overline{\mathcal{C}}$	already existing containers owned by client r
$\{c_0\}$	dummy container generating all external traffic
$\{s_0\}$	dummy server representing the WAN
Parameters	
$\overline{CPU}_s$	residual cpu available on server s
$\overline{MEM}_s$	residual RAM available on server s
$\overline{BDW}_s^{in}$	residual out-bandwith available on server s
$\overline{BDW}_s^{out}$	residual in-bandwith available on server s
$\overline{DISK}_s$	residual disk IOPS available on server s
CPU_{cs}	cpu demand for container c on server s, see problem 2 for details
MEM_c	ram demand for container c
$DISK_c$	i/o per second disk demand for container c
$COST_{s_1 s_2}$	pre-computed cost(hops) from server s_1 to server s_2
$d^r_{c_1 c_2}$	traffic demand matrix for client r
$\overline{x}_{cs}$	= 1 if container c is already placed in server s
Variables	
x_{cs}	= 1 if container c is placed in server s
$w^r_{c_1 s_1 c_2 s_2}$	= 1 if container c_1 is placed on server s_1 *and* c_2 is placed on s_2, both owned by client r
$t_{s_1 s_2}$	new traffic between server s_1 and server s_2

9.2 Long-term VM Migration Problem's final formulation

$$\mathcal{C}_m \equiv \mathcal{C}_{ob} \cup \mathcal{C}_f$$
$$P_s \equiv P_s^{max} - P_s^{idle}$$

$$\begin{aligned} min \quad & \sum_{s \in \mathcal{S}} P_s \cdot \Delta u_s + P_s^{idle} \cdot (b_s^{new} - b_s^{old}) \\ & + \alpha \cdot \sum_{s_1 \in \mathcal{S}, s_2 \in \mathcal{S}} COST_{s_1 s_2} \cdot \Delta t_{s_1 s_2}^{new} \\ & + \beta \cdot \sum_{c \in \mathcal{C}_f} \sum_{s \in \mathcal{S}} x_{cs}^{old} \cdot x_{cs}^{new} \end{aligned} \tag{2.42}$$

s.t.

$$\sum_{s \in \mathcal{S}} x_{cs}^{new} = 1 \quad \forall c \in \mathcal{C}_m \tag{2.2}$$

$$x_{cs}^{new} \le b_s^{new} \quad \forall c \in \mathcal{C}_m, \quad \forall s \in \mathcal{S} \tag{2.3}$$

$$x_{cs}^{old} \le b_s^{new} \quad \forall c \notin \mathcal{C}_m, \quad \forall s \in \mathcal{S} \tag{2.4}$$

$$\sum_{c \in \mathcal{C}_m} CPU_{cs} \cdot (x_{cs}^{new} - x_{cs}^{old}) \le \overline{CPU}_s - (1 - \rho_3) \cdot CPU_s \quad \forall s \in \mathcal{S} \tag{2.28}$$

$$\sum_{c \in \mathcal{C}_m} MEM_c \cdot (x_{cs}^{new} - x_{cs}^{old}) \le \overline{MEM}_s \quad \forall s \in \mathcal{S} \tag{2.29}$$

$$\sum_{c \in \mathcal{C}_m} DISK_c \cdot (x_{cs}^{new} - x_{cs}^{old}) \le \overline{DISK}_s \quad \forall s \in \mathcal{S} \tag{2.30}$$

$$\begin{aligned} & \sum_{r \in \mathcal{R}} \sum_{c_1 \in \mathcal{C}_m \cap \mathcal{C}_r} \sum_{c_2 \in \mathcal{C}_m \cap \mathcal{C}_r} d_{c_1 c_2}^r \cdot (x_{c_1 s}^{new} - x_{c_1 s}^{old} + (x_{c_1 s}^{old} \cdot x_{c_2 s}^{old}) - w_{c_1 s c_2 s}^r) \quad + \\ & \sum_{r \in \mathcal{R}} \sum_{c_1 \in \mathcal{C}_m \cap \mathcal{C}_r} \sum_{c_2 \in (\mathcal{C}_r \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2}^r \cdot (1 - x_{c_2 s}^{old}) \cdot (x_{c_1 s}^{new} - x_{c_1 s}^{old}) \quad + \\ & \sum_{r \in \mathcal{R}} \sum_{c_1 \in \mathcal{C}_r \setminus \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m \cap \mathcal{C}_r} d_{c_1 c_2}^r \cdot x_{c_1 s}^{old} \cdot (x_{c_2 s}^{old} - x_{c_2 s}^{new}) \\ & \le \overline{BDW}_s^{out} \quad \forall s \in \mathcal{S} \end{aligned} \tag{2.31}$$

$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_m\cap\mathcal{C}_r}\sum_{c_2\in\mathcal{C}_m\cap\mathcal{C}_r} d^r_{c_2c_1}\cdot(x^{new}_{c_1s}-x^{old}_{c_1s}+(x^{old}_{c_1s}\cdot x^{old}_{c_2s})-w^r_{c_1sc_2s}) \quad +$$
$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_m\cap\mathcal{C}_r}\sum_{c_2\in(\mathcal{C}_r\setminus\mathcal{C}_m)\cup\{c_0\}} d^r_{c_2c_1}\cdot(1-x^{old}_{c_2s})\cdot(x^{new}_{c_1s}-x^{old}_{c_1s}) \quad +$$
$$\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r\setminus\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m\cap\mathcal{C}_r} d^r_{c_2c_1}\cdot x^{old}_{c_1s}\cdot(x^{old}_{c_2s}-x^{new}_{c_2s})$$
$$\le \overline{BDW}^{in}_s \quad \forall s\in\mathcal{S} \tag{2.32}$$

$$\sum_{s\in\mathcal{S}} x^{old}_{cs}\cdot x^{new}_{cs}=0 \quad \forall c\in\mathcal{C}_{ob} \tag{2.33}$$

$$\sum_{j\in\delta^+(s)} f^c_{sj}=\frac{Q_c}{T_1}\cdot x^{old}_{cs}\cdot(1-x^{new}_{cs}) \quad \forall c\in\mathcal{C}_m,\quad \forall s\in\mathcal{S} \tag{2.11}$$

$$\sum_{i\in\delta^-(s)} f^c_{is}=\frac{Q_c}{T_1}\cdot(1-x^{old}_{cs})\cdot x^{new}_{cs} \quad \forall c\in\mathcal{C}_m,\quad \forall s\in\mathcal{S} \tag{2.12}$$

$$\sum_{i\in\delta^-(n)} f^c_{in}-\sum_{j\in\delta^+(n)} f^c_{nj}=0 \quad \forall n\in\mathcal{N}\setminus\mathcal{S},\quad \forall c\in\mathcal{C}_m \tag{2.13}$$

$$\sum_{c\in\mathcal{C}_m} f^c_{ij}-$$
$$\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}}\sum_{r\in\mathcal{R}}\sum_{c_1,c_2\in\mathcal{C}_r\cap\mathcal{C}_m} x^{old}_{c_1s_1}\cdot x^{old}_{c_2s_2}\cdot d^r_{c_1c_2}\cdot\pi^r_{c_1c_2} \quad -$$
$$\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}}\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r\setminus\mathcal{C}_m,c_2\in\mathcal{C}_r\cap\mathcal{C}_m} x^{old}_{c_1s_1}\cdot x^{old}_{c_2s_2}\cdot d^r_{c_1c_2}(1-\sum_{s\in\mathcal{S}} x^{old}_{c_2s}\cdot x^{new}_{c_2s}) \quad -$$
$$\sum_{s_1,s_2\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_1s_2}}\sum_{r\in\mathcal{R}}\sum_{c_1\in\mathcal{C}_r\cap\mathcal{C}_m,c_2\in\mathcal{C}_r\setminus\mathcal{C}_m} x^{old}_{c_1s_1}\cdot x^{old}_{c_2s_2}\cdot d^r_{c_1c_2}(1-\sum_{s\in\mathcal{S}} x^{old}_{c_1s}\cdot x^{new}_{c_1s}) \quad -$$
$$\sum_{s\in\mathcal{S}:(i,j)\in\mathcal{P}_{ss_0}}\sum_{r\in\mathcal{R}}\sum_{c\in\mathcal{C}_r\cap\mathcal{C}_m} x^{old}_{cs}\cdot d^r_{cc_0}\cdot(1-\sum_{s\in\mathcal{S}} x^{old}_{cs}\cdot x^{new}_{cs}) \quad -$$
$$\sum_{s\in\mathcal{S}:(i,j)\in\mathcal{P}_{s_0s}}\sum_{r\in\mathcal{R}}\sum_{c\in\mathcal{C}_r\cap\mathcal{C}_m} x^{old}_{cs}\cdot d^r_{c_0c}\cdot(1-\sum_{s\in\mathcal{S}} x^{old}_{cs}\cdot x^{new}_{cs})$$
$$\le \overline{K}_{ij} \quad \forall (i,j)\in\mathcal{E}$$

$$w^r_{c_1s_1c_2s_2} \leq x^{new}_{c_1s_1} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.36}$$

$$w^r_{c_1s_1c_2s_2} \leq x^{new}_{c_2s_2} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.37}$$

$$w^r_{c_1s_1c_2s_2} \geq x^{new}_{c_1s_1} + x^{new}_{c_2s_2} - 1 \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m, \quad \forall s_1, s_2 \in \mathcal{S} \tag{2.38}$$

$$\pi^r_{c_1c_2} \geq 1 - \sum_{s \in \mathcal{S}} x^{old}_{c_1s} \cdot x^{new}_{c_1s} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m \tag{2.39}$$

$$\pi^r_{c_1c_2} \geq 1 - \sum_{s \in \mathcal{S}} x^{old}_{c_2s} \cdot x^{new}_{c_2s} \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m \tag{2.40}$$

$$\pi^r_{c_1c_2} \leq 2 - \sum_{s \in \mathcal{S}} (x^{old}_{c_1s} \cdot x^{new}_{c_1s} + x^{old}_{c_2s} \cdot x^{new}_{c_2s}) \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m \tag{2.41}$$

$$\Delta u_s = \frac{\sum_{c \in \mathcal{C}_{ob} \cup \mathcal{C}_f} CPU_{cs} \cdot (x^{new}_{cs} - x^{old}_{cs})}{CPU_s} \quad \forall s \in \mathcal{S} \tag{2.43}$$

$$\begin{aligned} \Delta t^{new}_{s_1s_2} = & \\ & \sum_{r \in \mathcal{R}} \sum_{c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m} d^r_{c_1c_2} \cdot (w^r_{c_1s_1c_2s_2} - x^{old}_{c_1s_1} \cdot x^{old}_{c_2s_2}) \quad + \\ & \sum_{r \in \mathcal{R}} \sum_{c_1 \in \mathcal{C}_r \cap \mathcal{C}_m, c_2 \in \mathcal{C}_r \setminus \mathcal{C}_m} d^r_{c_1c_2} \cdot x^{old}_{c_2s_2} \cdot (x^{new}_{c_1s_1} - x^{old}_{c_1s_1}) \quad + \\ & \sum_{r \in \mathcal{R}} \sum_{c_1 \in \mathcal{C}_r \setminus \mathcal{C}_m, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m} d^r_{c_1c_2} \cdot x^{old}_{c_1s_1} \cdot (x^{new}_{c_2s_2} - x^{old}_{c_2s_2}) \qquad \forall s_1, s_2 \in \mathcal{S} \end{aligned} \tag{2.35}$$

$$x^{new}_{cs}, b^{new}_s \in \{0, 1\} \qquad \forall c \in \mathcal{C}_m, \quad \forall s \in \mathcal{S}$$

$$f^c_{ij} \in \mathbb{R}^+ \qquad \forall (i, j) \in \mathcal{E}, \quad \forall c \in \mathcal{C}_m$$

$$w^r_{c_1s_1c_2s_2} \in \{0, 1\} \quad \forall r \in \mathcal{R}, \quad \forall c_1, c_2 \in \mathcal{C}_m \cap \mathcal{C}_r, \quad \forall s_1, s_2 \in \mathcal{S}$$

$$\pi^r_{c_1c_2} \in \{0, 1\} \quad \forall r \in \mathcal{R}, \quad \forall c_1, c_2 \in \mathcal{C}_m \cap \mathcal{C}_r$$

$$\Delta t_{s_1 s_2} \in \mathbb{R} \quad \forall s_1, s_2 \in \mathcal{S}$$

$$\Delta u_s \in \mathbb{R} \quad \forall s \in \mathcal{S}$$

9.3 Notes

Regarding the full transformation from Constraints (2.22) to Constraints (2.31):

$$\begin{aligned}\overline{BDW}_s^{out} = BDW_s^{out} - &\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} - \\ &\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} - \\ \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} - &\sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old}\end{aligned}$$

is the residual bandwidth that, reshaped to explicit the total bandwidth, becomes

$$\begin{aligned}BDW_s^{out} = \overline{BDW}_s^{out} + &\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} + \\ &\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} + \\ \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old} + &\sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1 c_2} \cdot (1 - x_{c_2 s}^{old}) \cdot x_{c_1 s}^{old},\end{aligned}$$

which, substituted into Constraints (2.22), gives

Table 9.2: VMMP *parameters.*

Sets	
$\mathcal{R}$	customers
$\mathcal{N}$	nodes of the network (i.e. switch and servers)
$\mathcal{E}$	arcs of the network (i.e.physical links)
$\mathcal{S}$	physical servers (~ 10k)
$\mathcal{C}$	containers (~ 100k)
$\mathcal{C}_r$	containers owned by client r
$\mathcal{C}_{ob}$	containers that must be migrated
$\mathcal{C}_f$	containers whose migration is not mandatory
$\mathcal{P}_{s_1 s_2}$	set of arcs that constitute the precomputed path between server s_1 and server s_2
$\{c_0\}$	dummy container generating all external traffic
$\{s_0\}$	dummy server representing the WAN
Parameters	
CPU_s	number of cpu cores of server s
$\overline{CPU}_s$	residual cpu available on server s
MEM_s	RAM capacity of server s
$\overline{MEM}_s$	residual RAM on server s
BDW_s^{in}	in-bandwidth capacity of server s
BDW_s^{out}	out-bandwidth capacity of server s
$\overline{BDW}_s^{in}$	residual in-bandwidth on server s
$\overline{BDW}_s^{out}$	residual out-bandwidth on server s
$DISK_s$	disk IOPS capacity of server s
$\overline{DISK}_s$	residual IOPS available on server s
CPU_{cs}	cpu utilization demand for container c on server s
MEM_c	RAM demand for container c
$DISK_c$	disk IOPS demand for container c
$d^r_{c_1,c_2}$	traffic demand matrix of customer r
$\overline{x}^{old}_{cs}$	$= 1$ if container c placed on server s pre-migration
T_1	average migration time
T_2	max migration time
Q_c	size of container c's state
$P_s^{max} - P_s^{idle}$	constant related to server s power consumption
K_{ij}	capacity of physical link (i,j)
$\overline{K}_{ij}$	residual capacity of physical link (i,j)
$COST_{s_1 s_2}$	cost(hops) from server s_1 to server s_2 in the new configuration,
$t^{old}_{s_1 s_2}$	traffic demand from server s_1 to server s_2 in the old configuration
α, β	energy coefficients
ρ_1, ρ_2, ρ_3	cpu utilization thresholds

Table 9.3: VMMP *variables.*

Variables	
x^{new}_{cs}	$= 1$ if container c placed on server s post-migration,
b^{new}_{s}	$= 1$ if server s is ON post-migration
f^{c}_{ij}	traffic burst related to migration of container c on link (i,j)
$\pi^{r}_{c_1c_2}$	$= 1$ if c_1 *or* c_2, owned by r, is involved in the migration
$w^{r}_{c_1s_1c_2s_2}$	$= 1$ if container c_1 is placed on server s_1 *and* c_2 is placed on s_2, both owned by client r
Δu_s	cpu utilization's variation on server s after migration
$\Delta t^{new}_{s_1s_2}$	traffic variation between every couple of servers after migration

$$\sum_{c_1\in\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m} d_{c_1c_2}\cdot(x^{new}_{c_1s}-w_{c_1sc_2s})+\sum_{c_1\in\mathcal{C}_m}\sum_{c_2\in(\mathcal{C}\setminus\mathcal{C}_m)\cup\{c_0\}} d_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot x^{new}_{c_1s}+$$
$$\sum_{c_1\notin\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m} d_{c_1c_2}\cdot(1-x^{new}_{c_2s})\cdot x^{old}_{c_1s}+\sum_{c_1\notin\mathcal{C}_m}\sum_{c_2\in(\mathcal{C}\setminus\mathcal{C}_m)\cup\{c_0\}} d_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot x^{old}_{c_1s}$$
$$\leq$$
$$\overline{BDW}^{out}_{s}+\sum_{c_1\in\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m} d_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot x^{old}_{c_1s}+$$
$$\sum_{c_1\in\mathcal{C}_m}\sum_{c_2\in(\mathcal{C}\setminus\mathcal{C}_m)\cup\{c_0\}} d_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot x^{old}_{c_1s}+\sum_{c_1\notin\mathcal{C}_m}\sum_{c_2\in\mathcal{C}_m} d_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot x^{old}_{c_1s}+$$
$$\sum_{c_1\notin\mathcal{C}_m}\sum_{c_2\in(\mathcal{C}\setminus\mathcal{C}_m)\cup\{c_0\}} d_{c_1c_2}\cdot(1-x^{old}_{c_2s})\cdot x^{old}_{c_1s}$$
$$\forall s\in\mathcal{S}.$$

The last term on the left-hand side cancels out with the last term of the right-hand side:

$$\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (x^{new}_{c_1s} - w_{c_1sc_2s}) + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot x^{new}_{c_1s} + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - x^{new}_{c_2s}) \cdot x^{old}_{c_1s}$$
$$\leq$$
$$\overline{BDW}^{out}_s + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot x^{old}_{c_1s} + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot x^{old}_{c_1s} + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot x^{old}_{c_1s} \quad \forall s \in \mathcal{S}$$

Now, bringing terms from the right-hand side to the left-hand side and merging them with their respective counterparts, we get:

$$\sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot (x^{new}_{c_1s} - x^{old}_{c_1s} + (x^{old}_{c_1s} \cdot x^{old}_{c_2s}) - w_{c_1sc_2s}) + \sum_{c_1 \in \mathcal{C}_m} \sum_{c_2 \in (\mathcal{C} \setminus \mathcal{C}_m) \cup \{c_0\}} d_{c_1c_2} \cdot (1 - x^{old}_{c_2s}) \cdot (x^{new}_{c_1s} - x^{old}_{c_1s}) + \sum_{c_1 \notin \mathcal{C}_m} \sum_{c_2 \in \mathcal{C}_m} d_{c_1c_2} \cdot x^{old}_{c_1s} \cdot (x^{old}_{c_2s} - x^{new}_{c_2s}) \leq \overline{BDW}^{out}_s,$$

that is ready to be partitioned by customers to obtain exactly Constraints (2.31). Constraints (2.24) and (2.25) could be merged into a single set of constraint:

$$\pi_{c_1c_2} \geq \frac{1}{2}(z_{c_1} + z_{c_2}) \quad \forall c_1, c_2 \in \mathcal{C}_m \tag{2.24.alt}$$

This way we could reduce a bit the linearization overhead on the model. However, even if this choice does not change the solution when solving the problem in an exact way, it may have an impact when looking for approximate solutions (e.g. by relaxing variables' integrality). Constraints (2.39) and (2.40) could be merged accordingly into:

$$\pi^r_{c_1c_2} \geq 1 - \frac{1}{2} \sum_{s \in \mathcal{S}} (x^{old}_{c_1s} \cdot x^{new}_{c_1s} + x^{old}_{c_2s} \cdot x^{new}_{c_2s}) \quad \forall r \in \mathcal{R} \quad \forall c_1, c_2 \in \mathcal{C}_r \cap \mathcal{C}_m \tag{2.39.alt}$$

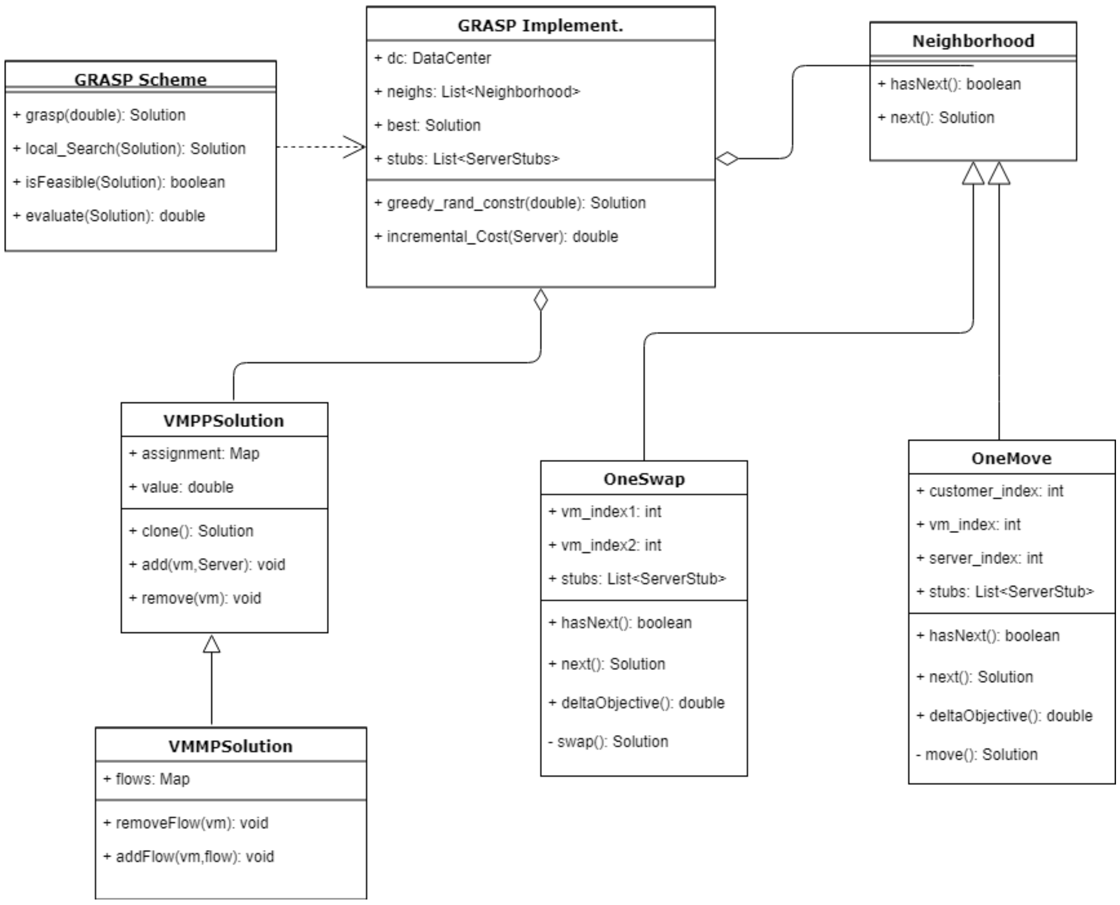


Figure 9.1: *High level GRASP Class Diagram*

9.4 High level UML Class Diagram

Here we report a very high level diagram for the two main components of our heuristics: the GRASP + Local Search procedure and the Path Relinking procedure. They are meant just as conceptual scheme to understand the code's organization, since the implementation details are much more involved.

Figure 9.1 summarize the structure of the GRASP heuristic's code. A GRASP template collects very general interfaces for interacting with a solution or starting the local search phase. Then, different implementations specify how the Greedy Randomized Construction is applied and how the incremental cost i computed. An implementation is also provided with a list of $Neighborhoods$ to run the Local Search procedure. Such Neighborhoods cycle through all solutions thanks to a repeated invocation of the procedure *next*, helped by the procedure $hasNext$ whose task is to detect the end of the neighborhood. Each

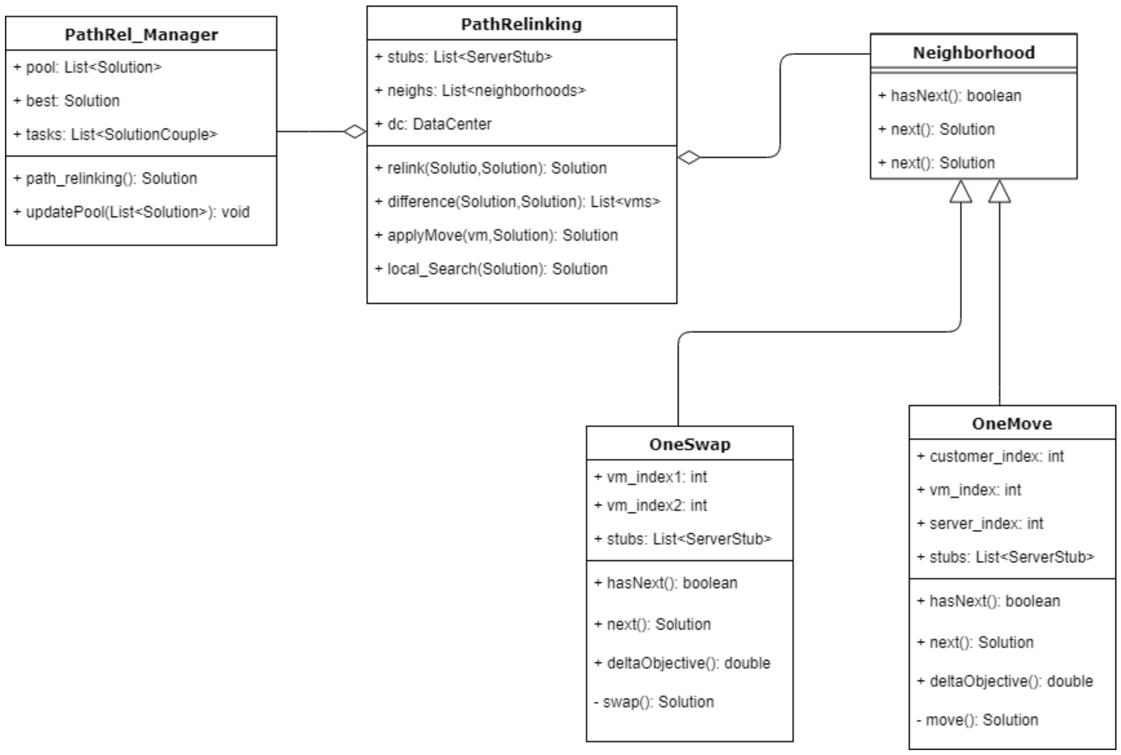

PathRel_Manager
+ pool: List<Solution>
+ best: Solution
+ tasks: List<SolutionCouple>
+ path_relinking(): Solution
+ updatePool(List<Solution>): void
PathRelinking
+ stubs: List<ServerStub>
+ neighs: List<neighborhoods>
+ dc: DataCenter
+ relink(Solutio,Solution): Solution
+ difference(Solution,Solution): List<vms>
+ applyMove(vm,Solution): Solution
+ local_Search(Solution): Solution
Neighborhood
+ hasNext(): boolean
+ next(): Solution
+ next(): Solution
OneSwap
+ vm_index1: int
+ vm_index2: int
+ stubs: List<ServerStub>
+ hasNext(): boolean
+ next(): Solution
+ deltaObjective(): double
- swap(): Solution
OneMove
+ customer_index: int
+ vm_index: int
+ server_index: int
+ stubs: List<ServerStub>
+ hasNext(): boolean
+ next(): Solution
+ deltaObjective(): double
- move(): Solution

Figure 9.2: *High level Path Relinking Class Diagram*

Neighborhoods carries ad-hoc interfaces to evaluate the cost of a new solution and to generate it.

A solution of the VMPP is stored as a mapping between VMs and servers, plus a value. Instead, a solution of the VMMP carries an additional information related to the paths/links used by a VM for its migration flow. Residual capacities related to a solution are not stored, since they can always be computed starting from the assignment mapping.

One key component of the whole framework is constituted by the *server stubs*. Every component that, at any point, has to manipulate a solution, needs a copy (stub) of each server. Indeed, since residual capacities are not stored for every solution, at every step the algorithm needs to manipulate them and keep them updated to the last solution generated. Therefore, a copy of each server is needed, otherwise different GRASP procedures working in parallel would interfere with each others making the capacities non-coherent with the generated solutions.

Figure 9.2 shows the structure of the Path Relinking code. As explained in Chapter 5.2.3, a component called *PathRel Manager* is in charge of maintaining a pool of solutions. Such pool is repeatedly updated after applying Path Relinking to every couple of different solutions in the pool. The actual Path Relinking between solution pairs is performed in parallel by a list of other identical components, called Path Relinking. This type of component is able to compute the difference between two solutions and to iteratively reduce it by applying a sequence of moves. It is also able to invoke a Local Search procedure, which works exactly as for the GRASP.

Both the GRASP code and the Path Relinking code read the input/instance from a complex data structure here summarized as DataCenter. It recursively contains pods, racks and servers as well as links and a graph in case of the VMMP. Additionally, another data structure, that is not reported in the figure, contains the list of customers and related traffic matrices.

Bibliography

[1] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[2] Microsoft Azure. General purpose virtual machine sizes. `https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-general`. Accessed: 2017-11-30.

[3] Amazon EC2. Amazon ec2 instance types. `https://aws.amazon.com/ec2/instance-types/?nc1=h_ls`. Accessed: 2017-11-30.

[4] SPEC. Specweb2009 results. `https://www.spec.org/web2009/results/`. Accessed: 2017-12-06.

[5] `http://www.raritan.com/blog/detail/the-top-five-data-center-management-challen` Accessed: 2018-03-04.

[6] `https://www.seagate.com/gb/en/tech-insights/data-center-management-master-ti/`. Accessed: 2018-03-04.

[7] Petter Svärd, B Hudzia, S Walsh, Johan Tordsson, and Erik Elmroth. The noble art of live vm migration-principles and performance of pre copy, post copy and hybrid migration of demanding workloads. *Technical report, 2014. Tech Report UMINF 14.10. Submitted*, 2014.

[8] Prateek Sharma, Lucas Chaufournier, Prashant J Shenoy, and YC Tay. Containers and virtual machines at scale: A comparative study. In *Middleware*, page 1, 2016.

[9] David Breitgand, Amir Epstein, Alex Glikson, Assaf Israel, and Danny Raz. Network aware virtual machine and image placement in a cloud. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 9–17. IEEE, 2013.

[10] Mohammad Masdari, Sayyid Shahab Nabavi, and Vafa Ahmadi. An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, 66:106–127, 2016.

[11] Xiang Sun, Nirwan Ansari, and Ruopeng Wang. Optimizing resource utilization of a data center. *IEEE Communications Surveys & Tutorials*, 18(4):2822–2846, 2016.

[12] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[13] Dallal Belabed, Stefano Secci, Guy Pujolle, and Deep Medhi. Striking a balance between traffic engineering and energy efficiency in virtual machine placement. *IEEE Transactions on Network and Service Management*, 12(2):202–216, 2015.

[14] Federico Larumbe and Brunilde Sansò. Elastic, on-line and network aware virtual machine placement within a data center. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 28–36. IEEE, 2017.

[15] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52:11–25, 2015.

[16] Jiaqiang Liu, Li Su, Yuchen Jin, Yong Li, Depeng Jin, and Lieguang Zeng. Optimal vm migration planning for data centers. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2332–2337. IEEE, 2014.

[17] Huandong Wang, Yong Li, Ying Zhang, and Depeng Jin. Virtual machine migration planning in software-defined networks. *IEEE Transactions on Cloud Computing*, 2017.

[18] Thuan Duong-Ba, Thinh Nguyen, Bella Bose, and Tuan Tran. Joint virtual machine placement and migration scheme for datacenters. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2320–2325. IEEE, 2014.

[19] Mauricio GC Resende and Celso C Ribeiro. Greedy randomized adaptive search procedures: Advances and applications. *Handbook of Metaheuristics, 2nd Edition, Springer*, 2008.

[20] Ramón Alvarez-Valdés, Francisco Parreño, and José Manuel Tamarit. A grasp/path relinking algorithm for two-and three-dimensional multiple bin-size bin packing problems. *Computers & Operations Research*, 40(12):3081–3090, 2013.

[21] `http://jgrapht.org/`. Accessed: 2018-03-02.

[22] Mauricio GC Resendel and Celso C Ribeiro. Grasp with path-relinking: Recent advances and applications. In *Metaheuristics: progress as real problem solvers*, pages 29–63. Springer, 2005.

[23] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[24] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[25] SPEC. Specpower_ssj2008 results. `https://www.spec.org/power_ssj2008/results/`. Accessed: 2017-11-30.

[26] Dong Zhang, Bing-Heng Yan, Zhen Feng, Chi Zhang, and Yu-Xin Wang. Container oriented job scheduling using linear programming model. In *Information Management (ICIM), 2017 3rd International Conference on*, pages 174–180. IEEE, 2017.

[27] Qinghua Zheng, Rui Li, Xiuqi Li, Nazaraf Shah, Jianke Zhang, Feng Tian, Kuo-Ming Chao, and Jia Li. Virtual machine consolidated placement based on multi-objective biogeography-based optimization. *Future Generation Computer Systems*, 54:95–122, 2016.

[28] Xibo Yao, Hua Wang, Chuangen Gao, Fangjin Zhu, and Linbo Zhai. Vm migration planning in software-defined data center networks. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*, pages 765–772. IEEE, 2016.

[29] SPEC. Specvirt2013 results. `https://www.spec.org/virt_sc2013/results/`. Accessed: 2018-01-26.

[30] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.

[31] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[32] `http://www.wikibench.eu/?page_id=60`. Accessed: 2018-01-14.

[33] Kuangyu Zheng, Wenli Zheng, Li Li, and Xiaorui Wang. Powernets: Coordinating data center network with servers and cooling for power optimization. *IEEE Transactions on Network and Service Management*, 2017.

[34] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. Wikipedia workload analysis. *Vrije Universiteit, Amsterdam, The Netherlands, Tech. Rep. IR-CS-041, Sepember*, 2007.

[35] `https://www.amazon.it/`.

[36] `https://www.mediaworld.it/`.

[37] `https://www.zalando.it/uomo-home/`.

[38] `https://www.intesasanpaolo.com/`.

[39] `https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html`. Accessed: 2018-03-18.