PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

UILLIAN LUIZ LUDWIG

OPTIMIZING THE PERFORMANCE OF MULTI-TIER APPLICATIONS USING INTERFERENCE AND AFFINITY-AWARE PLACEMENT ALGORITHMS

Porto Alegre
2018

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
COMPUTER SCIENCE GRADUATE PROGRAM

OPTIMIZING THE PERFORMANCE OF MULTI-TIER APPLICATIONS USING INTERFERENCE AND AFFINITY-AWARE PLACEMENT ALGORITHMS

UILLIAN LUIZ LUDWIG

Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fullfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. César A. F. De Rose

Porto Alegre
2019

Ficha Catalográfica

L948o Ludwig, Uillian Luiz

Optimizing the Performance of Multi-tier Applications Using Interference and Affinity-aware Placement Algorithms / Uillian Luiz Ludwig . – 2018.
65.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. César Augusto Fonticielha De Rose.

1. Multi-tier applications. 2. Performance interference. 3. Network Affinity. 4. Application placement. I. De Rose, César Augusto Fonticielha. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Uillian Luiz Ludwig

Optimizing the Performance of Multi-tier Applications Using Interference and Affinity-aware Placement Algorithms

This Dissertation/Thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor/Master of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on March 16, 2018.

COMMITTEE MEMBERS:

Prof. Dr. Thiago Ferreto (PPGCC/PUCRS)

Prof. Dr. Andrea Charrão (PPGCC/UFSM)

Prof. Dr. César A. F. De Rose (PPGCC/PUCRS - Advisor)

MELHORANDO A PERFORMANCE DE APLICAÇÕES MULTI-TIER USANDO ALGORITMOS DE PLACEMENT BASEADOS EM INTERFERÊNCIA E AFINIDADE

RESUMO

Os provedores de nuvem estão constantemente buscando tornar-se mais econômicos, onde uma estratégia comum é consolidar múltiplas aplicações em uma máquina física, usando técnicas como a virtualização. Esta consolidação, no entanto, traz alguns problemas relacionados ao desempenho, como a interferência de recursos. Além disso, dadas as características das aplicações multicamadas, nas quais as camadas precisam se comunicar através da rede, temos outra fonte de perda de desempenho, que, neste documento, nos referimos como afinidade de rede. Para reduzir os efeitos desses problemas, técnicas de *placement* são usadas para melhor distribuir as aplicações nas máquinas físicas. Várias dessas técnicas de *placement* consideram a interferência de recursos ou a afinidade da rede para decidir a melhor distribuição; No entanto, nenhum deles se concentra em ambos os critérios ao mesmo tempo. Esse fato leva a um *placement* que não está totalmente otimizado. Por esse motivo, implementamos um conjunto de algoritmos de *placement* que denominados CIAPA. Ele visa encontrar o melhor meio termo entre interferência e afinidade. CIAPA usa um modelo de degradação de desempenho, funções de custo e heurísticas para encontrar o *placement* com o custo mínimo. Nos experimentos, comparamos o custo alcançado pela CIAPA com outras estratégias de *placement* e verificamos que, para as cargas de trabalho utilizadas, CIAPA gera decisões de *placement* com melhor custo e, consequentemente, com melhor o desempenho.

Palavras-Chave: aplicações multicamadas, interferência de recursos, afinidade de rede, placement de aplicações.

OPTIMIZING THE PERFORMANCE OF MULTI-TIER APPLICATIONS USING INTERFERENCE AND AFFINITY-AWARE PLACEMENT ALGORITHMS

ABSTRACT

Cloud providers are constantly seeking to become more cost effective, where a common strategy is to consolidate multiple applications in a physical machine, using techniques such as virtualization. This consolidation, however, brings some performance related problems such as resource interference. Moreover, given the characteristics of multi-tier applications, in which tiers need to communicate through the network, we have another source of performance degradation, which we refer as network affinity. In order to reduce the effects of such problems, placement techniques are used to better distribute the applications in the physical machines. Several of these placement techniques consider resource interference or network affinity in order to decide the best placement; however, none of them focus on both criteria at the same time. This fact leads to a placement that is not fully optimized. For this reason, we implemented a set of placement algorithms called CIAPA, which aims at finding the best trade-off between interference and affinity. CIAPA uses a performance degradation model, a cost function, and heuristics to find the placement with minimum cost. In the experiments, we compared the cost achieved by CIAPA with other placement strategies, and we have verified that, for the workloads used, it generates placement decisions with better cost, and, consequently, improved performance.

Keywords: multi-tier applications, performance interference, network affinity, application placement.

LIST OF FIGURES

1.1 Conflicting placement of a multi-tier application: resource interference (a) vs. network bottleneck (b). 20
2.1 Example of the multi-tier architecture of an e-commerce system. 26
4.1 Architecture of the node-tiers benchmark. 32
4.2 Example of an IntP output for a disk intensive application. 36
4.3 Response time of application CPU-CPU while varying the workload. 37
4.4 Response time of application disk-disk while varying the workload. 37
4.5 Response time of application none-none while varying the workload. 38
4.6 Impact of increasing the workload in the response time and request status of the execution of the disk-disk application with high network affinity. 38
4.7 Illustration of the placement policies based on the forces pushing them. . . 39
5.1 Class diagram of the core of CIAPA. 50
5.2 Screenshot of the list of tiers added to the CIAPA web interface. 50
5.3 Placement settings that allow to customize the heuristics execution. 51
5.4 Cost comparison between executed heuristics. 51
5.5 Result of a placement execution of the simulated annealing heuristic. 52
5.6 Placement decision workflow using CIAPA. 52
6.1 Cost comparison between hill climbing, simulated annealing, and round robin. 56
6.2 Number of PMs needed to meet the cost threshold using the threshold-based heuristics. 57
6.3 Cost comparison of CIAPA-SA vs. interference vs. affinity-aware strategies. 58
6.4 Characteristics of the multi-tier applications used in the use case analysis. 59
6.5 Cost and average response time comparison of different placement strategies. 60

LIST OF TABLES

3.1 Comparison of placement strategies from the related works. 29
4.1 Example of tiers available in the node-tiers benchmark. 32
4.2 Environment architecture and characteristics. 34
5.1 Performance degradation generated by resource interference and network affinity. 42
5.2 Notations for the problem formulation. 43
6.1 Distribution of tiers per PM using each placement strategy for case III. 60
6.2 Distribution of tiers per PM using each placement strategy for case IV. 60

LIST OF ALGORITHMS

5.1 Round robin decreasing placement algorithm. 46
5.2 Stochastic Hill Climb heuristic. 46
5.3 Randomize function that generates a modified solution. 47
5.4 Simulated Annealing heuristic. 47
5.5 Base algorithm for the threshold-based heuristics. 49

CONTENTS

1 INTRODUCTION 19
1.1 OBJECTIVES 20
1.2 DOCUMENT STRUCTURE 21

2 BACKGROUND 23
2.1 APPLICATION PLACEMENT 23
2.2 PERFORMANCE INTERFERENCE 24
2.3 MULTI-TIER ARCHITECTURE 25

3 RELATED WORK 27

4 PRELIMINARY EXPERIMENTS 31
4.1 INTERFERENCE AND AFFINITY CHARACTERIZATION 31
4.1.1 NODE-TIERS: A MULTI-TIER BENCHMARK 31
4.1.2 INTP: AN INTERFERENCE PROFILER 34
4.2 MULTI-TIER APPLICATIONS PERFORMANCE ANALYSIS 35
4.3 PLACEMENT POLICIES 38

5 PLACEMENT MODEL AND ALGORITHMS 41
5.1 PERFORMANCE DEGRADATION MODEL 41
5.2 PROBLEM FORMULATION 43
5.3 HEURISTICS 45
5.3.1 OPTIMIZATION-BASED HEURISTICS 46
5.3.2 THRESHOLD-BASED HEURISTICS 48
5.4 VISUALIZATION TOOL 49
5.5 PLACEMENT DECISION WORKFLOW 51

6 PERFORMANCE EVALUATION 55
6.1 OPTIMIZATION-BASED HEURISTICS 55
6.2 THRESHOLD-BASED HEURISTICS 56
6.3 COMPARISON OF CIAPA AGAINST INTERFERENCE AND AFFINITY AWARE ONLY ALGORITHMS 58
6.4 USE CASE ANALYSIS 58

7 CONCLUSION 61

REFERENCES **63**

1. INTRODUCTION

With the advent of the use of consolidate environments, such as clouds, and the constant changes in software development, multi-tier applications have become a very popular method of development of several types of applications, including web and mobile [CHRD03]. These web and mobile applications are of high importance for the society, where several companies rely on them to operate their business. The performance of these applications is a key factor for companies' revenue because application slowdowns or downtimes lead to unsatisfied clients and, consequently, revenue loss.

Cloud computing has an interesting concept of pay-as-you-go, where customers pay only for what they actually use [MG11]. Also, the customers have the notion that they have unlimited resources, so they should be able to easily scale their applications. With this definition, it may sound like, by using clouds, performance problems would be eliminated at a fair price. However, cloud providers use consolidate environments, and multiple applications share the same resource of a physical machine. Even though the resource sharing techniques have evolved, they still bring problems such as resource interference, which may severely degrade the applications' performance.

In addition to the problems brought by consolidation, the use of the multi-tier architecture generates a great deal of pressure on the network. By implementing applications in the multi-tier architecture, the application is broken down into several modules, called tiers, and each module is responsible for executing a specific task. Moreover, modules may depend on the execution of other modules; thus, they may need to communicate through the network in order to finalize the requested task. This strategy of development brings several advantages, such as the ease of reusability and maintainability [HHM04]. On the other hand, the network may become a bottleneck given the high necessity of communication between tiers [FMCB17].

Given these problems of resource interference and network bottleneck, one may think that cloud computing and multi-tier applications are not the best fit for the development of applications. However, these methods of development provide the best performance among other strategies, but in the Computer Science world, it is always necessary to seek to extract more performance from the available resources. This better resource utilization leads to better services provided by clouds and to fewer expenses to customers. Therefore, the objective of this master's thesis is to optimize the performance of multi-tier applications that are running in consolidated environments. This optimization will be done by generating a better distribution of applications per physical machines so that resource interference and network bottlenecks are minimized.

In order to better illustrate the challenge, Figure 1.1 shows two placements of one multi-tier application with two tiers. In placement a, the application is suffering from

performance degradation due to resource interference. On the other hand, in placement b, the application is suffering from performance degradation due to network overhead. The challenge here, and the goal of this work, is not to eliminate performance degradation, but rather to find the placement that leads to the lowest performance degradation possible.

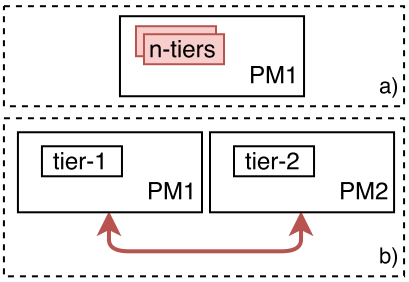


Figure 1.1: Conflicting placement of a multi-tier application: resource interference (a) vs. network bottleneck (b).

1.1 Objectives

The main objective of this master's thesis is to **create a family of placement algorithms that is able to optimize the performance of multi-tier applications that are running in consolidated environments**. This is important because we verified the importance of multi-tier applications for businesses, where they are widely used, and slowdowns and downtimes are very damaging to companies' profits. Moreover, we plan to achieve this objective by considering the resource interference and network affinity of these multi-tier applications. Given that, as a side objective, **we demonstrate the impact of resource interference and network affinity on multi-tier applications, and how different placement settings are able to improve the overall application's performance**.

The importance of this master's thesis is supported by several related works that focus on similar issues. However, **we demonstrate that these works lack at considering both criteria (interference and affinity) at the same time, and the most optimal placement is not achieved**. This fact is demonstrated through experiments that take into consideration a cost metric that we have created as well as the performance of applications running under each placement setting. Furthermore, we created two different placement approaches, using different heuristics. Given this fact, **we analyze and compare the performance of the approaches and heuristics that we have implemented**.

1.2 Document Structure

This document is divided into seven chapters, where each chapter builds on top of the previous ones. In the current chapter, we detailed the motivation and objectives of this master's thesis. Next, on Chapter 2, we give a detailed review of the important topics of this paper, including application placement, performance interference, and multi-tier architecture. On Chapter 3, we go further and detail other research works that have similar objectives. This review of related works is important to understand what has been done by other researchers, and what is still opened to contributions.

Given the understanding of what has been done and what is still missing, the following chapters provide our views and solutions to fill this gap. Chapter 4 shows the preliminary experiments that we have done, including the implementation of a multi-tier benchmark, analysis of the application's performance under different placements, and the creation of a set of placement policies. Then, expanding the findings and contributions of Chapter 4, on Chapter 5, we describe a detailed placement solution, which consists of models, equations, heuristics, and user interface. This solution should be able to ease the placement decision process, generating placement decisions that lead to better performance.

Taking the implemented solution, Chapter 6 shows the performance evaluation of the implemented placement algorithms. We compare the costs achieved by implemented heuristics, deciding which heuristic returns the best results. Moreover, we compare the results with some placement strategies from the literature. Finally, Chapter 7 ends the document, listing the contributions, publications, and future works.

2. BACKGROUND

In this chapter, we detail the three most important areas to understand the remaining of this document. Firstly, we characterize application placement strategies. Secondly, we explain the problem of performance interference and its impact on performance. Lastly, we explore the multi-tier architecture and its importance.

2.1 Application Placement

In multi-tenant environments, such as clouds, a physical machine (PM) shares its resources in order to host several related or unrelated applications. This resource sharing is done through the use of specific techniques, such as virtualization and containerization [DRK14]. These techniques help to keep the infrastructure costs low since it enables a higher number of applications to be deployed in the same PM. However, since the PMs' resources are limited, there needs to be a definition of how many and which applications will be hosted in each PM. Therefore, application placement strategies are valuable to find the best distribution of applications per physical machines.

The main goal of placement strategies is to use the available infrastructure in the most efficient way. This efficiency, however, is seen differently by each cloud provider. For example, some providers aim at placing the most applications per PM, while others aim at generating a placement that minimizes the service-level agreement violations (SLA) [MNA16]. In either way, the use of smart placement strategies is important to meet the desired efficiency.

Ideally, placement strategies should share these following objectives [MNA16]:

- Reduction of data-center traffic and distribution of this traffic evenly in the data-center network.
- Maximization of resource utilization to effectively use every PM.
- Minimization of the number of active PMs, which leads to a reduction of energy consumption and costs.
- Minimization of service level agreement (SLA) violations.
- Improvement of application locality, placing applications closer to the user.

In addition, Masdari et al. classify placement strategies in four groups [MNA16]. The first group is *resource-aware*, where it takes into consideration the resource requirements of each application. In most of the strategies, CPU, memory, and network are the

resources that are considered. In a few strategies, I/O and the number of active PMs are also considered. The second group is *power-aware*, which aims to reduce power consumption, leading to less cost and less CO_2 emission. The main factors considered are the CPU utilization, the PMs power usage, and the PMs states. The third group is *network-aware*. In this group, the strategies consider mainly the traffic between PMs, generating a placement with lower network overhead. The last group is *cost-aware*, which analyses the PM cost and cooling cost as its main factors.

All these strategies need to have efficient and automatic methods to find the solution that meets the desired objective. Most of them rely on the bin-packing problem, which considers application placement as the classical problem of fitting items in a bin. Since the bin-packing problem is NP-complete, it is important to use heuristics that are able to find a good solution in a polynomial time. In this case, the main heuristics used are the first fit, best fit, and worst fit.

While these bin-packing heuristics provide good results, there are cases that a more complex approach must be used. One of these approaches is the constraint programming, which is a programming paradigm where the solution must satisfy all of the problem's constraints. Moreover, other approaches rely on optimization strategies, and given an initial solution, they iterate in order to find the optimal solution. Some examples of such approaches would include integer programming, genetic algorithms, and simulated annealing [US16].

2.2 Performance Interference

With the use of resource sharing techniques, PMs host multiple applications. Each application uses a slice of the PM's resource, such as storage, and memory. However, some resources, such as disk I/O, network I/O, and LLC cannot be sliced [IHJ11]. Therefore, when multiple applications need to use these resources at the same time, there will be resource contention. This problem is known as performance interference, and it may lead to severe performance degradation.

Yang et al. analyzed the performance degradation caused by performance interference in MapReduce jobs [YDZ16]. They verified that the resource interference has a great impact on the performance of MapReduce jobs. Moreover, they have implemented a scheduler that tries to minimize such impact. This scheduler tries to predict the interference a job will have based on other jobs placed in the same PM. In their experiments, they demonstrated that the implemented scheduler was able to reduce the impact of performance interference, leading MapReduce jobs to have a better performance.

Chi et al. demonstrate the performance degradation caused by performance interference [CQL14]. In their experiments, they have tested several combinations of appli-

cations running in the same PM. They have found that depending on the applications that are co-hosted, the performance varies considerably. Moreover, CPU intensive applications did not suffer much while being placed with other CPU intensive applications. On the other hand, disk I/O intensive applications suffered a great deal while being placed with other disk I/O applications.

Jersak and Ferreto showed in their experiments how CPU, memory, and disk I/O intensive applications are affected by other virtual machines (VMs) using the same resource intensively in the same PM [JF16]. They also have verified that each resource is affected differently. For example, the addition of several CPU intensive applications led to a performance degradation of 14%. On the other hand, for memory and disk I/O intensive applications, the performance degradation was as high as 90%. Therefore, it is clear that performance interference is a problem, and the performance degradation varies depending on the resource that is used the most.

In addition to VMs being affected by performance interference, Xavier et al. studied how performance interference affects container-based clouds [XDOR+15]. They analyzed the performance degradation suffered on disk-intensive applications caused by containers that use different resources intensively. They have tested several combinations of workloads running on the same host. While some of these combinations led to a performance degradation of 38%, they could also combine the workloads with no interference. Thus, it shows the importance of understanding each application's workload and smartly placing them in order to minimize interference.

2.3 Multi-tier Architecture

In the past, applications were commonly implemented in the client/server style, where the client was a desktop application and the server was a database running remotely. This client/server architecture is rarely used nowadays because applications have grown more complex, and a simple database running remotely is not able to handle this complexity [Dat17]. Therefore, this client/server architecture has adapted to what we know today as multi-tier architecture.

In a multi-tier architecture, the application is broken down into several modules, called tiers. Each tier is responsible for executing a specific task, such as handling user authentication or parsing logs. Moreover, tiers may depend on other tiers to execute, and they must communicate through the network in order to answer each request [CHRD03]. This communication plays a major role in the applications' performance, and network delays may cause significant slowdown. Therefore, this complexity of network dependency of tiers puts a high pressure on having a good management of the infrastructure so that network bottlenecks can be avoided [HHM04].

As example of a multi-tier application, Figure 2.1 shows the simplified architecture of an e-commerce system. There, we can notice a high dependency between layers and tiers, which is denoted by the arrows connecting each tier. There, in order to a request be answered, multiple communications between tiers must be executed. Furthermore, tiers have different resource requirements and generate different levels of resource interference. Thus, a good understanding of each tier characteristics is necessary for better managing the infrastructure.

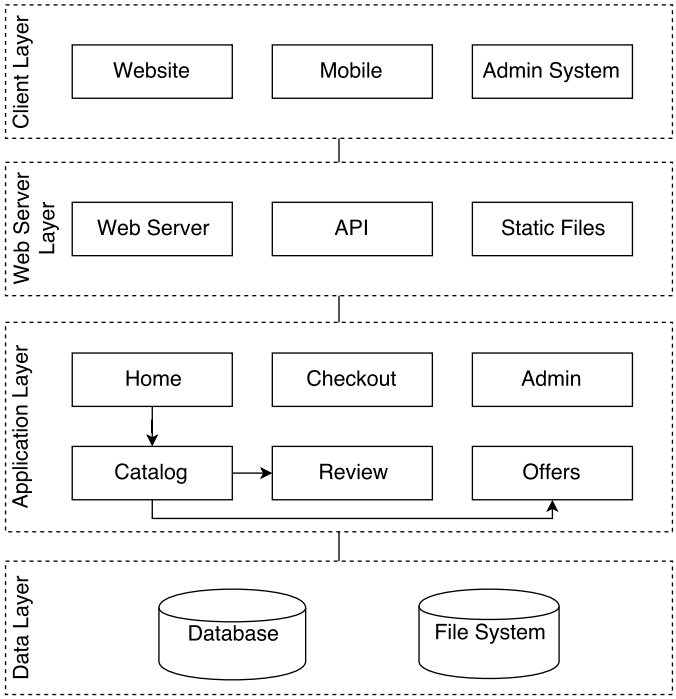


Figure 2.1: Example of the multi-tier architecture of an e-commerce system.

With the rise of applications running on the cloud, including the multi-tier ones, cloud data centers have become more concerned with network utilization. An overloaded network impacts the cloud in several ways, such as increase of infrastructure costs and reduction of Quality of Service (QoS) [FMCB17]. Moreover, as multi-tier applications are composed of several individual applications and as clouds host multiple applications in the same PM, resource interference may be another factor of concern. Therefore, given these problems, it is not only important for cloud providers to use techniques to reduce network traffic, placing applications close to the data they need, but it is also important to avoid placing applications that use the same resource intensively in the same PM. However, these are conflicting requirements, and it is difficult to be achieved. In order to better understand these placement techniques, next chapter will detail some works that are focused on generating placements that take these criteria into consideration.

3. RELATED WORK

The focus on improving performance by using smart placement techniques has generated a great number of research works in the literature. These works use several strategies, but here, we are concerned about the ones that use performance interference or network affinity metrics. In this chapter, we are going to detail the most relevant of these works.

Somani et al. presented VUPIC, which is a virtual machine placement scheme that takes into consideration the performance isolation and the resource utilization in order to decide the best placement configuration [SKP12]. They classify the VMs into three different groups based on the resources they use more intensively: CPU, disk, and network. Then, when they are making placement decisions they favor to place VMs that use different resources or same resources but with lower intensity in the same PM. With their experiments, they demonstrate that VUPIC was able to increase performance while still maintaining good resource utilization.

Caglar et al. presented the harmonious Art of Living Together (hALT), which is a middleware for placement of VMs that uses machine learning [CSG11]. hALT monitors CPU and memory utilization and the performance of virtual machines. Then, using this historical data, it finds patterns of performance interference that leads to performance degradation. These patterns are used with a neural network in order to make placement decisions that will minimize such interference.

Jersak and Ferreto presented a performance-aware placement of VMs, which uses as main metric the performance interference [JF16]. In order to generate such placement, they have built an interference model, considering the resources CPU, memory, and disk I/O. Their model returns the interference level of placing a given number of applications that use the same resource intensively in the same PM. Moreover, in order to generate a placement with the minimum PMs possible, they used an interference level threshold. Then, their algorithm indicates what is the minimum number of PMs necessary to meet the desired threshold.

Chiang and Huang presented TRACON, which is a task and resource allocation control framework [CH11]. This framework is most focused on reducing the interference generated by data-intensive applications. They built machine learning algorithms to infer the interference levels generated by each virtual machine. This inference is used by the scheduler algorithm to generate a distribution of VMs per PM that leads to best resource utilization. In their experiments, they demonstrated that TRACON is able to improve the applications' runtime by 50%.

In addition, there were three other works that generate placement decisions based on resource interference. Firstly, Jin et al. presented CCAP, which is a VM place-

ment for HPC cloud that takes into consideration the cache contention in order to decide the best placement [JQWG15]. Secondly, Bu et al. studied an interference and locality-aware scheduling of MapReduce jobs, where a job is placed closer to the data. It also takes into consideration the interference from other jobs running in the same PM [BRX13]. Lastly, Xu et al. presented iAware, which is a system that tries to avoid the interference on VMs when a migration process happens. This is important in order to verify if a migration would actually benefit the environment or if it would bring performance degradation due to the interference while migrating it [XLL+14].

Ferdaus et al. focus on the network affinity of tiers in order to decide the best placement of multi-tier applications [FMCB17]. Their main goal is to reduce the network overhead of data center infrastructures. Therefore, they place tiers that communicate closer. In their experiments, they demonstrate that their placement strategy is able to reduce network cost and to increase the number of applications deployed in the infrastructure.

Another work that focuses on network affinity is presented by Su et al. [SXC+15]. In their strategy, they group applications based on their communication patterns, and those applications that have a network relation are placed in the same group. Then, when deciding on which PM to place each application, they favor placing applications from the same group on the same PM. Thus, reducing the network overhead across physical networks.

Sonnek et al. present a migration strategy that reduces the communication overhead in virtualized environments [SGRC10]. They monitor the network communication between virtual machines, and they adapt the placement according to the communication behavior. In their experiments, they used MPI applications, and they were able to improve the runtime up to 42%. Also, it enabled a reduction of up to 85% network communication costs.

Also focusing on network, but at this time only on network interference, Lin and Chen focused on reducing the performance degradation caused by network I/O interference [LC12]. Network I/O is a resource that cannot be sliced between VMs, so if multiple VMs are using it intensively, it will lead to performance degradation. In order to reduce this problem, they have modeled a placement scheme that takes into consideration the resource demand, the QoS, and the resource interference. With their experiments, they demonstrate that the proposed placement scheme leads to a better profit and a lower number of QoS violations than other schemes that they compared with.

It is clear that resource interference and network affinity are important metrics for making placement decisions. However, to the best of our knowledge, there is not a work that uses both metrics at the same time. Moreover, we believe that they cannot be looked separately, because if just the resource interference is considered, tiers that need to communicate may be placed far, while if only network affinity is considered, tiers that

have high interference may be placed in the same host. Therefore, it is important to find the best trade-off between interference and affinity, generating a placement that leads to the best performance possible. Moreover, to better summarize the studied works, Table 3.1 lists the placement strategies along with the main factors taken into consideration, the resources analyzed, and the environment target.

Table 3.1: Comparison of placement strategies from the related works.

Reference	**Main factors**	**Resources analyzed**	**Target**
[SKP12]	Performance isolation, continuous resource usage	CPU, network, and disk I/O	IaaS clouds
[CSG11]	Resource interference, resource overbooking	CPU and memory history	Soft real-time applications in the Cloud
[JF16]	Resource interference, number of active PMs	CPU, memory, and disk I/O	Clouds
[JQWG15]	Cache interference	Cache	HPC Cloud
[BRX13]	Resource interference, data locality	CPU and disk I/O	MapReduce jobs
[MYXW13]	Resource interference, energy-efficiency	CPU and memory	Clouds
[CH11]	Resource interference	Disk I/O	Data-intensive applications
[XLL+14]	Resource interference	CPU, memory, network, and cache	VM migration
[SXC+15]	Resource affinity and conflict	Network	Heterogeneous Data Centers
[LC12]	Resource interference, QoS, resource demand	Network	Clouds
[FMCB17]	Network and data-aware	Network	Cloud data centers
[SGRC10]	Communication overhead	Network	VM Migration

4. PRELIMINARY EXPERIMENTS

In this chapter, we present the first experiments that we have done for this work. These experiments were the main focus of a paper that we published in the scheduling track of WSCAD 2017 conference [LKCDR17]. Here, we are primarily focusing on verifying the impact of resource interference and network affinity on different applications and placement strategies. Then, with the analysis of such impact, we generated a set of placement policies that should lead multi-tier applications to have better performance. Moreover, in order to generate these policies, Section 4.1 describes how we characterized the resource interference and network affinity of multi-tier applications. Then, the analysis is detailed in Section 4.2, and the placement policies are described in Section 4.3.

4.1 Interference and Affinity Characterization

In the first steps of this work, we had to decide how we would characterize the resource interference and network affinity of multi-tier applications. For this, we faced two main challenges. First, we had to define the application target that we would use for our experiments. Second, we had to understand the interference and affinity levels and find a way to measure them. The following sections describe how we solved each one of these challenges.

4.1.1 Node-tiers: a Multi-tier Benchmark

As the first challenge, we had to decide which multi-tier application we would use as our target. We analyzed the characteristics of the benchmarks that are used in other research projects in the literature, and we found that none of them were suitable for our purposes. We needed a multi-tier benchmark, but most of the benchmarks used to characterize performance interference are not implemented in this architecture. Most of them execute a large task, which usually takes up to several minutes to finish and stress a specific resource. However, the tasks in multi-tier applications take a short time to finish, usually less than a second. The high load comes from the high number of requests arriving at the same time, rather than the execution of a big task. Thus, this type of benchmarks would not be useful for characterizing the multi-tier applications.

Given the lack of an appropriate benchmark, in order to meet our research necessities, and, perhaps, to help other researchers, we have built an open-source multi-tier

benchmark called node-tiers[1]. This benchmark allows us to simulate a multi-tier application with any number of tiers, where each tier executes different tasks, using different computing resources. Moreover, node-tiers is able to simulate the communication between tiers, where it is possible to set the amount of data that is being communicated on each request. Thus, node-tiers allows simulating a wide variety of applications workloads. An overview of the architecture of node-tiers is shown in Figure 4.1.

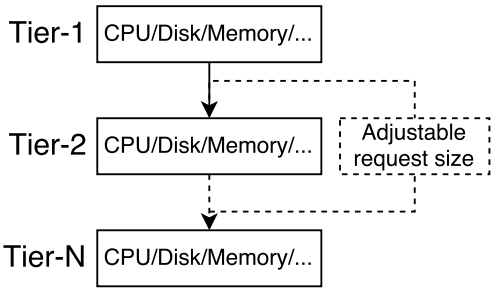


Figure 4.1: Architecture of the node-tiers benchmark.

Node-tiers has several types of tiers available, including tiers that use CPU, memory, disk, and cache. Some tiers use only one resource, while others use multiple resources in order to complete their task. Node-tiers is also easily extensible and other types of tiers can be created. This allows a simulation of a varied type of applications. As examples of tiers implemented, Table 4.1 shows a list of tiers implemented, the method that it executes, and the available customization of each method. Also, the table shows the ID that is the parameter passed to the application that identifies which tier should be executed. The definition of the type of operation executed by each tier was inspired by the popular benchmark stress-ng [Kin17].

Table 4.1: Example of tiers available in the node-tiers benchmark.

Resource	Method	Customization	ID
CPU	PI calculation	# of Iterations	pi
Memory	Memory allocation/deallocation	# of Iterations	memory
Disk I/O	Database insertion	Text length	writeDatabase
Cache	zlib compression/decompression	Text length, iterations	zlib
Mixed	Matrix multiplication	Matrix size	matrix
None	Nothing is executed	–	none

As example of the usability of node-tiers, we can consider the RUBiS benchmark, which is a widely used multi-tier benchmark that simulates an e-commerce system [OW201]. Even though we can use RUBiS itself, it is a benchmark that is hard to be setup and to put into execution. RUBiS consists of a database tier, which is mostly disk I/O intensive, and a web-server tier, which is mostly CPU intensive. We can simulate RUBiS using node-tiers by creating two tiers, where one would simulate the database tier running the

[1]Source code available at https://github.com/uillianluiz/node-tiers

writeDatabase tier, and the other would simulate the web-server tier running the *pi* tier. Depending on the workload of the web-server, however, it could be used the *matrix* tier, which provides a more mixed resource utilization.

Node-tiers runs as a web server, and it waits for HTTP requests on the specified port. Each request needs to inform which tier is going to be executed as well as any tiers it should communicate with. As example, Listing 4.1 shows a Javascript Object Notation (JSON) that should be sent through a POST request to the first tier. When the tier receives the JSON, it executes its task, and it generates a request to the next host, sending a request of the specified size. The size of the request is generated by sending a random text, which is generated with the specified request size. This process keeps happening until all hosts have processed. Moreover, as a response, each host returns the time it took to process the task and the time it has waited for the following tier to process its task.

```
1 { "nextTiers":
2     [
3         { "url": "http://hostTwo:3000/TIER_ID2", "requestSize": 1000},
4         { "url": "http://hostThree:3000/TIER_ID3", "requestSize": 16000},
5         ...,
6         { "url": "http://hostN:3000/TIER_IDN", "requestSize": 128000}
7     ]
8 }
```

Listing 4.1: Example of JSON message expected by the first tier.

In order to simulate the behavior of multi-tier applications, each request was designed to finish in a short time (less than 100ms). Therefore, in order to achieve higher load, thus, higher interference, it is necessary to send multiple simultaneous requests. For this reason, we used a load tool called Artillery, which creates virtual users and sends requests to the application [Sho17]. An example of an Artillery configuration file is seen in Listing 4.2. There, we defined on line 2 which host the request would be sent, as well as its task ID. Also, we defined the duration, request rate, and request timeout. Moreover, on line 12, we should include the JSON that indicates the tiers that this first host should communicate with.

```
1 config:
2   target: 'http://hostOne:3000/TIER_ID1'
3   phases:
4     - duration: 600
5       arrivalRate: 1
6       rampTo: 300
7   http:
8     timeout: 60
```

```
scenarios:
    - post:
        json:
          nextTiers: [...]
```

Listing 4.2: Example of configuration file used with Artillery.

For all experiments using node-tiers, we used the environment summarized in Table 4.2. Moreover, the resource sharing was done through processes, and no virtualization/containerization technique was utilized.

Table 4.2: Environment architecture and characteristics.

Resource	Description
Processor	Intel(R) Xeon(R) CPU X6550 @ 2.00GHz x2
Memory	64GB DDR3
Disk	146GB - Model: ST9146803SS
Network	Gigabit Ethernet
Number of PMs	2
Cores per tier	4
Memory per tier	1.76GB

4.1.2 IntP: an Interference Profiler

In order to characterize the interference generated by each application tier, we used a tool called IntP. IntP is an interference profiling tool developed by Xavier as part of his PhD in Computer Science [Xav18]. IntP monitors an application process that it expects as a parameter. It profiles the application during runtime, returning the interference the application generates on each resource. It communicates with the Linux kernel, collecting the necessary metrics, such as performance counters, for the calculation of the resource interference. Moreover, IntP returns the interference metrics, in percentage, every second, where the higher the metric is, the more interference the application being profiled generates.

IntP returns the interference metrics for CPU, disk, memory, network, and cache. More specifically, it returns the following metrics:

- **netp** - percentage of physical network interference.
- **nets** - percentage of network queue interference.
- **blk** - percentage of disk interference.
- **mbw** - percentage of memory bus interference.

- **llcmr** - percentage of cache miss.
- **llocc** - percentage of cache interference.
- **cpu** - percentage of cpu interference.

The profiling done with IntP returns the resource interference during the period that the application is monitored. This fact brings an important concern regarding the application workload. For example, given a web application that has a seasonal workload, the profiling will only know about the interference for the workload that the application had while it was profiled. If the workload changes, the resource interference levels might change as well; therefore, a new profiling must be made in order to characterize the new behavior. For this reason, it is important to have a good understanding of the application workload, profiling it with the most accurate one.

In addition to generating interference metrics, IntP is also useful for verifying the network affinity between two applications. IntP gives the network interference that an application generates, but for this work, we are not considering it as network interference, but rather as network affinity. For example, if an application has network interference, it means it communicates with another application. This means that the application has a network affinity with this other application. Moreover, the network affinity levels are defined by the interference level given by the network interference. Therefore, the higher the value is, the higher the affinity between two applications is.

Figure 4.2 shows the interference levels of an application while varying its workload from 0 to 300 requests per second. This application is disk intensive and it has a high network affinity with another application. It is noticeable that the interference levels tend to increase as the workload also increases. Moreover, the disk interference has a unique behavior, which varies between every data collection. This behavior is due to writing operations being executed in an asynchronous manner.

4.2 Multi-tier Applications Performance Analysis

For the performance analysis, we are using the node-tiers benchmark, considering three multi-tier applications with two tiers each, where both tiers stress the same resource. The first application was CPU-intensive, the second was disk-intensive, and the last did not use any resource intensively. Moreover, we generated an increasing workload, varying the request rate from 0 to 300 requests per second. This variation directly impacts the resource interference and network affinity levels since higher request rate leads to more resources needed to answer the requests. Furthermore, we have considered two placement variations, where in the first both tiers were placed in the same PM

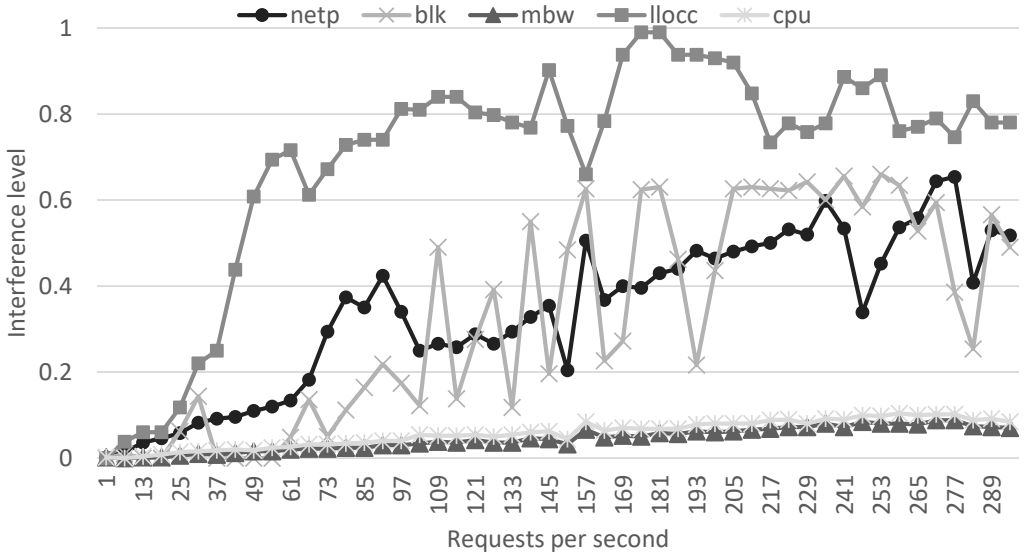


Figure 4.2: Example of an IntP output for a disk intensive application.

and in the second each tier was placed in a different PM. We have also considered two variations of network affinity, where in the first, the application would have a low communication affinity, where the request size was set to 1KB. In the second variation, the application had a high communication affinity, and the request size was set to 512KB. The analysis of these experiments is detailed in the following paragraphs.

Figure 4.3 shows the performance of an application consisted of two CPU intensive tiers. The response time axis is shown in logarithmic scale for better visualization. It can be noticed that the execution with higher request size (512KB) had a worse performance as compared with the lower request size (1KB). This is a natural behavior since the higher the request size is, the more pressure it puts on both operating system and physical network. Additionally, while the request rate was low, the performance for all executions remained stable. However, as the request rate increased, the execution with high network affinity running in different PMs suffered performance degradation. In this case, the network becomes flooded with many requests, and as the network bottleneck is reached, the response time increases exponentially. On the other hand, while running with same request size, but in the same PM, there is no impact on the performance. Therefore, we can conclude that, for this CPU intensive workload, network affinity is a more critical problem as compared to resource interference.

Figure 4.4 presents the response time of the application that had two disk I/O intensive tiers. The response time kept acceptable while the workload was low. However, as the workload increased, the application presents a different behavior from the one seen in the CPU intensive application. All four executions of this application have performance degradation, but this degradation comes earlier in the executions that run the tiers on the same PM. Furthermore, the network affinity also has an impact on the performance, and the higher the request size is, the higher the impact is. As a conclusion of this execution,

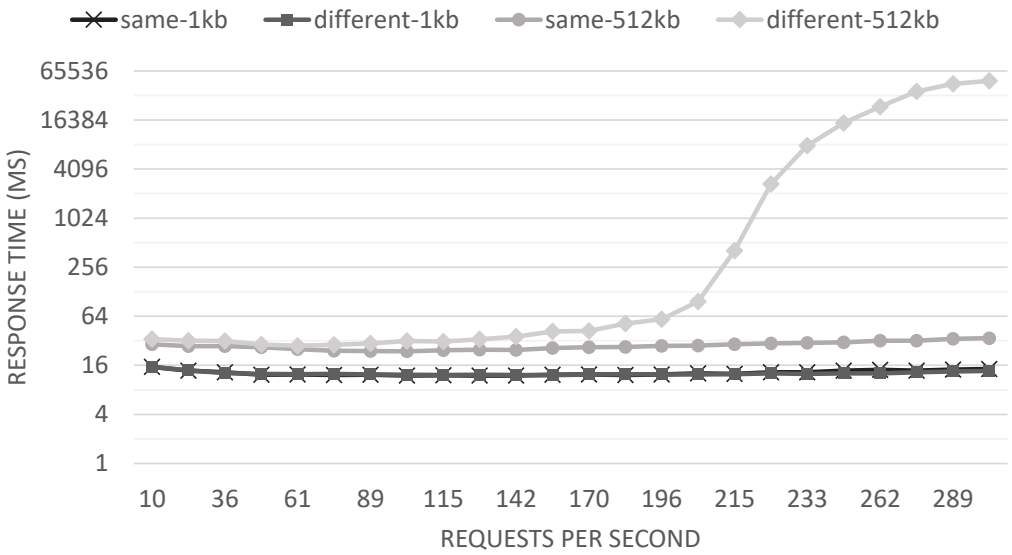


Figure 4.3: Response time of application CPU-CPU while varying the workload.

disk I/O intensive applications tend to suffer more from the interference of co-hosted tiers, but there is still impact generated from different network affinity levels.

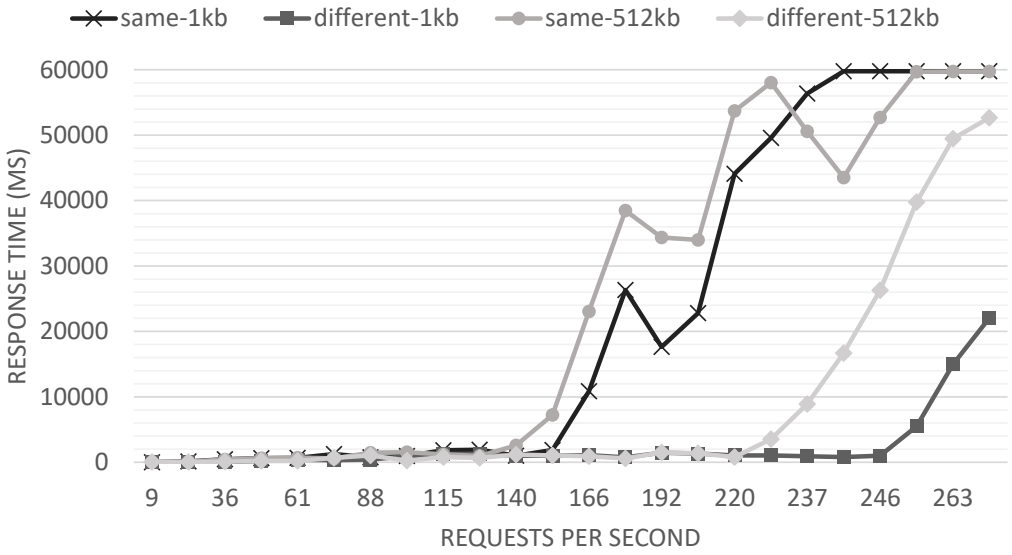


Figure 4.4: Response time of application disk-disk while varying the workload.

Figure 4.5, which contains the execution of an application with two non-intensive tiers, shows a similar behavior to the one seen in the CPU intensive application. This is due to the fact that both applications are not highly affected by interference, and the characteristic that generates the most impact is the network affinity.

Going further into the analysis of these executions, Figure 4.6 shows how the requests' status was affected by the increase in the request rate. It shows the impact of the execution of the disk intensive application with 512KB of bandwidth. The chart on the left shows the results of the execution in the same PM. It can be noticed that while the request rate was low, the response time was constant and the number of requests

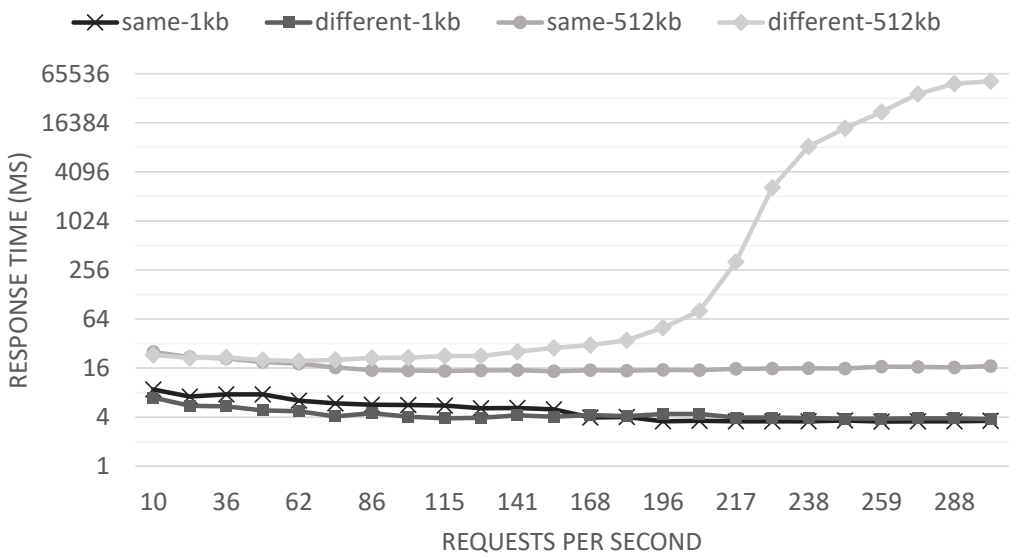


Figure 4.5: Response time of application none-none while varying the workload.

with OK status was growing linearly. However, as the request rate surpassed 150, the response time started to grow rapidly, the requests started taking longer to finish, and, finally, all of them were failing. In contrast, the chart on the right shows the execution of the same benchmark configuration but running in different PMs. As we saw in the previous analysis, the execution on different PMs had a better performance. Here, we can see that the effect on the request status matched the response time, and requests only failed when the request rate was above 270.

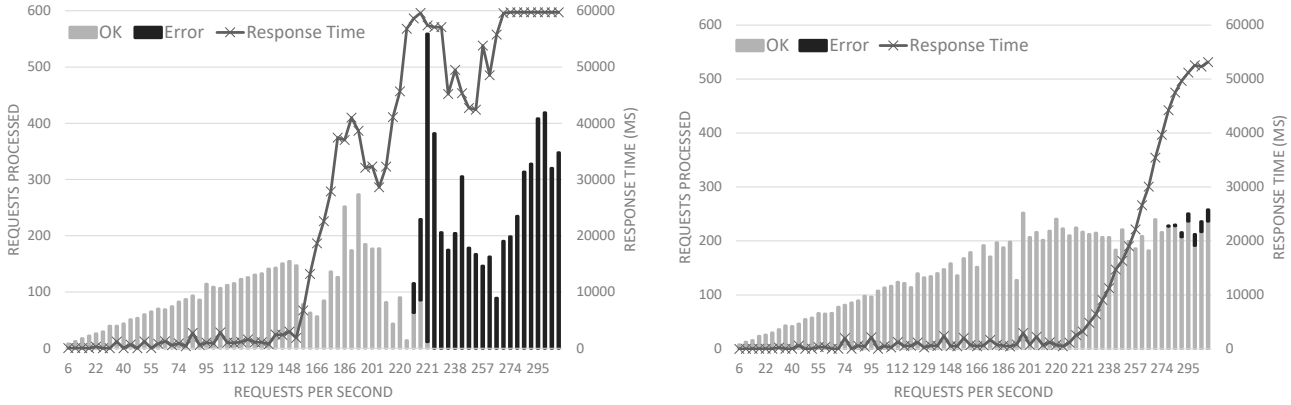


Figure 4.6: Impact of increasing the workload in the response time and request status of the execution of the disk-disk application with high network affinity.

4.3 Placement Policies

Taking the results of the experiments into consideration, we proceeded and created a set of placement rules. As seen in the experiments, the workload has a great impact

on the performance of the application. For this reason, the following rules focus on maximizing the workload that the application is able to handle without having performance degradation. Moreover, we detail these policies by describing forces that push the tiers closer or farther depending on the intensity of the interference and affinity.

R1. Tiers that interfere the same resource should be placed in different PMs. When there are few servers available, PMs might have to host tiers that interfere the same resource. However, it should be given preference for separating the tiers that generate the most performance degradation. In this case, disk I/O tiers have a strong repulsion, while CPU intensive tiers have a weak repulsion.

R2. Tiers that have network affinity should be placed in the same PM. When all the resources of a given PM are being used, it should be given preference to place in the same PM the tiers that have higher affinity, i.e. higher request size. Moreover, the higher the affinity is, the more attraction force it generates.

R3. Tiers that do not interfere nor have network affinity may be placed anywhere in the infrastructure. There is no force pushing the tiers.

R4. Tiers that interfere and have network affinity should follow a sub-set of rules. These sub-rules extract the best trade-off, which should generate a placement that leads to the best quality of service (QoS) possible.

R4a. CPU intensive tiers should be placed in the same PM. The attraction force generated by affinity is stronger than the repulsion force generated by interference.

R4b. Disk I/O intensive tiers should be placed in separate PMs. The performance degradation that comes from interference leads to a stronger repulsion than the attraction generated by network affinity.

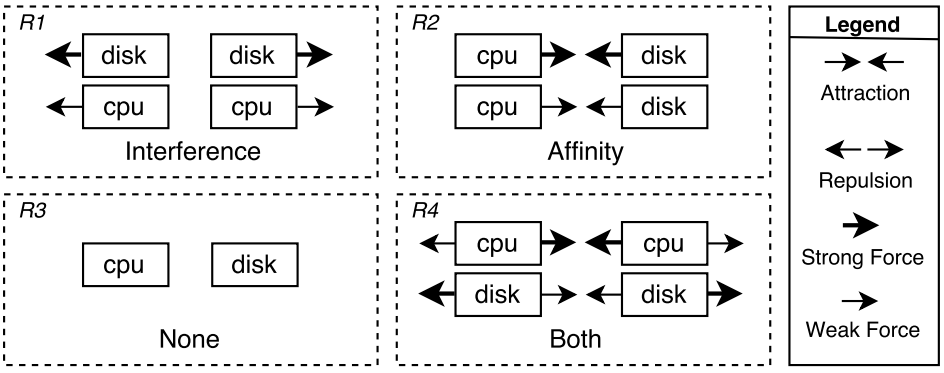


Figure 4.7: Illustration of the placement policies based on the forces pushing them.

Figure 4.7 shows an illustration of these placement policies. It demonstrates the forces that network affinity and resource interference put in the tiers, attracting them to

the same PM (affinity) or repelling them to different PMs (interference). The line width represents the strength of the force pushing them.

Even though these policies are useful for optimizing the placement, they have two main weaknesses. Firstly, it would be important to expand these policies, considering other resources, such as memory and cache. However, the policies would become too complex and hard to be understood. Secondly, these policies do not consider the levels of resource interference and network affinity. This would lead to applications to not have the optimal placement. For these reasons, in Chapter 5, we present a more advanced technique for deciding the best placement of multi-tier applications.

5. PLACEMENT MODEL AND ALGORITHMS

In the preliminary experiments, we demonstrated that resource interference and network affinity are important characteristics that must be considered when placing multi-tier applications. Moreover, we presented a set of placement policies that are able to aid administrators to make placement decisions. In this chapter, however, we present a more sophisticated approach, where we use models, functions, and heuristics to automatize the decision process, generating more reliable results. This approach generated a set of placement algorithms, which we called CIAPA, which stands for Capacity, Interference and Affinity-aware Placement Algorithms. In the following sections, we describe the process of creating of CIAPA.

5.1 Performance Degradation Model

In Section 4.1.2, we described how IntP returns the interference levels as the percentage of how much the profiled application interferes each resource. In a first moment, it sounds like a good idea to use these levels to determine how good a placement is. However, as we have seen in Chapter 2, each resource suffers differently from resource interference. For example, a high level of disk interference is much more prejudicial than a high level of CPU interference. For this reason, we are not going to use the interference levels by itself, but rather the performance degradation a given interference level generates.

In order to simplify the problem, avoiding to have to characterize the performance degradation of many levels (0.0-100.0), we classified the interference and affinity levels into four classes: absent, low, moderate, and high. Even though this classification reduces the breadth of the problem, it is still an improvement from related works, which mostly consider only two levels (absent and present). Moreover, the percentage levels that each class represents are the following: absent is 0; low varies between 1 to 20; moderate varies between 21 to 50; finally, high varies between 51 to 100. This division was done this way because we understood that these ranges better represents the classes as compared to an equal division.

Given these four classes, we verified the impact that each class generates on co-hosted applications. For this, we considered the performance degradation generated by each class, where for node-tiers the performance metric considered was the response time. For the first class, absent, there is no performance degradation, i.e., it increases the response time in 1.0 times. For the following classes, we conducted experiments in order to obtain the performance degradation. These experiments led to the creation of a model

that gives the performance degradation expected from running two tiers in the same or different PMs. These experiments are detailed in the following paragraphs.

In order to obtain the performance degradation generated by interference, we used one application for each resource type. Then, we collected the average response time of the application while it was running without the effects of performance interference. This execution was done using Artillery, running with 50 concurrent users during 40 minutes. After, we executed three cases, adding in the same PM applications with low, moderate, and high interference levels. The performance degradation for each interference level was calculated using the following division $perf_{class}/perf_{absent}$, where $perf_{class}$ is the average response time for each interference class, and $perf_{absent}$ is the average response time while executing it without the interference effects.

To illustrate, let us take as example the memory resource. We executed a memory intensive application under no interference, and we retrieved the average runtime of all requests, which resulted in $perf_{absent}$ = 22.8ms. Then, when co-hosting a low intensive memory application, the runtime increased to $perf_{low}$ = 24.4ms, which gives a performance degradation of 1.07 times. When co-hosting a moderate intensive application, the response time increase to $perf_{moderate}$ = 36.9ms, resulting in a performance degradation of 1.62 times. Finally, when co-hosting a high intensive memory application, the response time increased to $perf_{high}$ = 39.7ms, resulting in a performance degradation of 1.74 times.

In addition to calculating the performance degradation due to resource interference, we used a similar approach to model the performance degradation due to network affinity. For this case, we have used an application with two tiers that did not stress any resource but the network. Moreover, the base case is when both applications were running in the same PM, thus, there is no performance degradation due to network overhead. Then, for the affinity levels low, moderate, and high, we placed the applications in different PMs. The affinity levels varied according to the size of each request, where low was 1KB, moderate was 128KB, and high was 256KB. The performance degradation was calculated using the same method as the interference one, taking into consideration the response time of each class. Furthermore, based on this methodology, we characterized the interference and affinity performance degradation on our environment, and the model shown in Table 5.1.

Table 5.1: Performance degradation generated by resource interference and network affinity.

Level/Resource	CPU	Memory	Disk	Cache	Affinity
Absent	1.00	1.00	1.00	1.00	1.00
Low	1.03	1.07	1.12	1.07	1.05
Moderate	1.15	1.62	1.82	1.18	1.32
High	1.33	1.74	2.25	1.26	1.57

In order to illustrate the usability of this model, let us take a simple example of a two-tier application. Tier 1 has a moderate level of CPU interference, while Tier 2 has a high level of CPU interference. Also, Tier 2 has a high level of affinity with Tier 1. In this case, if they are placed in different PMs, they will have no performance degradation due to interference, but Tier 2 response time will increase 1.57 times due to network overhead. On the other hand, if they are placed in the same PM, Tier 1 will suffer 1.33, while Tier 2 will suffer 1.15 times. Thus, taking this numbers into consideration, we are able to find the best placement setting with lowest performance degradation. For example, even if we add or multiply the performance degradation generated by interference, the performance degradation generated by network affinity is greater; therefore, the best placement setting would be to place both tiers in the same PM.

5.2 Problem Formulation

Placement algorithms are usually seen as bin-packing problems, where tiers need to be packed in physical machines. In our problem, however, it is not enough to pack the tiers in the minimum number of PMs. Here, we should try to minimize the performance degradation generated by resource interference and network overhead. Over this section, we detail the mathematical formulation that we used in order to solve this problem, including the cost and placement functions. As a start, Table 5.2 shows the summary of notations that we used in this document.

Table 5.2: Notations for the problem formulation.

Symbol	Meaning
T'	A set of tiers. A tier is defined as a tuple $T = (I, A, S)$.
I	A tuple of performance degradation generated by interference. The tuple is defined as $I = (cpu, memory, disk, cache)$. These values depend on the tiers' interference levels, and they are taken from the model seen in Table 5.1.
A	A set of network affinity between tier T and other tiers in set T'. An element of the set A is defined as $(T_i, affinity_i)$, where $T_i \in T'$ and $affinity_i$ is the performance degradation given by the model seen in Table 5.1.
S	Size of tier T, where $S > 0$.
I'	Set of interference elements from each tier in a set T'.
A'	Set of affinity elements from each tier in a set T'.
S'	Set of size elements from each tier in a set T'.
P'	A set of PMs that will host a sub-set of tiers. A PM is defined as $P = C$.
C	Capacity of a PM P, where $C > 0$.

CIAPA aims to find the best distribution of tiers per PM while keeping the interference and affinity costs low and not exceeding the PMs capacity. Taking these three criteria

into consideration, we built three cost functions, which return the cost of placing a set of tiers in the same PM.

The first cost function, seen in Equation 5.1, works as a capacity constraint. It guarantees that the given PM P has enough capacity to host all tiers represented by the size set S'. If P is not able to host all of them, i.e., the sum of tiers sizes is greater than the capacity, it returns a high cost value, defined as MAX, which basically invalidate the given configuration; otherwise, it returns 1, which is a number that when multiplied will not modify the total cost.

$$f_{cap}(S', P) = \begin{cases} MAX & \sum_{S \in S'} S > P_C \\ 1 & otherwise \end{cases} \tag{5.1}$$

The second cost function, seen in Equation 5.2, gives the total interference cost of running a set of tiers T', represented by their interference set I', in the same PM. Moreover, the total interference cost is given by the multiplication of the cost of each resource, which is calculated by using the function seen in Equation 5.4. Also, there is a helper function g, seen in Equation 5.3, that returns a set of values that are greater than 1, i.e., which cause performance degradation. Furthermore, the function $f_{int'}$ obtains the set of values that cause performance degradation, and, in the case when there is more than one tier that generates interference over the same resource, their performance degradation values are multiplied and returned as the cost; otherwise, if there is zero or one tier that interfere a given resource, it returns 1.

$$f_{int}(I') = f_{int'}(I'_{cpu}) * f_{int'}(I'_{memory}) * f_{int'}(I'_{disk}) * f_{int'}(I'_{cache}) \tag{5.2}$$

$$g(I'_{res}) = \{I \mid I \in I'_{res}, I > 1\} \tag{5.3}$$

$$f_{int'}(I'_{res}) = \begin{cases} \prod_{I \in g(I'_{res})} I & |g(I'_{res})| > 1 \\ 1 & otherwise \end{cases} \tag{5.4}$$

The last cost function, seen in Equation 5.5, returns the cost of running the set of tiers T', represented by their affinity set A', in the same PM. It iterates through each affinity entry of each tier, and by calling a helper function, seen in Equation 5.6, it calculates the total cost. The cost is 1 (no cost) when all tiers that have affinity relation are also present in the same set of tiers T', which are placed in the same PM. If a tier is not present, the returned cost is the multiplication of each performance degradation that it will have by running tiers in different PMs.

$$f_{aff}(A', T') = \prod_{A \in A'} \prod_{a \in A} f_{aff'}(a, T') \tag{5.5}$$

$$f_{aff'}(a, T') = \begin{cases} 1 & a_T \in T' \\ a_{affinity} & otherwise \end{cases} \quad (5.6)$$

Given these three cost functions, Equation 5.7 shows a function that returns total cost of running a set of tiers T' in a PM P. Then, Equation 5.8 shows the placement function, which tries to minimize the total cost by testing all possible combinations of tiers per PMs. Moreover, the total cost of a placement is given by the average costs of each PM. The choice for the average of costs per PM was made in order to try to keep the costs the same among all PMs. While if we would use multiplication of costs, it could lead to some PMs with cost low, while others with cost really high.

$$f(T', P) = f_{int}(I') * f_{aff}(A', T') * f_{cap}(S', P) \quad (5.7)$$

$$p(T', P') = min(avg(f(T'_1, P_1), ..., f(T'_n, P_m)), \\ avg(f(T'_2, P_1), f(T'_3, P_2)), ..., avg(f(T', P_1))) \quad (5.8)$$

The placement function should be able to return the best placement configuration; however, this problem is harder than bin-packing since it is not enough to fit all tiers in the PMs, but it is necessary to minimize the cost. Therefore, this problem is NP-complete, and it is not computable when the input is large. For this reason, Section 5.3 shows the heuristics that we have used in order to optimize the solution.

5.3 Heuristics

Given the high complexity of the placement function, heuristics are an alternative to find a solution in acceptable time. In this section, we demonstrate the use of two different types of heuristics. First, we use two optimization-based approaches, which works as the minimization function seen in Equation 5.8. These heuristics start from an initial placement, and they iterate in order to reduce the placement cost. The heuristics used in this approach were the stochastic hill climbing and the simulated annealing. Second, we used a threshold-based approach, inspired by the one used by Jersak and Ferreto. In this case, there is no limit of number of PMs, since more PMs may be needed in order to meet the cost threshold. The heuristics used with this objective were first fit, best fit, and worst fit. In the following paragraphs, we are going into detail how each heuristic was implemented.

5.3.1 Optimization-Based Heuristics

In the optimization-based heuristics, an initial state is generated, and a set of operations are executed, trying to reduce the placement cost. The initial solution is generated by calling a round robin decreasing (RRD) placement, as seen in Algorithm 5.1. In this strategy, the tiers are sorted decreasingly by their size, and they are placed in the PMs in a sequential order. Here, we refer to the placement distribution as a solution. Moreover, this RRD strategy is one of the most simple ones, yet it is widely used because it has virtually no computational cost.

Data: P', T'
T' = sortDecreasingBySize(T')
$pmIndex$ = 0
foreach T ***in*** T' **do**
  P'[$pmIndex$].push(T)
  $pmIndex$ += 1
  $pmIndex$ = $pmIndex$ % $P'.length$
end
return $newSolution(P')$

Algorithm 5.1: Round robin decreasing placement algorithm.

The first heuristic using the optimization approach implemented was the stochastic hill climbing, which its pseudo-code is shown in Algorithm 5.2 [SG06]. This heuristic expects three parameters: set of tiers, set of PMs, and number of iterations. Moreover, it starts by generating an initial solution by calling RRD algorithm. Then, for every iteration, it tries to optimize the placement cost, generating random modifications in the solution. A new solution is taken as the best if it reduces the current cost. This procedure keeps repeating for the defined number of iterations. In the end, the solution with the lowest cost is returned.

Data: $P', T', iterations$
Result: s_{best}
s_{best} = roundRobinDecreasing(P', T')
while $iterations > 0$ **do**
  s_{new} = randomize(s_{best})
  if $s_{new}.cost < s_{best}.cost$ **then**
    s_{best} = s_{new}
  end
  $iterations$ -= 1
end
return s_{best}

Algorithm 5.2: Stochastic Hill Climb heuristic.

This algorithm is stochastic due to the fact that on every iteration it generates a random modification of the current solution. In a deterministic algorithm, however, it

would have to explore all possible modifications from one solution, which would bring back the high complexity. Moreover, the $randomize$ function, which its pseudo-code is seen in Algorithm 5.3, receives the current solution, and it executes a move or a swap operation. The move operation moves a random tier from one PM to another. The swap operation swaps two random tiers from different PMs. Furthermore, the move and swap operations have a probability of 50% each to be executed.

Data: s
$probability$ = rand(0, 1)
if $probability < 0.5$ **then**
    $tier_1$ = randomTier(s)
    $tier_2$ = randomTier(s)
    swap($tier_1, tier_2$)
else
    $tier$ = randomTier(s)
    pm = randomPM(s)
    move($tier, pm$)
return s

Algorithm 5.3: Randomize function that generates a modified solution.

Hill climbing is an algorithm that returns the minimum local, and in some cases, it might not be the minimum global. This is because it only accepts a new solution if it has a lower cost; however, sometimes it is necessary to take a worse solution in order to improve later. For this reason, we also have used a heuristic called simulated annealing, which is designed to find the global minimum [KGV83]. Its pseudo-code is seen in Algorithm 5.4, and it works in a very similar way as compared to hill climbing.

Data: $P', T', temperature, coolingRate$
Result: s_{best}
s = generateInitialSolution(P', T')
s_{best} = s
while $temperature > 1$ **do**
    s_{new} = randomize(s)
    if $P(s, s_{new}, temperature) \geq random(0, 1)$ **then**
        $s = s_{new}$
    end
    if $s.cost < s_{best}.cost$ **then**
        $s_{best} = s$
    end
    $temperature$ =* $1 - coolingRate$
end
return s_{best}

Algorithm 5.4: Simulated Annealing heuristic.

Simulated annealing, instead of receiving the number of iterations as hill climbing, expects two other parameters: temperature and cooling rate. The temperature starts high, and the cooling rate determines how much it will decrease over each iteration. Fur-

thermore, the main reason behind the use of a temperature is that, when the temperature is high, the acceptance probability function P, which is seen in Equation 5.9, will have higher changes of accepting a worse solution. This means that in the beginning, the algorithm will try several different solutions, even if they are worse; however, as it is reaching the end of the execution, it will tend to accept only better solutions. However, like hill climbing, it had to be implemented in a stochastic manner by using the $randomize$ function; therefore, even though it will optimize the placement, it may still fail at returning the minimum global at all times.

$$P(s, s', T) = \begin{cases} 1 & s'_{cost} < s_{cost} \\ exp((s - s')/T) & otherwise \end{cases} \tag{5.9}$$

5.3.2 Threshold-Based Heuristics

In addition to this strategy that finds the best placement using only the PMs available, another approach, which was found in the literature and described in the related work, is to set a PM cost threshold and use as many PMs as necessary in order to meet the desired threshold. With this objective, we used three common bin-packing heuristics that use the capacity and the cost in order to decide the best placement. These heuristics are first fit decreasing, best fit decreasing, and worst fit decreasing.

Since these heuristics have a different approach, which works by giving as result the number of PMs necessary to meet the given cost, we believe it is a good option to add to CIAPA. In this approach, it is more focused on getting the desired performance, given by the cost threshold, whereas in the optimization-based, it is more focused on getting the best distribution given an environment. Moreover, we believe that both strategies are relevant, and further in the evaluation chapter, we describe the results achieved with both of them.

The heuristics used in this threshold-based approach are very simple, and their decision process works very similarly. They iterate through the sorted list of tiers, placing them in the PM that each strategy decides that is the best. For first fit, each tier is placed in the first PM that has enough capacity and meets the cost threshold. For best fit, the tier is placed in the PM that, after the tier is placed, will have the least free space and that will still meet the cost threshold. Finally, for worst fit, the tier is placed in the PM that, after the tier is placed, will have the most free space and that will still meet the cost threshold. If, for any of the heuristics, there is no PMs that meet capacity and cost threshold, the tier is added to a new PM that is created in the environment. The pseudo-code of these heuristics is seen in Algorithm 5.5. This pseudo-code is an abstraction for all three heuristics, and the decision of which PM each tier will be placed was simplified

by using the function $getPMMeetingThreshold$. If there is no PM available, the function returns NULL, and a new PM is added to the solution.

Data: $P', T', heuristic$
Result: P'
sortDecreasingBySize(T')
foreach $T \in T'$ **do**
  P = getPMMeetingThreshold($heuristic, P', T$)
  if P == *NULL* **then**
    P = newPM()
    P'.addPM(P)
  end
  P.addTier(T);
end
return P'

Algorithm 5.5: Base algorithm for the threshold-based heuristics.

5.4 Visualization Tool

In order to provide a working prototype of CIAPA, we built an open-source visualization tool that allows easy interaction with the placement algorithms[1]. It was developed using the most recent web technologies, such as TypeScript, which is a superset of JavaScript, and Angular, which is a front-end web application platform. Moreover, this web interface runs all in the client-side, i.e., an internet browser.

It is known that, for most tasks, JavaScript provides a lower performance as compared to other languages such as C++ and Java [Mat14]. However, the use of this web stack turned out to be useful because it provided a faster development process given our previous knowledge of such stack. Moreover, the focus of the interface is not to provide real time results, but rather to provide a method for easy and fast interaction with the placement algorithms. That being said, it still provides placement results in a reasonable time.

We can divide the implementation of CIAPA into two parts: core and user interface. The core is the most important part of CIAPA, and it contains the models and methods for selecting the best placement as described in the previous sections. Figure 5.1 shows the class diagram of application core. It lists the main elements along with their attributes and methods.

The user interface followed the core implementation and it was divided into three main areas: tiers, PMs, and placement. Figure 5.2 shows the tiers area, where all tiers added are listed. There, it displays all the tier related information, and by clicking in a row,

[1] Source code available at https://github.com/uillianluiz/ciapa

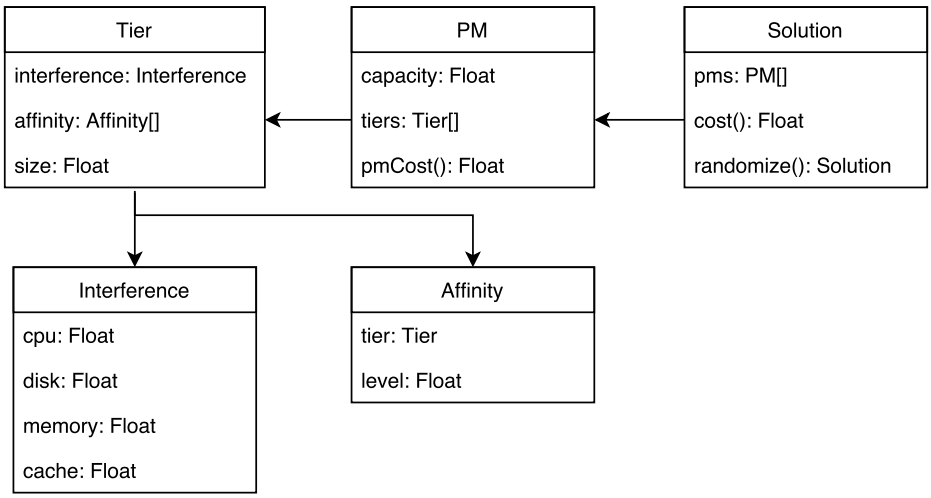


Figure 5.1: Class diagram of the core of CIAPA.

it allows the details to be updated. Moreover, the PMs area follows the same structure, where the only customizable parameter is the capacity of each PM.

Name	Size	CPU	Memory	Disk	Cache	Affinity
tier5	Nano	Moderate	Low	Low	Moderate	1 entries
tier4	Nano	Moderate	Low	Moderate	Low	0 entries
tier3	Micro	Absent	Low	High	Absent	1 entries
tier2	Xlarge	High	Low	Moderate	Moderate	0 entries
tier1	Large	Low	Moderate	Low	Low	1 entries

1 - 5 of 5 tiers

Figure 5.2: Screenshot of the list of tiers added to the CIAPA web interface.

In the placement area, which is shown in Figure 5.3, it is possible to customize the execution of the placement algorithms. The heuristics simulated annealing and hill climbing are always executed, and their parameters can be customized in order to provide a shorter or longer search. Moreover, it gives the option to select whether a comparison with placements that only consider resource interference or network affinity should be made. Finally, it gives an option to select whether the threshold-based algorithms should be executed.

When the placement is generated, it shows several tabs, displaying important information about the placement heuristics. The first tab, as seen in Figure 5.4, shows the cost comparison between all heuristics that were executed. This allows a quick overview of how well each placement strategy performs. Besides the cost comparison, it also shows a comparison of the number of physical machines used, which is a valuable information for the threshold-based heuristics.

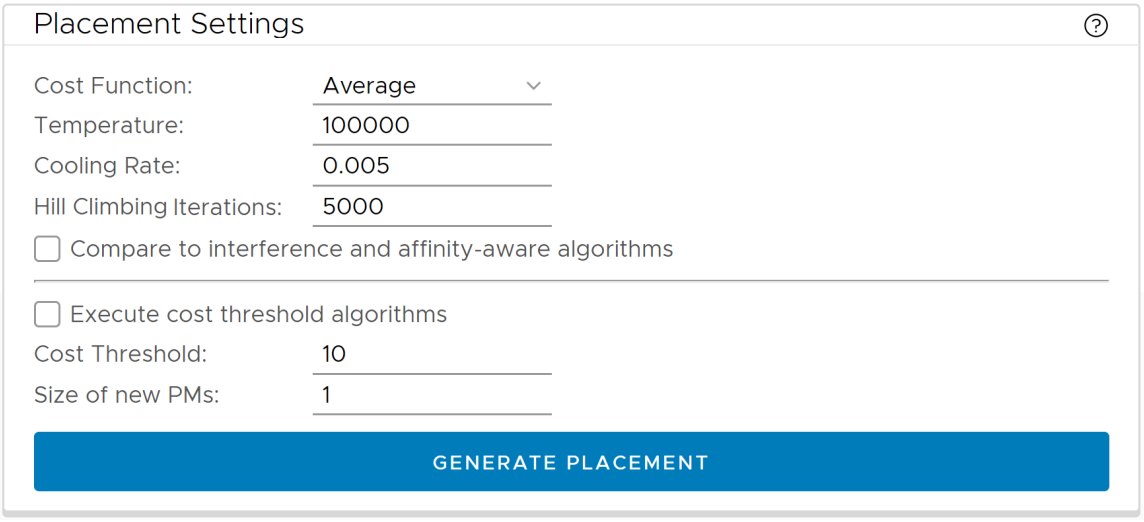


Figure 5.3: Placement settings that allow to customize the heuristics execution.

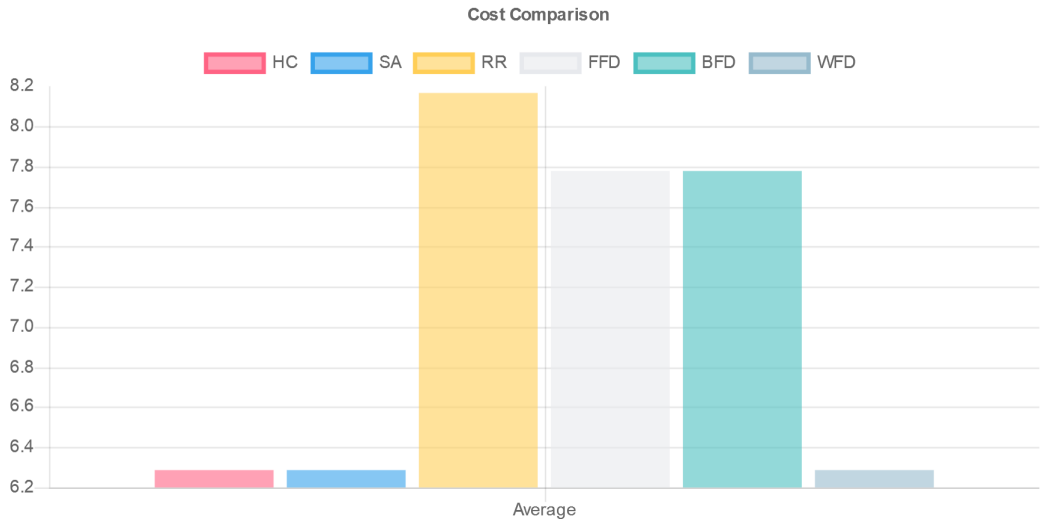


Figure 5.4: Cost comparison between executed heuristics.

The other tabs, one for each heuristic, displays the final distribution of tiers per PM. An example of such distribution is seen in Figure 5.5. There, we see the total cost of the placement, the cost per PM, and the capacity used. It also shows an export button, which allows the user to export a CSV file with the distribution of tiers per PM.

5.5 Placement Decision Workflow

In order to generate the placement decision, there is a basic workflow that should be followed. It consists of a set of steps, which are described in the following paragraphs and summarized in Figure 5.6.

The first step is to tune the performance degradation model that was presented in Section 5.1. The model that we created represents our current infrastructure. Even though

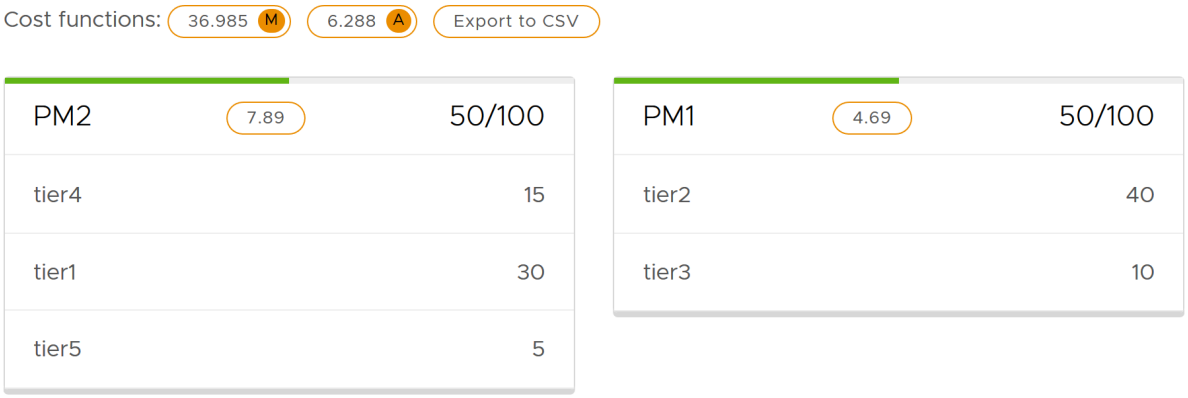


Figure 5.5: Result of a placement execution of the simulated annealing heuristic.

it could still be used for different environments, it may fail at providing the best placement because different environments suffer differently from resource interference and network affinity. Then, for the second step, it is necessary to characterize the interference and affinity levels of each tier, classifying them into one of the four classes. These two steps are done using the IntP profiling tool.

After this data collection, this information should be inputted in the CIAPA web interface. With this information, CIAPA is able to execute the placement algorithms described, generating placement distributions. Then, given the various results, the user should compare the placement cost of each heuristic, and choose the one with the lowest cost. Finally, the user should apply this configuration in the environment that they are running their applications on.

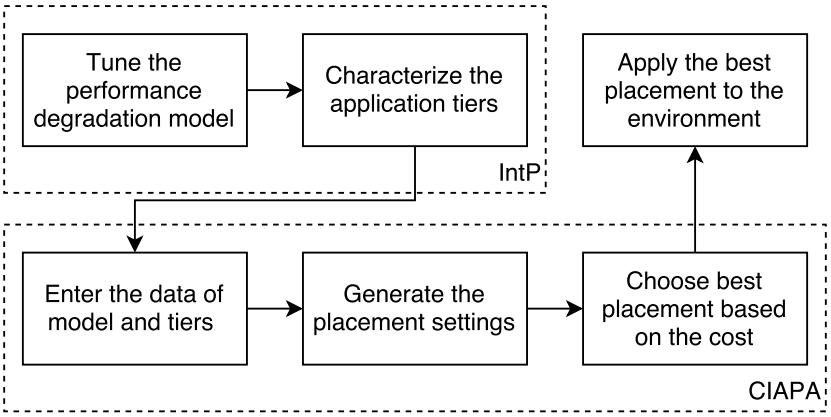


Figure 5.6: Placement decision workflow using CIAPA.

CIAPA works as an offline algorithm because it does not keep monitoring the status of the tiers or provide migration suggestions. Tiers need to be characterized in a given environment, and it is up to the user to give the real characterization of the application. For example, if a tier is characterized while having a low load, but actually, most of the time it will be running with a high load, the placement may fail at making the best decision. This happens because different workloads may result in different resource interference

and network affinity levels. Thus, if the application load changes, this placement workflow must be executed again in order to give the new best placement.

6. PERFORMANCE EVALUATION

In this chapter, we evaluate the performance of the implemented heuristics. Firstly, we analyze the performance of the optimization-based heuristics. Secondly, we analyze the performance of the threshold-based heuristics. Thirdly, we compare the performance of CIAPA with other placement strategies from the literature. Finally, we analyze the performance achieved by executing multi-tier applications under different placement settings.

In order to start, we first define two cases that we use along this chapter.

- **Case I**: We defined a set of 50 tiers with a mixed distribution of resources used. The tiers were divided into five groups, where each group used the following resources intensively: CPU, disk I/O, memory, cache, and mixed. Moreover, three tiers of each group had network affinity with other tiers in the same group. Other two tiers also had network affinity, but with tiers from other groups. Furthermore, the size of the tiers was defined sequentially, in increasing order. This case aims at simulating a more controlled environment, where there is a more equal distribution of resources utilized. For reference, this set of tiers is available in the online web interface[1].

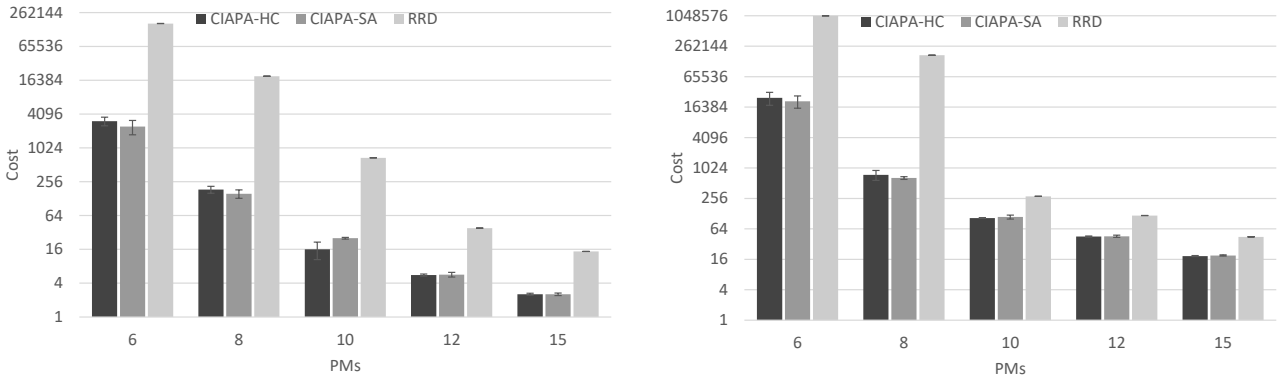
- **Case II**: We defined a set of 50 tiers with random distribution of resources used. Also, half of these tiers have network affinity with other tiers. This case aims at simulating what happens in cloud providers, where the applications that are deployed have unpredictable resource interference and network affinity levels. Moreover, tiers in this case have interference in multiple resources. For reference, this set of tiers is available in the online web interface[2].

6.1 Optimization-Based Heuristics

Figure 6.1 shows the cost comparison between the hill climbing (CIAPA-HC), simulated annealing (CIAPA-SA), and round robin decreasing (RRD), where the data is shown in logarithm scale. Here, we want to do two analysis: which heuristic implemented for CIAPA performs the best and how the CIAPA heuristics perform compared to the round robin approach. Moreover, due to the nondeterminism of the heuristics, the same placement configuration is not achieved in every execution. For this reason, all costs are the average of five executions of the placement algorithm, and the error bars are the standard deviation of each case.

[1] Set of tiers available at https://uillianluiz.github.io/ciapa/tiers/1
[2] Set of tiers available at https://uillianluiz.github.io/ciapa/tiers/2

Figure 6.1a shows the results for the case I, while 6.1b shows the results for the case II. In both cases, it is noticeable that hill climbing and simulated annealing generate placement settings with very similar costs. Simulated annealing had a lower cost for more extreme cases, where there are fewer PMs available. While for the other cases, it did not vary significantly, but in the case with 10 PMs hill climbing achieved a lower cost in both cases. Moreover, given the heuristics definition, simulated annealing should return the minimum global cost, while hill climbing should return only the minimum local. However, given the random generation of states, this behavior does not happen, and both algorithms perform very similarly. Even further, simulated annealing sometimes returns worse results due to the fact that it tries to do a broader exploration, and it spends a great deal of time on higher cost states.



(a) Case I (b) Case II

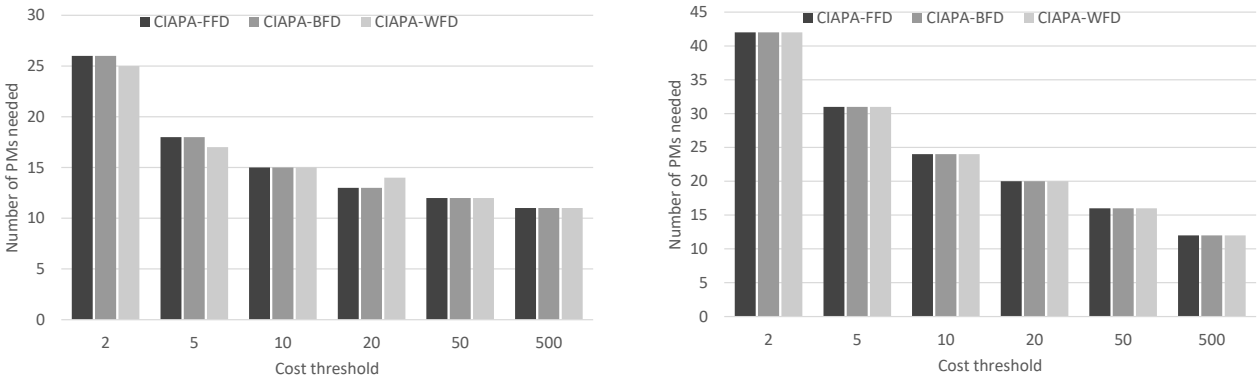
Figure 6.1: Cost comparison between hill climbing, simulated annealing, and round robin.

The other analysis from these charts is to verify how CIAPA optimization-based heuristics perform compared to the RRD approach. The RRD is a commonly used placement algorithm, which tries to evenly distribute the tiers per PMs, generating a balanced load per machines. Moreover, it is visible how CIAPA heuristics generate placements with lower cost. Also, it is possible to notice how the cost difference is much higher when there are fewer physical machines. In such cases, CIAPA is able to generate a placement with cost up to 100 times lower.

6.2 Threshold-based Heuristics

Figure 6.2 shows a comparison of the threshold-based heuristics for both cases. For these heuristics, instead of giving the exact environment, it uses the heuristics to find the minimum number of PMs necessary to host all tiers meeting the desired cost threshold. In the results, we notice that lower cost thresholds lead to higher number of PMs. This is a expected behavior since interference plays a major role in the cost. Moreover, in case II,

it is noticeable how it requires more PMs to meet the same cost threshold. This is due to the fact that the tiers in case II have higher interference levels as compared to case I.



(a) Case I (b) Case II

Figure 6.2: Number of PMs needed to meet the cost threshold using the threshold-based heuristics.

The number of PMs needed by the different heuristics did not differ considerably. They were identical for almost all cases, where only in the case I, the WFD heuristic had a slight difference of one PM. Moreover, another factor that should be noticed is that in these heuristics the cost is not close to the optimal. In case I, the optimization-based heuristics achieve costs lower than 10 using 12 PMs, while with the threshold-based ones, it is necessary a minimum of 15 PMs. In case II, the optimization-based heuristics achieve costs lower than 50 with 12 PMs, while with the threshold-based ones, it is necessary a minimum of 16 PMs.

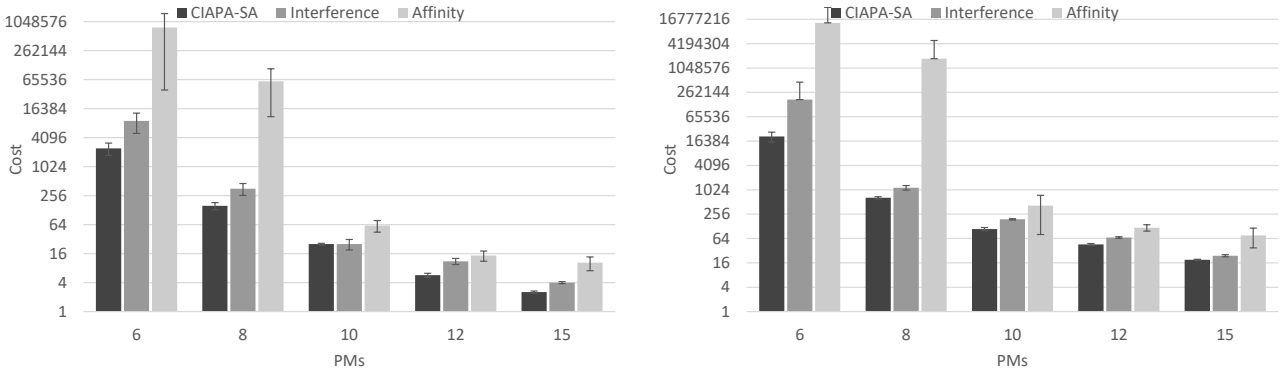
These results show how the optimization-based heuristics are able to provide results with lower costs as compared to the threshold-based ones. Nevertheless, the threshold-based heuristics are still able to generate results that are better than the commonly used RRD. Moreover, the execution time of these heuristics is much faster, where while they take about 3 ms in both case I and case II, the threshold-based heuristics take around 14 seconds using Google Chrome 61 and 18 seconds using Firefox Quantum 57. Furthermore, we have done some experiments executing the optimization-based heuristics in the back-end, using Node.js 9.4, and the execution time reduced to 8 seconds.

These time-wise performance metrics lead to two conclusions. Firstly, since the algorithms were implemented using JavaScript, there is still room for performance improvement by using a more powerful language. Also, JavaScript is a single-threaded language; therefore, other languages may be able to take advantage of the multi-threading to improve even further the execution time. Secondly, if the placement decision must be real time, the threshold-based heuristics are already a good option, where they generate improved placement costs in a timely manner. However, the main goal of CIAPA is to work as a planning tool, so it being in real time should not be a requirement; thus, the optimization-based heuristics can be used.

6.3 Comparison of CIAPA against Interference and Affinity Aware Only Algorithms

In this experiment, we verify how the optimization-based heuristics of CIAPA perform compared to the group of algorithms found in the literature. As we have seen in the related work, most algorithms in the literature consider only one criterion at the same time, while CIAPA considers both. Here, given the similarity of results between hill climbing and simulated annealing, we choose the costs given by simulated annealing due to its slight better results. Moreover, we selected the algorithm implemented by Somani et al. to represent the interference-aware [SKP12], and the algorithm implemented by Su et al. to represent the affinity-aware [SXC+15]. We did not fully implement the algorithms as they are presented by the authors, but we implemented them using the ideas that the authors presented.

Figure 6.3 shows the cost comparison between the strategies for both cases being studied. It is noticeable that CIAPA-SA generates placement configurations with lower cost as compared to both other strategies. This difference is higher whenever there are fewer PMs available, and it decreases when more PMs are used. Moreover, the interference-aware placement resulted in better cost results as compared to the affinity-aware. This is due to the fact that interference has a higher impact on the cost function. However, since CIAPA considers both criteria, it can reduce the cost even further, resulting in a placement that should give the ideal optimal performance.



(a) Case I (b) Case II

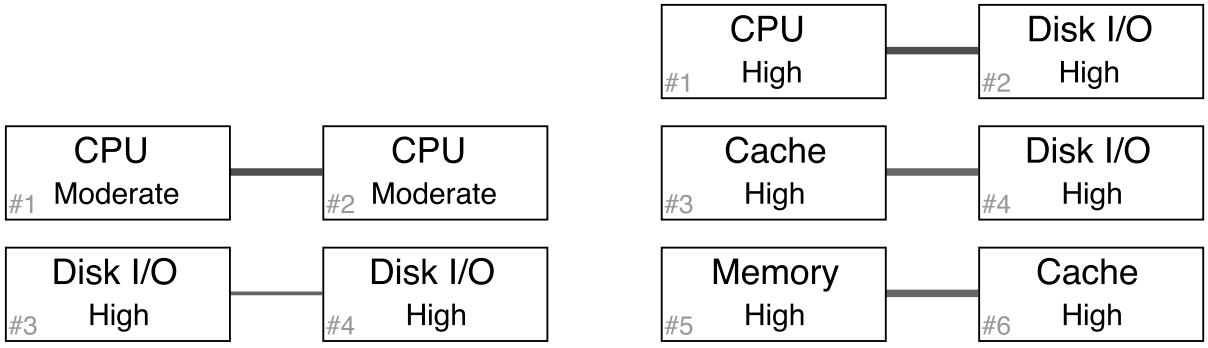
Figure 6.3: Cost comparison of CIAPA-SA vs. interference vs. affinity-aware strategies.

6.4 Use Case Analysis

In the experiments until now, we were only concerned about the placement costs. Since these costs were taken by the performance degradation model, it should give a good

indicator of the actual application's performance. However, in order to validate if the cost is actually correlated with the performance, we have executed a couple cases against different placement strategies. Given the fact that cases I and II have 50 tiers, we are not able to deploy it in our environment. Therefore, we created other two cases, which are described below.

- **Case III**: We created two multi-tier applications with high conflict between resource interference and network affinity. The first application has two CPU moderate-intensive tiers, with a high network affinity, while the second has two disk I/O high-intensive tiers with low network affinity. A diagram that represents this case is seen in Figure 6.4a. For reference, this set of tiers is available in the online web interface[3].
- **Case IV**: We created three multi-tier application with less conflict in the same application, but where there is still high affinity levels, and resource interference between tiers of different applications. A diagram that represents this case is seen in Figure 6.4b. For reference, this set of tiers is available in the online web interface[4]



(a) Case III (b) Case IV

Figure 6.4: Characteristics of the multi-tier applications used in the use case analysis.

We executed the placement algorithms for both cases, comparing CIAPA with the interference and affinity-aware strategies. Table 6.1 shows the placement distribution of tiers per PM for the case III. As expected, the interference strategy placed tiers with interference in different PMs, while the affinity strategy placed tiers with affinity in the same PM. CIAPA, on the other hand, generate a different placement, where three tiers were placed in one PM. This happens because CIAPA looks for the best trade-off between interference and affinity, generating the placement with the lowest cost.

In case IV, we have a similar behavior, where the strategies that only look at one criterion generate a placement with higher cost. For the interference-aware, tier #3 has performance degradation because it needs to communicate with tier #4, but it was placed in other PM. For the affinity-aware, tiers #2 and #4 have disk I/O interference; therefore,

[3]Set of tiers available at https://uillianluiz.github.io/ciapa/tiers/3
[4]Set of tiers available at https://uillianluiz.github.io/ciapa/tiers/4

Table 6.1: Distribution of tiers per PM using each placement strategy for case III.

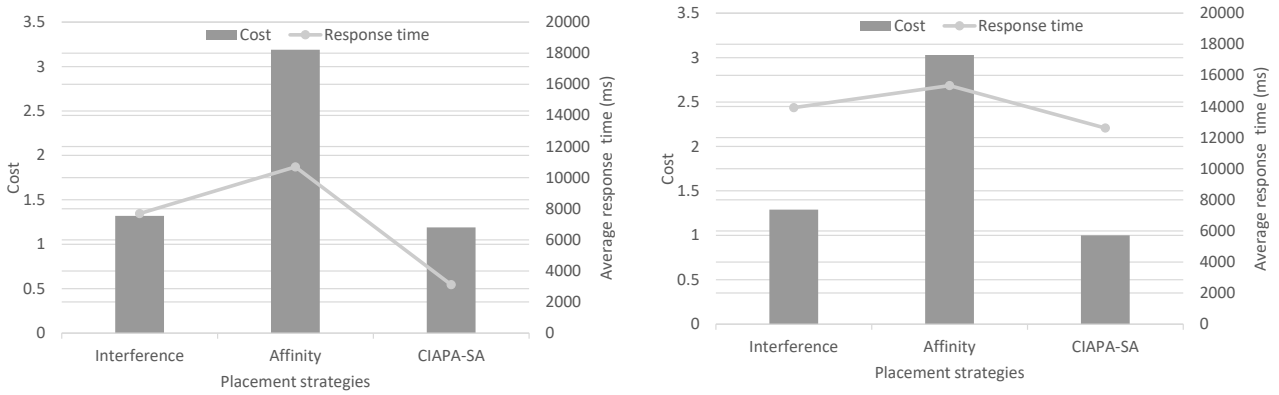
Placement	PM1	PM2
Interference	#1, #3	#2, #4
Affinity	#1, #2	#3, #4
CIAPA-SA	#1, #2, #4	#3

they have a high performance degradation for being placed in the same PM. On the other hand, CIAPA-SA is able to handle both cases, and for this case, it is able to generate a placement with no performance degradation.

Table 6.2: Distribution of tiers per PM using each placement strategy for case IV.

Placement	PM1	PM2
Interference	#1, #2, **#3**	#4, #5, #6
Affinity	#1, **#2**, #3, **#4**	#5, #6
CIAPA-SA	#1, #2, #5, #6	#3, #4

Using these placement distributions, we executed the application in our environment. Figure 6.5 shows the cost generated by the placement as well as the average response time achieved while running under the given placement setting for both cases. Under these two cases, we can see that CIAPA was not only able to generate a placement with lower cost, but this cost also led to the best overall application's performance.



(a) Case III

(b) Case IV

Figure 6.5: Cost and average response time comparison of different placement strategies.

7. CONCLUSION

In this master's thesis, we explored the problem of multi-tier applications placement in consolidated environments, focusing on resource interference and network affinity. While there were several research works focusing on such topics, there was still the necessity to look at both characteristics at the same time. For this reason, we modeled and implemented CIAPA, which is a set of Capacity, Interference and Affinity aware Placement Algorithms. Moreover, the main findings of this master's thesis are:

- Multi-tier applications have a different behavior as compared to regular applications. Each tier of a multi-tier application executes a small task, which, usually, takes less than a second to finish. For this reason, the high resource utilization comes from the high request rates.

  This facts raises an important concern regarding to the workload. Since these applications have a seasonal workload, i.e. varies with time, it is important to characterize them using the appropriate workload so that the interference and affinity levels collected match the actual behavior.

- The related work and the experiments have shown that resource interference and network affinity are characteristics that have high impact on applications performance. Also, different placement settings have a great impact on the applications' performance.

- It is not possible to eliminate the performance degradation given the conflicts between interference and affinity. For this reason, it is necessary to find the best trade-off, which leads to the best performance available.

In addition to these findings, we have generate several contributions, which are listed below.

- We created an open-source multi-tier benchmark called Node-tiers. This benchmark was very helpful to characterize both resource interference and network affinity of multi-tier applications. Moreover, it is widely extensible, and more features can be easily added. Therefore, we expect that it can be useful for other researches that need to do experiments with multi-tier applications.

- We described a set of placement policies that are able to aid administrators to make placement decisions. Even though we concluded that they were still a weak solution, they were able to summarize the experiments that we done using CPU and disk I/O intensive applications.

- We demonstrated the creation of a performance degradation model, which is able to describe the performance loss given by resource interference and network bottlenecks. This model is the alternative to use only interference levels, given the fact that some resources are different affected by interference.
- We created a set of costs functions that return the expected cost of placing a given set of tiers in the same PM. Moreover, even though this cost cannot be directly converted to performance, it has a good correlation, and high cost values lead to lower performance.
- We created CIAPA, which is a set of placement algorithms that aim at generating placements that lead to higher performance. These algorithms were divided into two categories, and through experiments, we demonstrate that they are able to provide better results than other strategies found in the literature. Moreover, we created a visualization tool, which provides a way to execute the algorithms easily.

We understand that for the purpose of this master's thesis, the objectives were achieved; however, some areas still need improvement. For this reason, for future works, we have planned a few improvements. Firstly, we should look into how to improve the placement heuristics. Currently, they might not return the best cost every time, so we should look at how to improve the optimization-based heuristics so that they can have a more deterministic approach. Secondly, we should optimize CIAPA for migration purposes. For this, we would analyze the current placement distribution, and find how to reduce the cost by doing the minimum changes possible. Thirdly, we need to better characterize the interference under different workload variations. For this, we would have to apply a dynamic placement instead of the current which is static. Finally, we are going to look into how to integrate CIAPA with the environment, collecting metrics, generating placements, and applying it to the infrastructure.

REFERENCES

[BRX13] Bu, X.; Rao, J.; Xu, C.-z. “Interference and locality-aware task scheduling for mapreduce applications in virtual clusters”. In: Proceedings of the 22nd international symposium on High-performance parallel and distributed computing, 2013, pp. 227–238.

[CH11] Chiang, R. C.; Huang, H. H. “Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments”. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011, pp. 47.

[CHRD03] Christoph, H.; Heinz, S.; Ralf-Detlef, K. “The Distributed Architecture of Multi- Tiered Enterprise Applications”. In: *A Middleware Architecture for Transactional, Object-Oriented Applications*, Berlin: FREIEN UNIVERSITÄT BERLIN, 2003, chap. 3, pp. 15–40.

[CQL14] Chi, R.; Qian, Z.; Lu, S. “Be a good neighbour: Characterizing performance interference of virtual machines under xen virtualization environments”. In: 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014, pp. 257–264.

[CSG11] Caglar, F.; Shekhar, S.; Gokhale, A. “Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud”, *3rd IEEE International Workshop on Real-time and distributed computing in emerging applications*, 2011.

[Dat17] Data Abstract. “Why multi-tier?” Source: https://docs.dataabstract.com/Introduction/WhyMultiTier/, accessed: 2017-05-24, 2017.

[DRK14] Dua, R.; Raja, A. R.; Kakadia, D. “Virtualization vs containerization to support paas”. In: 2014 IEEE International Conference on Cloud Engineering, 2014, pp. 610–614.

[FMCB17] Ferdaus, M. H.; Murshed, M.; Calheiros, R. N.; Buyya, R. “An Algorithm for Network and Data-aware Placement of Multi-Tier Applications in Cloud Data Centers”, *Journal of Network and Computer Applications (JNCA)*, 2017.

[HHM04] Huang, D.; He, B.; Miao, C. “A Survey of Adaptive Resource Management in Multi-Tier Web Applications”, *IEEE Communications Surveys & Tutorials*, vol. 16–3, 2004, pp. 1574–1590.

[IHJ11] Ibrahim, S.; He, B.; Jin, H. “Towards pay-as-you-consume cloud computing”, *Proceedings - 2011 IEEE International Conference on Services Computing, SCC 2011*, 2011, pp. 370–377.

[JF16] Jersak, L. C.; Ferreto, T. “Performance-aware server consolidation with adjustable interference levels”. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 420–425.

[JQWG15] Jin, H.; Qin, H.; Wu, S.; Guo, X. “Ccap: a cache contention-aware virtual machine placement approach for hpc cloud”, *International Journal of Parallel Programming*, vol. 43–3, 2015, pp. 403–420.

[KGV83] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. “Optimization by simulated annealing”, *science*, vol. 220–4598, 1983, pp. 671–680.

[Kin17] King, C. I. “Stress-ng”. Source: http://kernel.ubuntu.com/~cking/stress-ng/, accessed: 2017-11-10, 2017.

[LC12] Lin, J.-W.; Chen, C.-H. “Interference-aware virtual machine placement in cloud computing systems”. In: Computer & Information Science (ICCIS), 2012 International Conference on, 2012, pp. 598–603.

[LKCDR17] Ludwig, U. L.; Kirchoff, D. F.; Cezar, I. B.; De Rose, C. A. “Policies for interference and affinity-aware placement of multi-tier applications in private cloud infrastructures”, *XVIII Simpósio em Sistemas Computacionais de Alto Desempenho-WSCAD*, 2017.

[Mat14] Mateo, C. “Performance of several languages”. Source: http://blog.carlesmateo.com/2014/10/13/performance-of-several-languages/, accessed: 2017-12-29, 2014.

[MG11] Mell, P. M.; Grance, T. “Sp 800-145. the nist definition of cloud computing”, Technical Report, Gaithersburg, MD, United States, 2011.

[MNA16] Masdari, M.; Nabavi, S. S.; Ahmadi, V. “An overview of virtual machine placement schemes in cloud computing”, *Journal of Network and Computer Applications*, vol. 66, 2016, pp. 106–127.

[MYXW13] Moreno, I. S.; Yang, R.; Xu, J.; Wo, T. “Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement”. In: Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on, 2013, pp. 1–8.

[OW201] OW2 Consortium. “RUBiS”. Source: http://rubis.ow2.org/, accessed: 2017-02-24, 2001.

[SG06] Selman, B.; Gomes, C. P. “Hill-climbing search”, *Encyclopedia of Cognitive Science*, 2006.

[SGRC10] Sonnek, J.; Greensky, J.; Reutiman, R.; Chandra, A. “Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration”. In: Parallel Processing (ICPP), 2010 39th International Conference on, 2010, pp. 228–237.

[Sho17] Shoreditch Ops Ltd. “Artillery”. Source: https://artillery.io/, accessed: 2017-06-05, 2017.

[SKP12] Somani, G.; Khandelwal, P.; Phatnani, K. “Vupic: Virtual machine usage based placement in iaas cloud”, *arXiv preprint arXiv:1212.0085*, 2012.

[SXC+15] Su, K.; Xu, L.; Chen, C.; Chen, W.; Wang, Z. “Affinity and conflict-aware placement of virtual machines in heterogeneous data centers”. In: Autonomous Decentralized Systems (ISADS), 2015 IEEE Twelfth International Symposium on, 2015, pp. 289–294.

[US16] Usmani, Z.; Singh, S. “A Survey of Virtual Machine Placement Techniques in a Cloud Data Center”, *Procedia Computer Science*, vol. 78, 2016, pp. 491–498.

[Xav18] Xavier, M. G. “Intp: Quantifying cross-application interference in smp machines via resource-driven instrumentation”, Ph.D. Thesis, Pontifical Catholic University of Rio Grande do Sul, 2018.

[XDOR+15] Xavier, M. G.; De Oliveira, I. C.; Rossi, F. D.; Dos Passos, R. D.; Matteussi, K. J.; De Rose, C. A. F. “A performance isolation analysis of disk-intensive workloads on container-based clouds”, *Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015*, 2015, pp. 253–260.

[XLL+14] Xu, F.; Liu, F.; Liu, L.; Jin, H.; Li, B.; Li, B. “iaware: Making live migration of virtual machines interference-aware in the cloud”, *IEEE Transactions on Computers*, vol. 63–12, 2014, pp. 3012–3025.

[YDZ16] Yang, L.; Dai, Y.; Zhang, B. “MapReduce scheduler by characterizing performance interference”, 2016, pp. 253–262.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br